Karl Fogel

Tvorba open source softwaru

Jak řídit úspěšný projekt svobodného softwaru

Edice CZ.NIC

Karl Fogel

TVORBA OPEN SOURCE SOFTWARU
Jak řídit úspěšný projekt svobodného softwaru

Vydavatel:
CZ.NIC, z. s. p. o.
Americká 23, 120 00 Praha 2
Edice CZ.NIC
www.nic.cz

1. vydání, Praha 2012
Kniha vyšla jako 5. publikace v Edici CZ.NIC.
ISBN 9978-80-904248-5-2

© 2005–2010 Karl Fogel
V licenci CreativeCommons Attribution-ShareAlike (3.0).

CZ.NIC | SPRÁVCE DOMÉNY CZ

Tato kniha je věnována dvěma drahým přátelům, bez nichž by nemohla vzniknout:
Karen Underhillové a Jimovi Blandymu.

ISBN 9978-80-904248-5-2

— Karl Fogel

Tvorba open source softwaru

Jak řídit úspěšný projekt svobodného softwaru

— Edice CZ.NIC

Předmluva vydavatele

Vážení čtenáři,

naše předchozí publikace v Edici CZ.NIC byly orientovány na nějakou konkrétní technologii, od technologie IPv6 přes programování v Pythonu po používání elektronických podpisu. Touto knihou jsme se vydali do trochu jiné oblasti. Přeložili jsme pro vás knihu, která se zabývá tím, jak produkovat otevřený (open-source) software. Kniha pro vás může být zajímavá ať už jste začínající programátor s dobrým nápadem nebo zavedená firma produkující již existující software.

V Edici CZ.NIC vycházejí především knihy, které jsou blízké vám ale i nám, a není tomu jinak ani v případě této knihy. Centrální registr pro správu domén FRED byl od počátku vydáván pod svobodnou licencí a díky tomu je používán i na dalších místech naší zeměkoule. Mezi poslední přírůstky mezi národními registry používající FRED můžeme zmínit například Estonsko. Stejně se chováme i k dalšímu softwaru, který je vyvíjen ve sdružení CZ.NIC – alternativní klient pro přístup do datových schránek – Datovka, velmi úspěšný projekt směrovacího daemona Bird, nebo náš poslední projekt rychlého autoritativního DNS serveru – Knot DNS.

Autor této publikace Karl Fogel není ve světě otevřeného a svobodného softwaru žádným nováčkem. Kromě psaní o otevřeném softwaru také přispívá do významných projektů jako je Launchpad, Subversion nebo GNU Emacs.

Doufáme, že tato kniha bude přínosná pro Vás a přeneseně i pro celou komunitu okolo otevřeného softwaru, protože po přečtení této knihy bude Váš další projekt pod otevřenou licencí. Přeji Vám příjemnou četbu.

Ondřej Surý
Praha, 17. ledna 2012

Obsah

Přehled kapitol

Předmluva vydavatele — 5

Předmluva — 19
Proč byla tato kniha napsána? — 21
Kdo by měl tuto knihu číst? — 22
Zdroje — 22
Poděkování — 23
Prohlášení — 25

1. — Úvod — 27
2. — Zahájení projektu — 43
3. — Technická infrastruktura — 71
4. — Společenská a politická infrastruktura — 113
5. — Peníze — 127
6. — Komunikace — 149
7. — Vytváření balíčků, vydávání releasů a každodenní práce na vývoji — 189
8. — Řízení dobrovolníků — 219
9. — Licence, autorská práva a patenty — 255

Přílohy

A. — Svobodné systémy pro správu verzí — 275
B. — Volně přístupné bug trackery (záznamníky chyb) — 283
C. — Měl bych se starat o to, jakou barvu má přístřešek pro kola? — 289
D. — Vzorové pokyny pro hlášení chyb (bugů) — 297
E. — Copyright — 303

Předmluva — 19
Proč byla tato kniha napsána? — 21
Kdo by měl tuto knihu číst? — 22
Zdroje — 22
Poděkování — 23
Prohlášení — 25

1. — Úvod — 29

Historie — 32
Vzestup proprietárního softwaru a svobodného softwaru — **32**
Vědomé hnutí odporu — **33**
Neúmyslné hnutí odporu — **36**
„Free“ versus „open source“ — **37**

Současná situace — 40

2. — Zahájení projektu — 45
Nejdříve se ale rozhlédněte — **47**

Začněte od toho, co máte k dispozici — 47
Vyberte dobré jméno — **48**
Určete jasné cíle projektu — **49**
Uveďte, že jde o svobodný projekt — **50**
Seznam funkcí a požadavků — **51**
Stav vývoje — **51**
Stahování — **52**
Zpřístupnění systémů správy verzí a sledování chyb — **53**
Komunikační kanály — **54**
Pokyny pro vývojáře — **55**
Dokumentace — **55**
Dostupnost dokumentace — **57**
Dokumentace pro vývojáře — **58**
Vzorový výstup a snímky obrazovky — **58**
Kompletní hosting — **59**

Výběr licence a její uplatnění — 59
Licence typu „vše je povoleno“ — **60**
GPL — **60**
Jak licenci aplikovat — **60**

Udávání tónu — 61
Vyhněte se soukromým diskuzím — **62**
Nezdvořilé chování potlačte hned v zárodku — **64**
Kontrolujte kód veřejně — **65**
Když otvíráte dosud uzavřený projekt, uvědomte si rozsah změn — **67**

Oznamování — 68

3. — Technická infrastruktura — 73

Co je třeba pro projekt zajistit — 74

Mailing listy — 75
Ochrana před nežádoucími zprávami — **77**
Filtrování příspěvků — **77**
Skrývání adres v archivech — **79**
Identifikace a správa hlaviček — **80**
Velká debata o Reply-to — **82**
Dvě ideální řešení — **84**
Archivace — **85**
Software — **86**

Správa verzí — 87
Slovníček pojmů správy verzí — **87**
Výběr systému pro správu verzí — **91**
Používání systému pro správu verzí — **92**
Sledujte verze u všeho — **92**
Možnost prohlížení — **93**
E-maily oznamující commit — **93**
Používejte větve, abyste nezpomalovali vývoj — **95**
Jedinečnost informace — **96**
Autorizace — **97**

Systém pro sledování chyb — 99
Interakce s mailing listy — **102**
Předběžné filtrování v bug trackeru — **103**

IRC a jiné systémy pro diskuzi v reálném čase — 104
Roboti — **106**
Archivace IRC — **107**

RSS — 107
Wiki — 108

Webové stránky — 110
Kompletní hosting — **110**
Jak si vybrat kompletní hosting — **111**
Anonymita a zapojení se do projektu — **112**

4. — Společenská a politická infrastruktura — 115
Schopnost vytvářet odnože — **115**

Benevolentní diktátoři — 116
Kdo může být dobrým benevolentním diktátorem? — **117**

Demokracie založená na konsenzu — 118
Řízení verzí znamená, že můžete být v pohodě — **119**
Když nelze dosáhnout konsenzu, hlasujte — **119**
Kdy hlasovat — **120**
Kdo hlasuje? — **121**
Ankety versus hlasování — **122**
Veto — **123**

Jak to všechno zapsat — 123

5. — Peníze — 129

Typy zapojení do projektu — 130
Pracovníky najímejte na dlouhodobý úvazek — 132
Vystupujte jako jednotlivci, nikoli jako celek — 133
Otevřeně hovořte o své motivaci — 134
Lásku si za peníze nekoupíte — 136

Dodavatelské smlouvy — 137
Kontrola a přijetí změn — **140**
Případová studie: protokol pro ověřování hesla CVS — **140**

Financování neprogramovacích činností — 141
Zajišťování kvality (tj. Profesionální testování) — **141**
Právní poradenství a ochrana — **143**
Dokumentace a použitelnost — **143**
Poskytování hostingu/síťové propustnosti — **144**

Marketing — 145
Pamatujte na to, že jste sledováni — **145**
Nekritizujte konkurenční open source produkty — **147**

6. — Komunikace — 151

Jste to, co píšete — 152
Struktura a formátování — **152**
Obsah — **154**
Tón — **155**
Jak rozeznat hrubost — **156**
Tvář — **158**

Jak předejít častým potížím — 160
Nepřispívejte zbytečně — **160**
Produktivní a neproduktivní vlákna — **161**
Čím jednodušší téma, tím delší debata — **163**
Vyvarujte se svatých válek — **164**
Efekt „hlučné minority“ — **166**

Obtížní lidé — 166
Jak se s obtížnými lidmi vypořádat — **167**
Případová studie — **168**

Jak se vyrovnat s růstem — 170
Nápadné využívání archivů — **171**
Se všemi zdroji zacházejte jako s archivy — **173**
Kodifikace tradic — **174**

Nediskutujte v bug trackeru — 177
Publicita — **179**
Ohlášení bezpečnostních chyb — **181**
Přijměte report — **181**
Opravu vyvíjejte potichu — **182**
Čísla CAN/CVE — **183**
Předběžné oznámení — **184**
Distribuujte opravu veřejně — **186**

7. — Vytváření balíčků, vydávání releasů a každodenní práce na vývoji — 191

Číslování verzí — 192
Z čeho se číslo verze skládá — **193**
Jednoduchá strategie — **195**
Strategie sudá / lichá — **196**

Release větve — 197
Jak release větve fungují — **198**

Stabilizace release — 199
Diktatura vlastníka release — **200**
Hlasování o změnách — **201**
Spolupráce na stabilizaci release a jak ji řídit — **202**
Správce release — **203**

Vytváření balíčků — 204
Formát — **204**
Jméno balíčku a adresářový strom — **205**
Velká nebo malá písmena — **207**
Pre-release — **207**
Kompilace a instalace — **208**
Binární balíčky — **209**
Testování a releasing — **210**
Release kandidát — **211**
Oznamování release — **211**

Udržování více release řad — 212
Bezpečnostní release — **213**

Vydávání releasů a každodenní práce — 214
Plánování releasů — **215**

8. — Řízení dobrovolníků — 221

Jak dostat z dobrovolníků to nejlepší — 222
Delegování — **222**
Jasně rozlišujte mezi poptáváním a zadáváním — **223**
Postup po delegování — **224**
Všímejte si, o co se lidé zajímají — **225**

Pochvala a kritika — **225**
Zabraňte teritorialitě — **226**
Poměr automatizace — **229**
Automatizované testování — **229**
Ke všem uživatelům se chovejte jako k potenciálním dobrovolníkům — **231**

Podělte se o řídící i technické úkoly — 234
Patch Manager (manažer záplat) — **235**
Translation Manager (manažer překladu) — **236**
Documentation Manager (manažer dokumentace) — **238**
Issue manager (manažer problémů) — **239**
FAQ Manager (manažer sekce s nejčastěji kladenými otázkami) — **240**

Personální změny — 241

Committeři — 244
Volba committerů — **244**
Odebrání commit access — **245**
Částečný commit access — **246**
Nečinní committeři — **247**
Nedělejte s ničím tajnosti — **247**

Uvádění tvůrců a spoluautorů (Credit) — 248

Odnože (Forks) — 249
Zvládání odnoží — **250**
Iniciování odnože — **252**

9. — Licence, autorská práva a patenty — 257

Terminologie — 257
Aspekty licence — 260
GPL a kompatibilita licence — 262

Volba licence — 263
MIT / X Window System License — **263**
GNU General Public License — **264**
Je GPL svobodná licence, či nikoli? — **265**
A co licence BSD? — **266**

Převod a vlastnictví autorských práv — 267
Nedělat nic — **267**
Licenční smlouvu s přispěvatelem (CLA) — **268**
Převod autorských práv — **268**

Systém dvojitých licencí — 269
Patenty — 270
Další zdroje — 274

Přílohy

A. — Svobodné systémy pro správu verzí — 277
B. — Volně přístupné bug trackery (záznamníky chyb) — 285
C. — Měl bych se starat o to, jakou barvu má přístřešek pro kola? — 291
D. — Vzorové pokyny pro hlášení chyb (bugů) — 299
E. — Copyright — 305

Předmluva

Předmluva — 19

Proč byla tato kniha napsána? — 21
Kdo by měl tuto knihu číst? — 22
Zdroje — 22
Poděkování — 23
Prohlášení — 25

Proč byla tato kniha napsána?

Když dnes na nějakém večírku řeknu, že píšu svobodný software, už se nesetkávám jen s prázdnými pohledy. „Aha, open source – takže něco jako Linux?“ říkají. Horlivě přikyvuji. „Přesně tak. To je můj obor.“ Je příjemné, když už vaše práce není úplně neznámá. Dříve se dalo i docela dobře předvídat, jaká otázka bude následovat: „Jak se tím dají vydělat peníze?“ Na tohle se dá odpovědět jenom shrnutím celé ekonomické situace svobodného softwaru: že existují organizace, v jejichž zájmu je, aby určitý software existoval, ale které jej nepotřebují prodávat; chtějí mít pouze jistotu, že je tento software dostupný a udržovaný – jako nástroj a ne jako zboží.

V poslední době se to ale změnilo a další otázka se ne vždy týká peněz. Obchodní stránka open source softwaru[1] už není tolik záhadná a velká část neprogramátorské veřejnosti chápe nebo přinejmenším není překvapena tím, že je možné mít v tomto oboru zaměstnání na plný úvazek. Ta otázka, která následuje, poslední dobou stále častěji zní: *„To je zajímavé, ale jak to vlastně funguje?“*

A na to jsem žádnou uspokojivou odpověď dlouho neměl. Čím víc jsem se snažil nějakou najít, tím víc jsem zjišťoval, jak složité téma to vlastně je. Řízení projektu svobodného softwaru je něco trochu jiného než běžné podnikání – představte si, že byste se o podobě svého produktu museli neustále dohadovat se skupinou dobrovolníků, z nichž jste většinu nikdy osobně nepotkali! Z různých důvodů se to nepodobá ani vedení klasické neziskové organizace, ani řízení vlády. Všem těmto věcem se to sice v něčem podobá, ale časem jsem dospěl k závěru, že svobodný software je entitou *sui generis*. Najdete mnoho věcí, k nimž jej lze celkem výstižně přirovnat, ale nic, co by bylo stejné. Ostatně už samotné tvrzení, že by bylo možné projekt svobodného softwaru nějak „řídit“, je celkem odvážné. Projekt svobodného softwaru může být nepochybně spuštěn; může být také ovlivněn zainteresovanými stranami, často velmi silně. Ale výsledky jeho práce se nemohou stát vlastnictvím žádného jednotlivce, a dokud se někde – a to naprosto kdekoliv – najdou lidé, kteří mají zájem v něm pokračovat, nemůže jej ani nikdo zastavit. Všichni mají neomezenou moc a zároveň nikdo nemá žádnou.
A to vytváří zajímavou dynamiku.

Proto jsem se tedy rozhodl napsat tuto knihu. Projekty svobodného softwaru daly vzniknout jedinečné kultuře, étosu, v němž je hlavní zásadou svoboda psát software, který dělá, cokoliv budete chtít, ale kde tato zásada nevede k tomu, že by si každý psal svůj vlastní kód zcela nezávisle na ostatních, ale k nadšené spolupráci. Ostatně schopnost dobře spolupracovat s ostatními je v oblasti svobodného softwaru jednou z dovedností, které se cení nejvíc. K řízení takovýchto projektů je nutné zapojit se do jakési hypertrofní formy spolupráce, při níž může softwaru výrazně prospět jak schopnost pracovat s ostatními, tak i objevování nových způsobů takové spolupráce. Tato kniha se pokouší popsat techniky, jimiž toho lze dosáhnout. Neklade si ambice celé téma vyčerpat, ale přinejmenším je to dobrý začátek.

---

[1] Termíny „open source“ a „svobodný“ (free) software jsou v tomto kontextu v podstatě synonymní. Podrobněji budou probírány v části **„Free“ versus „open source“** v kapitole **1. Úvod.**

Dobrý svobodný software je sám o sobě hodnotným cílem a já doufám, že čtenáři, kteří hledají způsob, jak jej dosáhnout, budou s obsahem této knihy spokojeni. Ale kromě toho také doufám, že se mi podaří předat něco z té čiré radosti, jež vzniká ze spolupráce v motivovaném týmu open source vývojářů a ze sympaticky přímého kontaktu s uživateli, který se v open source projektech často objevuje. Účast na úspěšném projektu svobodného softwaru je totiž velká zábava, což je koneckonců ten hlavní důvod, proč celý systém může fungovat.

Kdo by měl tuto knihu číst?

Tato kniha je určena vývojářům softwaru a těm, kdo softwarové projekty řídí, a to ať už v situaci, kdy o spuštění projektu svobodného softwaru teprve uvažují, nebo když už nějak začali a teď si nejsou jisti, jak dál. Měla by pomoci také lidem, kteří by se do nějakého open source projektu chtěli zapojit, ale nikdy předtím nic takového nedělali.

Čtenář této knihy nemusí být nutně programátor, ale měl by znát základní koncepty softwarového inženýrství, jako je zdrojový kód, kompilátory a záplaty (patche).

Není potřeba ani mít předchozí zkušenost s open source softwarem, a to ani jako uživatel, ani jako vývojář. Těm, kteří již v projektech svobodného softwaru někdy pracovali, budou některé části knihy připadat celkem samozřejmé, takže je možná budou chtít přeskočit. Protože jsem se snažil psát pro potenciálně velmi široké spektrum čtenářů, dal jsem všem podkapitolám srozumitelné nadpisy a vždy zdůraznil, které části mohou ti zkušenější bez obav vynechat.

Zdroje

Mnoho zdrojového materiálu pro tuto knihu pochází z pěti let práce na projektu Subversion (**http://subversion.tigris.org/**). Subversion je open source systém pro správu a verzování zdrojových kódů, který byl napsán zcela od nuly; byl zamýšlen jako náhrada pro CVS, jehož použití bylo v té době v celé open source komunitě *de facto* standardem. Projekt byl zahájen na začátku roku 2000 mým zaměstnavatelem, společností CollabNet (**http://www.collab.net/**), která naštěstí už od začátku pochopila, že je třeba jej vést v duchu spolupráce a distribuované činnosti. Velmi záhy se k projektu začalo přidávat mnoho dobrovolníků; dnes se jej účastní kolem 50 vývojářů, z nichž jenom několik je zaměstnáno v CollabNet.

V mnoha ohledech je Subversion zcela klasickým příkladem open source projektu a nakonec jsem z něj čerpal víc, než jsem původně očekával. Do určité míry je to proto, že to pro mě bylo nejjednodušší – když jsem potřeboval příklad nějakého konkrétního jevu, obvykle se mi hned vybavil nějaký, který jsme zažili při práci na Subversion. Je to ale také zdroj pro porovnávání informací. Ačkoliv se v různé míře účastním i jiných projektů svobodného softwaru a hovořím s přáteli a známými, kteří jsou aktivní v mnoha dalších, při psaní textu určeného k vydání si člověk rychle uvědomí, že by si

všechna svá tvrzení měl ověřit. Nechtěl jsem se vyjadřovat k událostem, které se staly v jiných projektech, jen na základě toho, co jsem si mohl přečíst v archivech veřejných diskuzních skupin.Věděl jsem totiž, že pokud by se někdo o něco takového pokusil u Subversion, dospěl by ke správnému závěru jen asi v polovině případů, zatímco v té druhé by se mýlil. Takže když jsem čerpal inspiraci nebo příklady z projektu, s kterým jsem neměl přímou zkušenost, zkusil jsem nejdříve oslovit někoho, kdo je o něm lépe informovaný a komu můžu věřit, že mi řekne, jak to skutečně bylo.

Subversion byl mým zaměstnáním posledních pět let, ale svobodným softwarem se zabývám už delší dobu – dohromady 12 let. Mezi další projekty, které obsah této knihy ovlivnily, patří:

- Projekt textového editoru GNU Emacs nadace Free Software Foundation, ve kterém udržuji několik malých balíčků.
- Concurrent Versions System (CVS), na kterém jsem intenzivně pracoval v letech 1994–1995 s Jimem Blandym, ale do nějž jsem se od té doby zapojoval jen příležitostně.
- Sbírka open source projektů známá jako Apache Software Foundation, zejména Apache Portable Runtime (APR) a Apache HTTP Server.
- OpenOffice.org, Berkeley Database od Sleepycat a databáze MySQL – těchto projektů jsem se osobně neúčastnil, ale sledoval jsem je a v některých případech jsem hovořil s jejich tvůrci.
- GNU Debugger (GDB) (totéž).
- The Debian Project (totéž).

Tento seznam samozřejmě není úplný. Podobně jako většina open source programátorů i já zdálky sleduji mnoho různých projektů, abych měl přehled o celkovém stavu věcí. Nebudu je tady všechny vyjmenovávat, ale na příslušných místech se o nich v této knize zmiňuji.

Poděkování

Psaní této knihy zabralo čtyřikrát víc času, než jsem si původně myslel, a často jsem při tom měl pocit, jako by mi pořád viselo nad hlavou koncertní křídlo. Za to, že jsem ji dokázal dokončit a zachovat si zároveň své duševní zdraví, vděčím mnoha lidem.

Andy Oram z O'Reilly je typem redaktora, o němž všichni autoři knih sní. Kromě toho, že celou problematiku důvěrně zná (a navrhl mnohá z témat knihy), má i vzácný dar rozpoznat, co jste chtěli říci, a pomoci vám najít ten správný způsob, jak to říci. Práce s ním pro mě byla ctí. Děkuji také Chucku Toporkovi za to, že můj návrh rovnou přesměroval na Andyho.

Brian Fitzpatrick recenzoval téměř veškerý text knihy hned po jeho napsání, což nejen zvýšilo jeho kvalitu, ale také mě udrželo při psaní ve chvílích, kdy jsem chtěl být kdekoliv jinde, jen ne u počítače. Ben Collins-Sussman a Mike Pilato také dohlíželi, zda práce pokračují, a vždycky se mnou ochotně diskutovali – někdy docela dlouze – o jakémkoliv tématu, které jsem se v daném týdnu snažil popsat. Všímali si také, když jsem začínal zpomalovat, a když to bylo nutné, jemně mě postrkovali. Díky.

Biella Colemanová psala svou dizertační práci ve stejné době, kdy jsem dával tuto knihu dohromady i já. Ví, jaké to je každý den sednout a psát, a byla mi inspirujícím příkladem a chápavou posluchačkou. Na hnutí svobodného softwaru také aplikuje svůj fascinující pohled antropologa, čímž mi poskytla jak zajímavé nápady, tak i odkazy na další materiál, který jsem v knize mohl použít. Alex Golub, další antropolog, který je jednou nohou ve světě svobodného softwaru a jenž zároveň také dokončoval svou dizertační práci, mě v počátcích mimořádně podporoval, což mi hodně pomohlo.

Micah Anderson vždycky vypadal, že ho jeho vlastní psaní nijak zvlášť nezatěžuje, což pro mě bylo velkou inspirací (i když přiznávám, že jsem mu to spíš záviděl); hlavně byl ale vždy připraven poskytnout přátelskou radu nebo (přinejmenším při jedné příležitosti) technickou podporu. Díky, Micahu! Jon Trowbridge a Sander Striker mi dodali jak povzbuzení, tak i konkrétní pomoc. Jejich rozsáhlé zkušenosti v oblasti svobodného softwaru mi poskytly materiál, k němuž bych se jinak jen těžko dostal.

Děkuji Gregu Steinovi nejen za přátelství a za dobře načasovanou podporu, ale i za to, že v projektu Subversion ukázal, jak jsou pravidelné revize kódu důležité pro budování programátorské komunity. Děkuji také Brianu Behlendorfovi, který nám taktně vtloukal do hlav, jak důležité je vést diskuze na veřejnosti. Doufám, že se tento princip odráží v celé knize.

Děkuji také Benjaminu „Makovi“ Hillovi a Sethu Schoenovi za rozličné rozpravy o svobodném softwaru a o jeho politických aspektech. Děkuji Zackovi Urlockerovi a Louisi Suarez-Pottsovi za to, že si ve svých nabitých plánech našli čas na rozhovor se mnou. Děkuji Shaneovi ze skupiny Slashcode za to, že svolil k citaci svých příspěvků, a Haggenovi So za jeho nesmírně užitečné srovnání serverů nabízejících kompletní hosting projektů.

Děkuji Alle Dekhtyar, Polině a Soni za jejich neustálé a trpělivé povzbuzování. Jsem velmi rád, že už nebudu muset ukončovat (nebo se spíš neúspěšně pokoušet ukončit) naše společné večery předčasně, abych mohl jít domů a pracovat na „té knize“.

Děkuji Jackovi Repenningovi za přátelství, mnohé rozhovory a tvrdohlavé odmítání přijmout jednoduchou chybnou analýzu, pokud je k dispozici analýza obtížnější, ale správná. Doufám, že se alespoň něco z jeho dlouhodobých zkušeností s vývojem softwaru a se softwarovým průmyslem jako takovým na této knize nějak odrazilo.

Společnost CollabNet prokázala mimořádnou velkorysost, když mi při psaní povolila velmi pružný harmonogram a nestěžovala si, když to trvalo mnohem déle, než se původně plánovalo. Nevím, jak přesně vypadá proces, jímž vedení společnosti k takovým závěrům dospěje, ale řekl bych, že v tom měli prsty Sandhya Kluteová a později Mahesh Murthy – za což jim oběma děkuji.

Celý vývojový tým projektu Subversion mi byl v uplynulých pěti letech velkou inspirací a mnohé z toho, co jsem napsal v této knize, jsem se naučil právě při práci s ním. Nebudu jim zde děkovat jmenovitě, protože je jich příliš mnoho, ale vyzývám každého čtenáře, který někdy narazí na přispěvatele projektu Subversion, aby ho pozval na nápoj podle jeho vlastní volby. Já osobně to dělat určitě budu.

Rachel Scollonovou jsem mnohokrát otravoval nekonečnými stížnostmi na aktuální stav knihy. Vždycky mi ochotně naslouchala a nějak dokázala, že mi po našem rozhovoru všechny problémy rázem připadaly menší. Moc mi to pomohlo – díky.

Děkuji (opět) Noelu Taylorovi, který se určitě musel divit, proč chci psát další knihu, když jsem si u té minulé tolik stěžoval, ale jehož přátelství a vedení sboru Golos pro mě znamenalo, že ani v okamžicích největší pracovní vytíženosti se z mého života nevytratila hudba a dobré vztahy s lidmi. Děkuji také Matthewovi Deanovi a Dorothei Samtlebenové, přátelům a dlouhodobě trpícím hudebním partnerům, kteří projevili velké pochopení pro mé stále se hromadící výmluvy, že nebyl čas cvičit. Megan Jenningsová mě neustále podporovala a upřímně se zajímala o zpracovávané téma, ačkoliv pro ni bylo neznámé – což je pro nejistého autora velká posila. Díky, Megan!

Knihu přečetli čtyři zasvěcení a pilní recenzenti: Yoav Shapira, Andrew Stellman, Davanum Srinivas a Ben Hyde. Kdybych byl schopen do knihy začlenit všechny jejich skvělé návrhy, byla by lepší. Časová omezení mě donutila vybrat si jen některé, které ale přesto celému textu výrazně prospěly. Za všechny chyby, které v knize zůstaly, jsem zodpovědný pouze já.

Mí rodiče, Frances a Henry, pro mě jako vždy byli báječnou oporou, a protože tato kniha je méně technická než ta předchozí, doufám, že pro ně bude i čitelnější.

Nakonec bych rád poděkoval těm, kterým jsem knihu věnoval – Karen Underhillové a Jimovi Blandymu. Karenino přátelství a pochopení pro mě znamenalo vše – nejen během psaní této knihy, ale také během posledních sedmi let. Bez její pomoci bych ji nikdy nemohl dokončit. Podobné je to s Jimem, opravdovým přítelem a skvělým hackerem, který mě jako první učil o svobodném softwaru – asi tak, jako by pták učil letadlo létat.

Prohlášení

Myšlenky a názory vyjádřené v této knize jsou mé vlastní. Nemusí nezbytně reprezentovat pohled společnosti CollabNet nebo projektu Subversion.

1. Úvod

1. Úvod — 29

Historie — 32
Vzestup proprietárního softwaru
a svobodného softwaru — **32**
Vědomé hnutí odporu — **33**
Neúmyslné hnutí odporu — **36**
„Free“ versus „open source“ — **37**

Současná situace — 40

1. Úvod

Většina projektů svobodného softwaru ztroskotá.

O těchto selháních ale obvykle není moc slyšet. Jednak proto, že pozornost přitahují pouze ty úspěšné, a jednak proto, že projektů svobodného softwaru existuje tak velké množství[2], že i když uspěje jen malé procento z nich, pořád to bude vytvářet navenek zcela opačný dojem. O krachu projektů se obvykle také nedozvíme proto, že jejich selhání se nedá popsat jako událost. Nelze poukázat na konkrétní okamžik, kdy nějaký projekt ztratil svou životaschopnost – lidé se od něj zkrátka časem nějak odklonili a přestali na něm pracovat. Existuje sice moment, kdy byla do projektu zanesena poslední změna, ale ti, kteří ji provedli, obvykle ještě netušili, že dál už nic nebude. Neexistuje dokonce ani jednoznačná definice toho, kdy projekt zanikl. Je to tehdy, když se na něm aktivně nepracovalo po dobu šesti měsíců? Když jeho uživatelská základna přestala růst, aniž by převýšila počet vývojářů? Co když jej vývojáři opustili, protože zjistili, že existuje jiný projekt se stejným cílem? A co když se pak k tomuto druhému projektu připojili a rozšířili jej o mnohé z toho, co vytvořili dříve? Skončil vůbec ten první projekt, nebo se jen přestěhoval?

Z těchto a podobných důvodů není možné přesně zjistit, kolik procent projektů uspěje a kolik ne. Různé přibližné informace, které jsem nashromáždil za svých více než deset let v open source, posbíral na SourceForge.net a našel pomocí Googlu, naznačují všechny zhruba totéž: že počet těch, které selžou, je obrovský, pravděpodobně v řádu 90–95 %. Toto číslo pak ještě naroste, pokud započítáme i ty projekty, které sice dosud přežívají, ale už nefungují, tedy takové, které stále ještě produkují funkční software, ale jejichž pracovní prostředí je nepříjemné, nebo jež se nevyvíjejí tak rychle nebo tak spolehlivě, jak by mohly.

Tato kniha pojednává o tom, jak se selhání vyhnout. Nezkoumá jen to, jak dělat věci správně, ale také to, jak se dělají špatně, abyste mohli problémy včas rozpoznat a napravit. Doufám, že po jejím přečtení budete mít k dispozici řadu technik, které vám umožní nejen vyhnout se častým nástrahám vývoje open source softwaru, ale také vypořádat se s růstem a údržbou jakéhokoliv úspěšného projektu. Úspěch není hra s nulovým součtem a tato kniha se nezabývá tím, jak vyhrát nebo jak předběhnout vaši konkurenci. Koneckonců jednou z důležitých součástí vedení open source projektu je i dobrá spolupráce s ostatními souvisejícími projekty. Z dlouhodobé perspektivy přispívá každý úspěch svým dílem k prosperitě veškerého svobodného softwaru světa.

Bylo by lákavé říct, že projekty svobodného softwaru selhávají ze stejných důvodů jako projekty proprietárního softwaru. Nepochybně platí, že svobodný software nemá monopol na nerealistické požadavky, nejasné specifikace, špatné řízení zdrojů, nedostatky ve fázích návrhu i všechny ty ostatní strašáky, kteří jsou v softwarovém průmyslu již dobře známí. Těmito tématy se už zabývalo mnoho

[2] SourceForge.net, jeden z populárních hostingových serverů, měl v polovině dubna 2004 registrováno 79 225 projektů. Tohle číslo ale samozřejmě ani zdaleka neodpovídá celkovému počtu projektů svobodného softwaru na internetu – jsou v něm pouze ty, které si vybraly SourceForge.

knih, proto se jim zde pokusím vyhnout. Místo toho se budu snažit popsat problémy, které se objevují jen ve světě svobodného softwaru. Pokud takový projekt selže, je to často proto, že jeho vývojáři (nebo vedoucí) podcenili specifické problémy spojené s vývojem open source – třebaže na ty známější obtíže spojené s vývojem closed source mohli být dobře připraveni.

Jednou z nejběžnějších chyb jsou nerealistická očekávání výhod open source přístupu jako takového. Otevřená licence nezaručí, že se na váš projekt hned vrhnou davy aktivních vývojářů a začnou mu dobrovolně věnovat svůj čas; to, že otevřete zdrojový kód problematického projektu, neznamená automatické vyřešení všech jeho potíží. Ve skutečnosti je to přesně naopak – otevřením projektu vytváříte celou řadu nových problémů a v krátkodobém horizontu vás to může stát více, než kdyby projekt zůstal uzavřený. Otevření projektu znamená, že je kód potřeba uspořádat tak, aby byl srozumitelný i těm, kdo ho vidí poprvé, že se musí vytvořit webové stránky a diskuzní skupiny (mailing listy) pro vývojáře a často i že je potřeba napsat vůbec první verzi dokumentace. To všechno představuje spoustu práce. A pokud se přece jen objeví vývojáři se zájmem o věc, bude to nějakou dobu znamenat zase jenom další zátěž – předtím, než začnou projektu jakkoliv přispívat, bude nutné nějakou dobu odpovídat na všechny jejich otázky. Jak řekl vývojář Jamie Zawinski o problematických začátcích projektu Mozilla:

> *Open source funguje, ale v žádném případě to není všelék. Pokud z toho všeho plyne nějaké poučení, je to fakt, že nemůžete vzít skomírající projekt, pokropit jej živou vodou zvanou „open source“, a vše tak zázračně vyřešit. Psát software je těžké. Jednoduchá řešení tu neexistují.*
>
> (citováno z **http://www.jwz.org/gruntle/nomo.html**)

S tím souvisí i další chyba, kterou je odbývání prezentace a přípravy instalačních balíčků s odůvodněním, že to přece můžeme dodělat později, až se projekt lépe rozjede. Prezentace a příprava instalačních balíčků v sobě zahrnují celou řadu úkolů, které mají společný cíl: snížit bariéru, jež brání vstupu. Pokud má být projekt přístupný pro nezasvěcené, je potřeba napsat dokumentaci pro uživatele a pro vývojáře, vytvořit webové stránky s informacemi pro nováčky, co nejvíc zautomatizovat kompilaci a instalaci softwaru atd. Mnozí programátoři, bohužel, považují uvedené práce za druhořadé – důležitější je podle nich kód samotný. Mají k tomu několik důvodů. Zaprvé jim to může připadat jako zbytečná námaha, protože tato činnost nejvíc prospívá těm, kdo projekt znají nejméně, a naopak. Koneckonců ti, kteří píšou zdrojový kód projektu, instalační balíčky vůbec nepotřebují. To, jak daný software nainstalovat, spravovat a používat, už dávno vědí, protože jej sami napsali. Zadruhé na to, abyste prezentaci a vytváření balíčků dělali dobře, potřebujete mít schopnosti, které se často značně liší od těch, které musí mít dobrý programátor. Lidé se obvykle zaměřují na to, v čem jsou dobří, a to i v případech, kdy by projektu lépe posloužilo, kdyby svůj čas věnovali i něčemu, co jim zas tolik nesedí. **V kapitole 2. Zahájení projektu** se budeme zabývat prezentací a vytvářením balíčků podrobněji a vysvětlíme si, proč je zcela zásadní, aby jim byla dána priorita už od samého začátku projektu.

Často se také objevuje mylný názor, že open source projekty potřebují řídit jen velmi málo, popřípadě vůbec, nebo zcela naopak, že je lze spravovat s použitím stejných metod, jaké se používají při uzavřeném vývoji. Řízení open source projektů nemusí být navenek příliš vidět, ale u úspěšných projektů

se přesto děje – obvykle v nějaké formě v zákulisí. Abychom si uvědomili, proč tomu tak je, stačí si představit následující situaci: Každý open source projekt je tvořen náhodným seskupením programátorů (a tato kategorie lidí, jak známo, bývá často poněkud nonkonformní), kteří se velmi pravděpodobně nikdy nesetkali a kteří prací na projektu mohou sledovat různé osobní cíle. Teď si představte, co by se stalo, kdyby taková skupina nebyla vůbec žádným způsobem řízena. Pokud by se nestal nějaký zázrak, buď by se okamžitě zhroutila, nebo rychle rozutekla. I kdybychom si to přáli sebevíc, věci se samy spravovat nebudou. I když může být projekt řízen celkem aktivně, děje se to často neformálními, jemnými a poměrně nenápadnými způsoby. Jediná věc, která drží skupinu vývojářů pohromadě, je společná víra, že dohromady toho zvládnou víc než každý zvlášť. Hlavním cílem vedení je tedy zajistit, aby této myšlence nepřestávali věřit, a to vytvořením zásad komunikace, zabraňováním tomu, aby byli užiteční vývojáři vytlačeni na okraj projektu jen kvůli svým osobnostním rozdílům, a obecně tím, že projekt bude místem, kam se vývojáři budou chtít vracet. O tom, jak přesně těchto věcí dosáhnout, pojednává zbytek této knihy.

Nakonec existuje ještě obecná kategorie problémů, kterou bychom mohli pojmenovat jako „kulturní nedorozumění“. Před deseti, možná i před pěti lety by bylo poněkud předčasné mluvit o něčem jako globální kultuře svobodného softwaru; to se ale změnilo. Pomalu se zde vyvinula skutečná, jedinečná kultura, která sice rozhodně není jednolitá – k vnitřním rozporům a k vytváření frakcí je přinejmenším stejně náchylná jako libovolná kultura vymezená geograficky –, ale jejíž jádro je v zásadě soudržné. Většina úspěšných open source projektů sdílí některé nebo všechny charakteristiky tohoto jádra. Tyto kultury odměňují určité typy chování a trestají jiné a vytvářejí atmosféru, která podněcuje neplánovanou spoluúčast, i když někdy za cenu centrálního řízení. Mají své vlastní pojetí toho, co je nezdvořilé a co se naopak sluší, které se může podstatně lišit od toho, co je běžné jinde. A co je nejdůležitější, dlouhodobí účastníci tyto standardy většinou přijali za své, takže v názoru na to, jaké chování se v této kultuře očekává, se zhruba shodují. Neúspěšné projekty se od tohoto jádra obvykle výrazným způsobem odchylují, třebaže možná neúmyslně, a často v nich nepanuje shoda, jak by rozumné standardní chování mělo vypadat. To znamená, že pokud se objeví jakýkoliv problém, může se situace začít prudce zhoršovat, protože účastníci nemají dostatečně zažité kulturní reflexy, které by se pro řešení takových rozporů daly použít.

Tato kniha je praktická příručka a ne antropologická studie nebo historický výklad. Nicméně alespoň základní povědomí o původu současné kultury svobodného softwaru je pro to, abychom mohli poskytnout nějaké praktické rady, zcela nezbytné. Člověk, který této kultuře rozumí, se ve světě open source může dostat na lecjaká místa, kde sice najde řadu místních odlišností ve zvycích a v dialektu, ale kde se přesto bude moct bez větších problémů a efektivně zapojit do spolupráce. Naproti tomu člověku, jenž tuto kulturu nechápe, bude připadat proces organizace projektu nebo účasti na něm obtížný a plný překvapení. Protože počet lidí vyvíjejících svobodný software stále prudce roste, mnoho z nich spadá do té druhé kategorie; celou kulturu zatím do velké míry tvoří čerství přistěhovalci a tenhle stav ještě nějakou dobu potrvá. Pokud si myslíte, že byste mohli být jedním z nich, poskytne vám další podkapitola základní informace pro pochopení diskuzí, s nimiž se setkáte později – jak v této knize, tak na internetu. (Na druhou stranu pokud v open source už nějakou dobu pracujete, je možné, že už o jeho historii víte dost, takže následující oddíl klidně přeskočte.)

Historie

Sdílení softwaru je tak staré jako software sám. V době, kdy počítače začínaly, byli výrobci přesvědčeni, že jejich konkurenční výhoda spočívá hlavně v inovaci hardwaru, a možnostem, jak zpeněžit software, proto nevěnovali příliš velkou pozornost. Většina lidí, kteří tyto první počítače kupovali a používali, byli vědci nebo technici, kteří byli schopni software dodávaný spolu s počítači upravovat a rozšiřovat sami. Tito zákazníci někdy nejen že posílali své úpravy zpět k výrobci, ale nabízeli je i ostatním vlastníkům podobných počítačů. Výrobci to často tolerovali a někdy k tomu dokonce i vybízeli – vylepšení softwaru, ať už ho provedl kdokoliv, činilo jejich počítače přitažlivějšími pro další potenciální zákazníky.

Ačkoliv toto rané období současnou kulturu svobodného softwaru v mnohém připomínalo, lišilo se od ní ve dvou zásadních ohledech. Zaprvé byl hardware dosud jen velmi málo standardizován – byla to doba mnoha zářných inovací v návrhu počítačů, ale tato rozmanitost výpočetních architektur znamenala, že všechno bylo nekompatibilní se vším. Takže software napsaný pro jeden počítač obvykle nefungoval na druhém. Programátoři se obvykle stávali odborníky na konkrétní architekturu nebo rodinu architektur (zatímco dnes by se spíš zdokonalovali v programovacím jazyce nebo v rodině jazyků a spoléhali by na to, že jejich odborné znalosti budou přenositelné na jakýkoliv počítačový hardware, se kterým kdy budou pracovat). Protože obvykle uměli opravdu dobře pracovat jen s určitým typem počítače, měla tato jejich specializace za následek to, že byl tento počítač pro ně a pro jejich kolegy přitažlivější. Bylo tedy v zájmu výrobců, aby se co nejvíce šířil kód závislý na jejich konkrétním produktu.

Zadruhé ještě neexistoval internet. Ačkoliv právních předpisů omezujících sdílení dat bylo mnohem méně než dnes, existovalo nesrovnatelně více technických překážek: prostředky pro přenos dat z místa na místo byly poměrně složité a nepraktické. Existovaly sice malé lokální sítě, které se daly používat pro sdílení informací mezi zaměstnanci ve stejné výzkumné laboratoři nebo ve firmě, ale pokud jste chtěli něco sdílet se všemi bez ohledu na to, kde se nacházejí, narazili jste na značné potíže. V mnoha případech lidé našli způsoby, jak tyto bariéry překonat. Někdy se různé skupiny navzájem kontaktovaly a posílaly si pak disky nebo pásky poštou, někdy fungovali jako středisko pro výměnu úprav samotní výrobci. Věci pomohlo i to, že v době prvních počítačů mnozí vývojáři pracovali na univerzitách, kde se očekávalo, že budou své znalosti publikovat. Ale způsoby, jimiž přenos dat fyzicky probíhal, znamenaly, že proti sdílení působil značný odpor, přímo úměrný vzdálenosti (ať už geografické nebo organizační), kterou musel software překonat. Celosvětové, okamžité sdílení dat, jaké známe dnes, nebylo jednoduše možné.

Vzestup proprietárního softwaru a svobodného softwaru

Jak počítačový průmysl dozrával, došlo k několika vzájemně souvisejícím změnám. Divoká rozmanitost hardwarových návrhů byla postupně vytlačena několika jasnými vítězi – ať už za jejich vítězství mohla lepší technologie, lepší marketing nebo obojí dohromady. Zároveň, a to nejen shodou okolností, se začaly vyvíjet takzvané „vyšší programovací jazyky“, což znamenalo, že bylo možné napsat

program jednou, v jednom jazyce, a nechat jej automaticky přeložit („zkompilovat“) tak, aby mohl běžet na různých typech počítačů. Výrobci hardwaru si dobře uvědomili, jaké důsledky bude tento vývoj mít: jejich zákazníci se teď mohli pouštět i do velkých softwarových projektů, aniž by se museli nezbytně vázat na konkrétní počítačovou architekturu. Jak byly z trhu postupně vytlačovány méně efektivní návrhy, začaly se snižovat i výkonnostní rozdíly mezi různými počítači, což znamenalo, že těm, kteří by obchodovali pouze s hardwarem, by začaly v blízké budoucnosti výrazně klesat zisky. Z hrubé výpočetní síly se stalo zaměnitelné zboží; jazýčkem na vahách se stal software. Prodávání softwaru, nebo přinejmenším jeho akceptování jako nedílné součásti prodeje hardwaru, se najednou stalo dobrou strategií.

To znamenalo, že výrobci museli začít dbát na přísnější dodržování autorských práv spojených se svými programy. Pokud by je uživatelé mezi sebou dál neomezeně sdíleli a upravovali, mohli by nezávisle implementovat některá zlepšení, která dodavatel začal prodávat jako „přidanou hodnotu“. A co by bylo ještě horší, sdílený zdrojový kód by se mohl dostat do rukou konkurence. Ironií všeho bylo, že se toto vše dělo přibližně ve stejné době, kdy se začal šířit internet. Právě v době, kdy začalo být neomezené sdílení softwaru konečně technicky možné, se stalo následkem změn v celém počítačovém průmyslu ekonomicky nežádoucím – přinejmenším z pohledu jednotlivých firem. Dodavatelé přitvrdili buď tím, že uživatelům odmítali přístup ke zdrojovému kódu, který na jejich počítačích běžel, nebo tím, že trvali na podepsání dohod o zachování důvěrnosti, které sdílení zamezovaly.

Vědomé hnutí odporu

Ve stejné době, kdy začal svět neomezované výměny kódu zvolna mizet, začala v hlavě přinejmenším jednoho programátora uzrávat myšlenka, jak se tomu vzepřít. Richard Stallman pracoval v sedmdesátých a raných osmdesátých letech minulého století v Laboratoři umělé inteligence (Artificial Intelligence Lab) na Massachusetts Institute of Technology – tedy při zpětném pohledu přímo uprostřed zlatého věku a v ideálním prostředí pro sdílení programů. V Laboratoři umělé inteligence se vyznávala přísná „hackerská etika“[3] a lidé zde nejen že byli vybízeni, aby se o svá zlepšení systému podělili s ostatními, ale dokonce se to od nich očekávalo. Jak Stallman později napsal:

> *Našemu softwaru jsme neříkali „svobodný software“, protože tento pojem ještě neexistoval, ale v zásadě to bylo totéž. Kdykoliv si chtěl někdo z jiné univerzity nebo nějaké firmy zkopírovat náš program, aby jej mohl u sebe používat, s radostí jsme mu ho poskytli. Pokud jste viděli, že někdo používá nějaký vám neznámý a zajímavý program, vždycky jste mohli požádat o zdrojový kód. Mohli jste si jej prostudovat, změnit jej, nebo z něj vykousnout nějaké části a použít je pro nový program.*
>
> (citováno z **http://www.gnu.org/gnu/thegnuproject.html**)

[3] Stallman používá slovo „hacker“ ve smyslu „někdo, kdo rád programuje a koho těší, že je v tom dobrý“, a ne v relativně novém významu „někdo, kdo se nabourává do počítačů“.

Změny, které se děly ve zbytku počítačového průmyslu, ale nakonec nedlouho po roce 1980 dostihly i Laboratoř umělé inteligence a celá rajská komunita kolem Stallmana se rozpadla. Jedna začínající společnost přetáhla z laboratoře mnoho programátorů, aby u ní pracovali na vývoji operačního systému. Ten se podobal tomu, na němž pracovali v laboratoři, ale měl velmi přísnou licenci. Ve stejné době Laboratoř umělé inteligence samotná získala nové zařízení, které bylo dodáno s proprietárním operačním systémem.

Stallman v tom, co se dělo, začal vnímat širší souvislosti:

> *Moderní počítače té doby, jako byl VAX nebo 68020, měly své vlastní operační systémy, ale žádný z nich nebyl svobodným softwarem – předtím, než jste mohli získat pouhou spustitelnou kopii programu, jste museli podepsat dohodu o důvěrnosti.*

To znamená, že prvním krokem při používání počítače byl slib, že nebudete ostatním pomáhat. Jakákoliv spolupráce v rámci komunity byla zakázána. Pravidlo, jež vlastníci proprietárního softwaru vyhlásili, znělo: „Pokud sdílíte s někým jiným, jste pirát. Pokud chcete nějaké změny, požádejte nás, abychom je provedli."

Něco v Stallmanovi se tomu ale vzpíralo; rozhodl se proto celému trendu postavit. Místo toho, aby pokračoval v práci v teď už téměř nefunkční Laboratoři umělé inteligence nebo aby přijal místo programátora v jedné z nových firem, kde by výsledky jeho práce ležely zamčené v trezoru, z laboratoře odešel a založil Projekt GNU a Free Software Foundation (FSF; Nadace svobodného softwaru). Cílem GNU[4] bylo vyvinout zcela svobodný a otevřený počítačový operační systém a balík aplikačního softwaru, v němž uživatelům nikdy nebude bráněno cokoliv měnit nebo své úpravy šířit. V podstatě začal Stallman znovu vytvářet to, co bylo v Laboratoři umělé inteligence zničeno, ale tentokrát v celosvětovém měřítku a bez slabin, kvůli nimž se kultura laboratoře nakonec rozpadla.

Kromě prací na novém operačním systému Stallman vymyslel autorskou licenci, jejíž znění zaručovalo, že jeho kód zůstane trvale svobodný. Celá GNU General Public License (GPL) je velmi rafinovaný právnický výkon. Říká, že kód může být kopírován a upravován bez omezení a že jak kopie, tak odvozená díla (tedy upravené verze) musí být distribuovány se stejnou licencí jako originál, bez jakýchkoliv dalších omezení. Používá tedy zákon o autorském právu tak, aby dosáhl zcela opačného efektu, než tradiční copyright má mít – místo toho, aby distribuci softwaru nějak omezoval, zabraňuje všem, dokonce i svému autorovi samotnému, aby jakékoliv omezení ustanovil. Pro Stallmana byl tento postup lepší, než kdyby jednoduše prohlásil svůj kód za volně šiřitelný. V takovém případě by totiž mohla být jakákoliv jeho kopie začleněna do proprietárního programu (což se u programů s tolerantními autorskými licencemi stávalo). I když by takové zabudování žádným způsobem nesnížilo dostupnost původního kódu, znamenalo by to, že by ze Stallmanova úsilí mohl těžit jeho nepřítel – proprietární software. O GPL lze uvažovat jako o jisté formě protekcionismu vztahující se na svobodný software,

---

[4] Název projektu je zkratka z „GNU's Not Unix" („GNU není Unix"), přičemž „GNU" v této delší verzi je zkratka pro… přesně to samé.

protože nesvobodnému softwaru zabraňuje naplno využít výhod kódu vydaného pod GPL. Na licenci GPL a její vztah k ostatním licencím svobodného softwaru se podíváme podrobněji v kapitole **9. Licence, autorská práva a patenty.**

S pomocí mnoha programátorů, z nichž někteří Stallmanovu ideologii sdíleli a někteří jednoduše chtěli, aby existovalo velké množství dostupného svobodného kódu, začal projekt GNU vydávat svobodné náhrady pro mnoho z těch nejdůležitějších komponent operačního systému. Vzhledem k tomu, že standardizace počítačového hardwaru i softwaru v té době už značně pokročila, bylo možné tyto náhrady z projektu GNU používat i v systémech, jež jinak svobodné nebyly, což také mnoho lidí dělalo. Zvlášť úspěšnými výstupy projektu GNU byly jeho textový editor Emacs a kompilátor jazyka C (GCC), které si získaly velkou a oddanou skupinu následovníků ne na základě ideologie, ale pro jejich technické kvality. Kolem roku 1990 GNU vytvořil již většinu komponent svobodného operačního systému s výjimkou jádra – tedy té části, kterou počítač zavádí při startu (boot) a která je zodpovědná za správu paměti, diskového prostoru a dalších systémových zdrojů.

Naneštěstí si projekt GNU zvolil návrh jádra, který se z hlediska implementace ukázal obtížnější, než se očekávalo. Celý vývoj se začal zpožďovat, a kvůli tomu to tedy nakonec nebyla Free Software Foundation, kdo vytvořil první zcela svobodný operační systém. Poslední chybějící kus skládanky místo nich dodal Linus Torvalds, finský student informatiky, který s pomocí dobrovolníků z celého světa dokončil svobodné jádro vycházející z konzervativnějšího návrhu. Dal mu název Linux; poté, co bylo toto jádro zkombinováno s existujícími programy GNU, vznikl zcela svobodný operační systém. Poprvé v historii jste mohli nastartovat počítač a pracovat bez použití jakéhokoliv proprietárního softwaru.[5]

Velká část softwaru tohoto nového operačního systému nepocházela z projektu GNU. Ve skutečnosti dokonce GNU nebyla jedinou skupinou, která by se snažila vytvořit svobodný operační systém – ve stejné době už byl například vyvíjen kód, z nějž se časem staly systémy NetBSD a FreeBSD. Význam Free Software Foundation nespočívá jen v tom, že psali programy, ale také v jejich politické rétorice. Tím, že svobodný software brali jako svůj hlavní program a nejen jako něco, co může být celkem užitečné, bylo pro programátory obtížné si na celou věc neudělat nějaký názor. Dokonce i ti, kteří s FSF nesouhlasili, se celým problémem museli zabývat, už jen proto, aby mohli vyjádřit svůj odlišný názor. Úspěch FSF jako šiřitelů propagandy spočívá v tom, že svůj zdrojový kód svázali prostřednictvím GPL a dalších textů i s jasným politickým poselstvím. Jak se jejich programy šířily dál, šířilo se s nimi i toto poselství.

---

[5] Technicky vzato nebyl Linux úplně první. Nedlouho před Linuxem se objevil jiný svobodný operační systém pro počítače kompatibilní s IBM; nazýval se 386BSD. Tento systém ale bylo mnohem obtížnější spustit a udržet v provozu. Význam Linuxu spočíval nejen v tom, že byl svobodný, ale také v tom, že u něj byla celkem vysoká pravděpodobnost, že se na vašem počítači po instalaci skutečně i rozběhne.

Neúmyslné hnutí odporu

Na rodící se scéně svobodného softwaru se děla spousta dalších věcí, ale málo z nich bylo tak výrazně ideologických jako Stallmanův projekt GNU. Jedním z nejdůležitějších byl systém Berkeley Software Distribution (BSD) vytvořený programátory University of California v Berkeley jako postupná reimplementace operačního systému Unix, jenž byl až do konce 70. let víceméně proprietárním výzkumným projektem společnosti AT&T. Skupina BSD nečinila žádná otevřeně politická prohlášení o tom, že se programátoři musí sjednotit a vzájemně sdílet svou práci, ale v praxi tuto myšlenku realizovala, a to s velkým nadšením, při koordinaci celého masivního distribuovaného vývoje; v rámci projektu byly od základů přepsány unixové programy pro příkazový řádek, jeho knihovny a nakonec i samotné jádro operačního systému, a to převážně díky dobrovolníkům. Projekt BSD se stal ukázkovým příkladem neideologického vývoje svobodného softwaru a byl také místem, kde načerpala své první zkušenosti řada vývojářů, kteří pak zůstali v open source světě aktivní.

Další zatěžkávací zkouškou kooperativního vývoje byl X Window System, svobodné, síťově transparentní grafické výpočetní prostředí vyvinuté v polovině 80. let na MIT ve spolupráci s prodejci hardwaru, kteří měli společný zájem na tom, aby byli svým zákazníkům schopni nabídnout systém pracující s okny. Zdaleka nešlo o odpor vůči proprietárnímu softwaru – licence pro X dokonce záměrně dovolovala proprietární rozšiřování svobodného jádra systému, protože každý člen celého sdružení chtěl mít možnost základní distribuci X vylepšit, a získat tím konkurenční výhodu oproti těm ostatním. Samotné X Windows[6] byly sice svobodným softwarem, ale především proto, aby existovaly rovné podmínky pro vzájemný souboj obchodních zájmů, a ne ve snaze ukončit převahu proprietárního softwaru. Dalším příkladem, který projekt GNU o pár let předběhl, byl TeX Donalda Knuthe, svobodný systém pro sázení dokumentů, jenž kvalitou svých výstupů odpovídal požadavkům nakladatelství. TeX byl vydán s licencí, která komukoliv umožňovala kód upravovat a šířit, ale výsledek těchto úprav se nesměl nazývat „TeX“, pokud nesplnil velmi přísnou sadu testů kompatibility (je to tedy příklad svobodné licence „chránící obchodní značku“; tento typ probereme podrobněji v kapitole **9. Licence, autorská práva a patenty**). Knuth se vůbec nesnažil zaujmout nějaké stanovisko v debatě svobodný versus proprietární software. Potřeboval zkrátka lepší sázecí systém, aby mohl dokončit to, co bylo jeho primárním cílem, totiž kniha o počítačovém programování, a neviděl žádný důvod, proč by svůj systém neměl po dokončení zveřejnit.

Aniž bychom jmenovali každý projekt a každou licenci, můžeme bezpečně říct, že na konci 80. let již existovalo velké množství svobodného softwaru s celou řadou různých licencí. Rozmanitost těchto licencí odpovídala tomu, jak různorodé byly i motivace těchto projektů. Dokonce i někteří z programátorů, kteří si zvolili GNU GPL, byli ideologicky mnohem méně vyhranění než projekt GNU samotný. Třebaže mnoho z těchto vývojářů práce na svobodném softwaru bavila, nechápali proprietární software jako společenské zlo. Našli se sice tací, kteří považovali za svou morální povinnost zbavit svět „softwarového hamounění“ („software hoarding“ – Stallmanův pojem pro nesvobodný software), ale existovali

[6] Projekt sám má raději název „X Window System“, ale v praxi se obvykle používá „X Windows“, protože to je kratší.

i jiní, které pohánělo jejich technické zaujetí nebo radost ze spolupráce s podobně smýšlejícími lidmi, či dokonce obyčejná lidská touha po slávě. Přesto ale obvykle nedocházelo k tomu, že by se tyto nesourodé motivace nějakým způsobem tloukly. To je dáno částečně tím, že na rozdíl od ostatních tvůrčích forem, jako je literatura nebo výtvarné umění, musí software k tomu, aby mohl být považován za úspěšný, splnit několik částečně objektivních testů: musí fungovat a být do rozumné míry prostý chyb. To všem účastníkům projektu automaticky dává jakýsi společný základ, důvod a rámec pro vzájemnou spolupráci, aniž by se museli příliš starat o jinou než technickou způsobilost.

K tomu, aby vývojáři drželi pospolu, pak měli ještě další důvod – ukázalo se totiž, že ve světě svobodného softwaru vzniká někdy zdrojový kód velmi vysoké kvality. V některých případech byly tyto programy prokazatelně technicky dokonalejší než jejich nejbližší proprietární alternativa, zatímco v dalších byly přinejmenším srovnatelné; vždy ale byly samozřejmě levnější. Pro pár lidí mohla mít motivace k používání svobodného softwaru čistě filozofický základ, ale většina lidí jej používala prostě proto, že byl lepší. A z těch, kteří ho používali, bylo vždy určité procento ochotné věnovat svůj čas a schopnosti tomu, aby jej pomohli udržovat a zlepšovat.

Tendence psát dobré programy se rozhodně neprojevovala všude, ale u projektů svobodného softwaru celého světa se dala pozorovat stále častěji. Toho si postupně začaly všímat firmy, které byly na svém softwaru výrazně závislé. Mnohé z nich zjistily, že svobodný software už při své každodenní práci používají a neví o tom – protože vedení společnosti nemusí vždy tušit, co přesně jejich IT oddělení dělá. Různé společnosti začaly k projektům svobodného softwaru i na veřejnosti zaujímat stále aktivnější postoj. Dávaly jim k dispozici svůj čas a své vybavení a v některých případech dokonce vývoj svobodných programů přímo financovaly. Podobné investice se jim totiž v nejlepších případech mohou i několikanásobně vrátit. Sponzor platí pouze malé množství odborníků, kteří se díky tomu mohou projektu naplno věnovat; přitom ale sklízí plody práce všech zúčastněných, tedy včetně neplacených dobrovolníků a programátorů, kteří jsou placeni jinými organizacemi.

„Free“ versus „open source“

S tím, jak začaly společnosti světa svobodnému softwaru věnovat čím dál tím větší pozornost, vyvstaly před programátory nové problémy – tentokrát s prezentací. Jedním z nich bylo samotné slovo „free“, které v angličtině může znamenat „svobodný“, ale také „zdarma“. Když lidé slyší pojem „free software“ poprvé, mnoho z nich se mylně domnívá, že to znamená „software zadarmo“. Pravda je taková, že veškerý svobodný software je sice dostupný zdarma,[7] ale to, že je nějaký software zadarmo, ještě neznamená, že je svobodný. Například během války prohlížečů v 90. letech poskytly jak Netscape, tak Microsoft při rvačce o získání podílu na trhu své konkurenční webové prohlížeče veřejnosti zdarma. Ani jeden z nich však nebyl svobodný software. Nebylo možné získat jejich

[7] Za šíření kopií svobodného softwaru si sice můžete účtovat nějaký poplatek, ale protože nelze zabránit tomu, aby příjemce pokračoval v dalším šíření zdarma, klesá vaše cena prakticky okamžitě k nule.

zdrojový kód; i kdyby se vám to podařilo, neměli jste právo jej upravovat a šířit dál.[8] Jedinou věcí, kterou jste mohli udělat, bylo stáhnout a spustit zkompilovaný program. Tyto prohlížeče nebyly o nic víc svobodné než software, jenž jste si koupili v obchodě v krabici – jenom byly levnější.

Tento zmatek kolem slova „free“ je způsoben výhradně jeho nešťastnou dvojznačností v anglickém jazyce. Většina jiných jazyků rozlišuje nízkou cenu od svobody (například mluvčím románských jazyků je ihned jasný rozdíl mezi *gratis* a *libre*). Ale protože je angličtina de facto hlavním komunikačním jazykem internetu, znamená to, že její problémy jsou do jisté míry problémy všech. Nedorozumění kolem slova „free“ bylo tak časté, že programátoři svobodného softwaru časem vytvořili tuto standardní formulku: „Je to *free* jako *freedom* (svoboda) – tedy ve smyslu *free speech* (svoboda slova) a ne *free beer* (pivo zdarma).“ To ale nic nemění na tom, že je poněkud unavující to pořád dokola vysvětlovat. Mnozí programátoři měli pocit – a to ne bezdůvodný –, že nejednoznačnost slova „free“ brání tomu, aby veřejnost pochopila, o jaký software jde.

Ale ukázalo se, že problém je ještě závažnější. Slovo „free“ totiž nese ještě další význam, tentokrát morální: pokud měla být cílem svoboda jako taková, pak nezáleželo na tom, zda byl shodou okolností tento svobodný software také lepší nebo pro nějaký podnik ekonomicky výhodnější. To byl jen příjemný vedlejší účinek; primární motivace zde ale nebyla v jádru ani technická, ani ekonomická, ale filozofická. Zastávání svobody softwaru navíc poukazovalo na to, že se často objevuje značný rozpor u společností, které sice v rámci své obchodní činnosti chtějí podporovat konkrétní svobodné programy, ale jinak dál pokračují v nabízení proprietárního softwaru.

A tak stálo před komunitou, která už tak mířila ke krizi identity, další dilema. Sami programátoři, kteří svobodný software píší, se nikdy neshodli na tom, jaký cíl tohoto hnutí vlastně je, pokud tedy vůbec nějaký existuje. Bylo by dokonce zavádějící tvrdit, že se zde objevuje celá škála názorů od jednoho extrémního postoje k druhému, protože taková lineární osa tu jednoduše neexistuje; místo toho panuje obrovská roztříštěnost názorů na mnoho dílčích aspektů věci. Pokud ale na chvíli odhlédneme od detailů, uvidíme zde v zásadě dvě velké skupiny uvažujících. Jedna z nich zastává Stallmanovo stanovisko, podle nějž je hlavním cílem svoboda sdílet a upravovat; pokud přestanete mluvit o svobodě, znamená to, že jste zapomněli na to nejdůležitější. Jiní mají pocit, že nejdůležitějším argumentem, který pro jakýkoliv software hovoří, je tento software samotný, a odmítají prohlásit, že veškerý proprietární software je ze své podstaty špatný. Někteří, ale ne všichni programátoři svobodného softwaru věří, že právo na to, určit podmínky pro distribuci softwaru, by měl mít jeho autor (nebo v případě placené práce jeho zaměstnavatel) a že volba licence nemusí nutně poukazovat na nějaké morální stanovisko.

Tyto rozdíly nebylo dlouho nutné nijak zvlášť zkoumat nebo se k nim vyjadřovat, ale rostoucí úspěch svobodného softwaru v podnikatelské sféře si to časem vynutil. V roce 1998 byl jako alternativa k termínu „svobodný software“ (free software) vytvořen pojem *open source*. Zasloužila se o to skupina

[8] Zdrojový kód Netscape Navigatoru byl nakonec v roce 1998 přece jen zveřejněn s open source licencí a stal se základem webového prohlížeče sdružení Mozilla. Viz **http://www.mozilla.org/**.

programátorů, kteří časem vytvořili The Open Source Initiative (Iniciativa open source, OSI).[9] Členové OSI měli pocit, že to není jen spojení „free software“, které způsobuje potíže, ale že zde existuje větší problém, jehož je slovo „free“ jako takové jen symptomem – totiž že celé hnutí potřebuje marketingový program, který by se dal představit světu obchodních společností, protože hovory o tom, jaké jsou morální a společenské výhody sdílení, na obchodních jednáních nikdo poslouchat nebude. Sdružení OSI celou věc vyjádřilo takto:

> *The Open Source Initiative je marketingový program pro svobodný software. Je to způsob, jak propagovat celou myšlenku svobodného softwaru postaveného na pevných pragmatických základech a ne na ideologickém zápalu. Podstata, a tedy i největší síla našeho softwaru se nemění; co se mění, je jeho symbolický rozměr a celý poraženecký přístup, který tato komunita má. …*
>
> *Většinu technicky založených lidí není nutné přesvědčovat o výhodách konceptu open source, ale o jeho jméně. Proč bychom neměli dál používat zavedený termín „svobodný software“?*
>
> *Jedním z důvodů je to, že pojem „free software“ lze snadno chápat nesprávně, což často vede ke konfliktům. …*
>
> *Ale tím hlavním, co nás k této změně označení vede, je marketing. Svou koncepci chceme nabídnout obchodnímu světu. Máme produkt, s nímž lze zvítězit, ale způsob jeho uvádění na trh byl dosud zcela špatný. Pojem „free software“ byl v podnikatelské sféře špatně pochopen; touha sdílet byla brána jako boj proti kapitalismu nebo ještě hůře jako krádež.*
>
> *Ředitelé velkých obchodních společností a jejich technologických oddělení na koncept „svobodného softwaru“ nikdy slyšet nebudou. Ale co když vezmeme stejné tradice, stejné lidi, stejné licence, které známe jako „svobodný software“, a změníme nálepku na „open source“? To už je něco, o čem se budou ochotni bavit.*
> *Někteří hackeři tomuhle odmítají uvěřit, ale to je dané tím, že jsou to technicky orientovaní lidé, kteří uvažují v konkrétních a reálných pojmech; nedokážou pochopit, jak je při prodeji čehokoliv důležitý celkový dojem, vaše image.*
>
> *Ve světě marketingu jste tím, čím vypadáte, že jste. Pokud vyvoláme dojem, že jsme ochotni slézt z barikád a spolupracovat se světem podnikání, bude to zrovna tak důležité jako naše skutečné chování, naše zásady a náš software.*

(citováno z **http://www.opensource.org/**. Přesněji řečeno z dřívější podoby těchto stránek. OSI je od té doby zřejmě stáhla, ale stále je lze najít na **http://web.archive.org/web/20021204155057/http://www.opensource.org/advocacy/faq.php** a **http://web.archive.org/web/20021204155022/http://www.opensource.org/advocacy/case_for_hackers.php#marketing.**)

---

[9] Domácí stránka OSI se nachází na **http://www.opensource.org/**.

V tomto textu se objevují náznaky mnoha sporných otázek. Zmiňuje se o „našich zásadách“, ale chytře se vyhýbá tomu říct, o jaké zásady přesně jde. Pro některé se může jednat o přesvědčení, že programy vyvinuté v rámci otevřeného procesu budou lepší; jiní těmito zásadami budou rozumět princip, že by se všechny informace měly sdílet. Používá se zde slovo „krádež“ (patrně) ve smyslu nelegálního kopírování – proti čemuž mnozí namítají, že pokud přitom původní vlastník o nic nepřišel, nemůže už z principu jít o krádež. Naznačuje se zde, že by hnutí za svobodný software mohlo být vnímáno jako antikapitalismus, ale tomu, zda je takové obvinění ve skutečnosti něčím podloženo, se pečlivě vyhýbá.

Tím ale nechci říct, že by byly stránky OSI nekonzistentní nebo úmyslně zavádějící. To rozhodně nejsou. Spíše jde o skvělý příklad toho, co podle OSI v hnutí za svobodný software chybělo, tedy dobrého marketingu; „dobrý“ zde znamená „životaschopný v obchodním světě“. Iniciativa open source tak dala řadě lidí přesně to, co hledali – způsob, jak mluvit o svobodném softwaru jako o metodologii vývoje a obchodní strategii a ne jako o filozofickém hnutí.

Vznik Iniciativy změnil celou situaci svobodného softwaru. OSI zde podala definici rozporu, který dlouho zůstával nepojmenován, čímž celé hnutí donutila připustit, že se nestačí zabývat politikou jen navenek, ale také uvnitř. Následkem toho je, že obě strany musely najít společnou řeč, protože většina projektů zahrnuje programátory z obou táborů i účastníky, které do žádné kategorie jednoznačně zařadit nelze. To neznamená, že by přestali mluvit o svých morálních zásadách – pokud se například někdo dopustí prohřešku vůči tradiční „hackerské etice“, bývá mu to někdy vyčteno. Jen málokdy ale nějaký vývojář svobodného nebo open source softwaru zpochybní motivaci ostatních účastníků projektu. Příspěvek je důležitější než jeho autor. Pokud někdo píše dobrý zdrojový kód, neptáte se ho, zda to dělá z morálních důvodů, zda jej za to někdo platí, zda si vylepšuje svůj profesní životopis nebo cokoliv jiného. Jeho příspěvek hodnotíte z technického hlediska a reagujete na technickém základě. Dokonce i explicitně politické organizace, jako je projekt Debian, jenž si dal za cíl nabídnout stoprocentně svobodné výpočetní prostředí, se k integraci nesvobodného kódu staví celkem otevřeně a spolupracují s programátory, kteří stejné cíle nesdílejí.

Současná situace

Při řízení projektu svobodného softwaru není potřeba řešit tyto závažné filozofické záležitosti každý den. Programátoři nebudou trvat na tom, aby všichni ostatní účastníci projektu plně souhlasili s jejich pohledem na věc – a ti, kteří na tom trvat budou, rychle zjistí, že pak nemůžou pracovat prakticky nikde. Musíte si ale být vědomi toho, že debata „svobodný“ versus „open source“ existuje, a to jak proto, abyste nechtěně neřekli něco, co by někteří účastníci mohli vnímat jako nepřátelské, tak proto, že pochopení motivace vašich vývojářů je tím nejlepším a do určité míry vlastně i jediným způsobem, jak projekt řídit.

Rozhodnutí se pro svobodný software je volbou každého jednotlivce. Abyste byli v této kultuře úspěšní, musíte pochopit, proč lidé tuto volbu učinili. Donucovací techniky zde nefungují. Pokud se vývojáři v jednom projektu necítí dobře, pak jednoduše odejdou jinam. Svobodný software je výjimečný i mezi jinými skupinami dobrovolníků, protože jejich osobní investice je zde jen velmi malá. Většina lidí zapo-

jených v projektu se s těmi ostatními nikdy osobně nesetká; projektu věnují kousky svého času zkrátka tehdy, když se jim chce. Běžné způsoby, kterými se lidé dávají dohromady a vytvářejí trvalé skupiny, jsou omezeny jen na uzounký kanál – psané slovo přenášené elektronicky. Z tohoto důvodu může trvat poměrně dlouho, než vznikne skupina, která opravdu drží pohromadě a jde za stejným cílem. A naopak je velmi snadné, aby projekt ztratil potenciálního dobrovolníka už během prvních pěti minut. Pokud nebude jeho první dojem z projektu dobrý, pak mu jen málokdy dá i druhou šanci.

Pomíjivost nebo přesněji řečeno potenciální pomíjivost vztahů je možná tím vůbec nejtěžším problémem, kterému musí nový projekt čelit. Jak všechny tyto lidi přesvědčit, aby zůstali pohromadě dost dlouho na to, aby vzniklo něco užitečného? Odpověď na tuto otázku je natolik složitá, že zabere prakticky celý zbytek této knihy. Pokud bychom ale měli odpovědět jednou větou, vypadala by asi takto:

> *Lidé by měli mít pocit, že míra jejich zapojení do projektu a jejich vliv na projekt jsou přímo úměrné jejich přínosu.*

Žádná skupina vývojářů (nebo potenciálních vývojářů) by neměla mít pocit méněcennosti nebo diskriminace z jiných než technických důvodů. Je jasné, že projekty sponzorované firmami nebo takové, v nichž existují i placení vývojáři, musí být v tomto ohledu zvlášť opatrné, jak podrobněji popíšeme v kapitole **5. Peníze**. To ale samozřejmě neznamená, že pokud není projekt sponzorovaný žádnou firmou, není se čeho bát. Peníze jsou jen jedním z mnoha faktorů, které mohou úspěšnost projektu ovlivnit. Jsou tu i další otázky: jaký programovací jazyk, jakou licenci a jaký vývojový proces bude nejlepší, jakou infrastrukturu je potřeba vytvořit, jak efektivně oznámit zahájení projektu a mnoho dalších. Tomu, jak při startu projektu vykročit tou správnou nohou, se věnuje následující kapitola.

2. Zahájení projektu

2. **Zahájení projektu** — 45
Nejdříve se ale rozhlédněte — **47**

Začněte od toho, co máte k dispozici — 47
Vyberte dobré jméno — **48**
Určete jasné cíle projektu — **49**
Uveďte, že jde o svobodný projekt — **50**
Seznam funkcí a požadavků — **51**
Stav vývoje — **51**
Stahování — **52**
Zpřístupnění systémů správy verzí a sledování chyb — **53**
Komunikační kanály — **54**
Pokyny pro vývojáře — **55**
Dokumentace — **55**
Dostupnost dokumentace — **57**
Dokumentace pro vývojáře — **58**
Vzorový výstup a snímky obrazovky — **58**
Kompletní hosting — **59**

Výběr licence a její uplatnění — 59
Licence typu „vše je povoleno“ — **60**
GPL — **60**
Jak licenci aplikovat — **60**

Udávání tónu — 61
Vyhněte se soukromým diskuzím — **62**
Nezdvořilé chování potlačte hned v zárodku — **64**
Kontrolujte kód veřejně — **65**
Když otvíráte dosud uzavřený projekt, uvědomte si rozsah změn — **67**

Oznamování — 68

2. Zahájení projektu

Klasický model toho, jak projekty svobodného softwaru začínají, popsal Eric Raymond ve svém proslulém eseji o procesech používaných v open source, který má název *The Cathedral and the Bazaar* (Katedrála a bazar). Napsal:

> *Veškerý dobrý software začal tak, že se nějaký vývojář potřeboval podrbat tam, kde jej něco svědilo.*
>
> (citováno z **http://www.catb.org/~esr/writings/cathedral-bazaar/**)

Povšimněte si, že Raymond neřekl, že open source projekty vznikají jen tehdy, když někoho něco svědí. Řekl něco trochu jiného: že dobrý software vzniká tehdy, když má programátor osobní zájem na tom, aby byl nějaký problém vyřešen; ukázalo se, že právě to bývá skutečně i tou nejčastější motivací pro založení projektu svobodného softwaru.

Pro většinu z nich to platí dodnes, ale už jich není tolik jako v roce 1997, kdy Raymond článek napsal. V současnosti můžeme pozorovat fenomén, kdy řada organizací – včetně klasických společností orientovaných na zisk – spouští velké, centrálně řízené open source projekty vyvíjené od základů. Zdrojem značné části svobodného softwaru nadále zůstává osamělý programátor, který napíše kód, jenž řeší jeho vlastní problém, a pak si uvědomí, že se výsledek jeho práce dá použít i jinde, ale existují i jiné varianty.

Raymondův postřeh nicméně stále platí. Základní podmínkou je, aby tvůrci softwaru měli přímý zájem na jeho úspěchu, protože jej sami používají. Pokud software nedělá to, co by měl, pak osoba, popřípadě organizace, která jej vytváří, bude při své každodenní práci nespokojen. Vezměme si například projekt OpenAdapter (**http://www.openadapter.org/**), který spustila investiční banka Dresdner Kleinwort Wasserstein jako open source framework pro integraci několika různých finančních informačních systémů; jen těžko bychom mohli tvrdit, že tento projekt řešil problém nějakého jednotlivého programátora. Byla to instituce, která něco potřebovala vyřešit. Tento problém ale přímo vyplýval ze zkušeností této instituce samotné a jejích partnerů, takže pokud by projekt nesplnil svůj úkol, nepochybně si toho všimnou. V takovémto uspořádání vzniká dobrý software, protože zpětná vazba míří tím správným směrem. Není to program, který by byl napsán proto, aby mohl být prodáván někomu jinému a řešil cizí problémy. Byl napsán jako reakce na řešení něčího problému a pak sdílen s ostatními, jako kdyby tento problém byla nějaká nakažlivá nemoc a software byl lékem šířeným proto, aby zabránil epidemii.

Tato kapitola se zabývá tím, jak do světa uvést nový projekt svobodného softwaru, ale mnohá z jejích doporučení by zněla velmi povědomě zdravotnickým organizacím distribuujícím léky. Tyto dva cíle jsou totiž velmi podobné. Chcete-li, aby bylo jasné, k čemu je lék dobrý, dejte jej do rukou těch správných lidí a ujistěte se, že jej budou umět správně použít. Ale u softwaru chcete navíc přilákat některé z příjemců k tomu, aby se k probíhajícímu výzkumu, jenž má lék vylepšit, sami připojili.

Proces šíření svobodného softwaru má dva aspekty. Software potřebuje získat jak nové uživatele, tak vývojáře. Tyto dva požadavky sice nejsou protichůdné, ale zvyšují složitost počáteční prezentace projektu. Některé informace jsou užitečné pro obě skupiny, některé ale jen pro jednu nebo pro druhou. V obou případech by měly být sdělovány na základě principu odstupňované prezentace. To znamená, že v každé etapě by měla míra uváděných podrobností odpovídat množství času a úsilí, které musí čtenář vynaložit. Čím více úsilí vynaloží, tím více informací by měl získat. Pokud zmíněné dvě oblasti nebudou úzce sladěny, můžete u zájemců rychle ztratit důvěru, takže se raději přestanou snažit úplně.

Z toho logicky vyplývá, že na vzhledu záleží. To je něco, čemu zejména programátoři často nechtějí věřit. Povyšování obsahu nad formu je u nich bezmála profesionální ctí. Není žádná náhoda, že mnozí programátoři mají odpor k marketingu a k práci s veřejností; zrovna tak není náhoda, že profesionální grafičtí návrháři jsou často tím, co programátoři sami vytvoří, upřímně zděšeni.

Je to škoda, protože existují situace, kdy právě forma je podstatou, a prezentace projektu je jejich součástí. Například jednou z prvních věcí, kterou se návštěvníci o projektu dozví, je to, jak vypadají jeho webové stránky. Tato informace je vstřebána ještě předtím, než začnou vnímat jejich skutečný obsah – dříve, než si přečtou jakýkoliv text nebo kliknou na kterýkoliv odkaz. I když se vám to může zdát nespravedlivé, získávání bezprostředního prvního dojmu je něco, čemu nikdo nedokáže zabránit. Vzhled webových stránek jasně ukazuje, zda byla prezentaci projektu věnována nějaká péče. Na rozpoznání, kolik času někdo něčemu věnoval, jsou lidé neobyčejně citliví. Většina z nás dokáže už na první pohled říct, zda byly webové stránky spíchnuty narychlo, nebo zda nad nimi někdo vážně uvažoval. Jde o vůbec první informaci, kterou se váš projekt zviditelňuje, a dojem, jenž vytvoří, pak bude vztažen na všechno ostatní.

Takže i když se značná část této kapitoly bude zabývat obsahem, s nímž by váš projekt měl začít, nezapomínejte, že záleží i na vzhledu a na celkovém dojmu. Protože webové stránky projektu musí být přizpůsobeny dvěma různým typům návštěvníků, uživatelům a vývojářům, je potřeba věnovat zvláštní pozornost jejich srozumitelnosti a přehlednosti. Ačkoliv tato kniha není tím správným místem pro obecné pojednání o webdesignu, jeden princip je natolik důležitý, že si zaslouží, abychom se o něm zde zmínili, zejména pokud vaše stránky slouží několika různým skupinám čtenářů, které se mohou překrývat: návštěvníci by vždy měli mít přibližnou představu o tom, kam daný odkaz vede, ještě předtím, než na něj kliknou. Například odkazy, jež vedou do uživatelské dokumentace, by měly na první pohled nějak oznamovat, že vedou právě tam a ne řekněme do vývojářské dokumentace. Velká část práce ve vedení projektu spočívá v poskytování informací, ale patří sem i nutnost poskytovat je komfortním způsobem. Už pouhá přítomnost určité standardní nabídky na očekávaných místech bude působit dobrým dojmem na uživatele i vývojáře, kteří se rozhodují, zda se budou chtít zapojit. Říká jim totiž, že projekt má celkem jasnou představu, co chce dělat, že se zamyslel nad tím, jaké otázky budou lidé klást, a že se na ně pokusil zodpovědět způsobem, který ze strany tazatele vyžaduje co nejmenší úsilí. Projekt, který dává najevo, že nepodcenil svou přípravu, vysílá návštěvníkům signál: „Pokud se zapojíte, nebude to pro vás ztráta času," což je přesně to, co lidé chtějí slyšet.

Nejdříve se ale rozhlédněte

Než spustíte libovolný open source projekt, nezapomínejte udělat jednu velmi důležitou věc:

Vždy se rozhlédněte kolem sebe, jestli už neexistuje nějaký projekt, který dělá to, co chcete dělat i vy. Je celkem pravděpodobné, že problémem, který jste se rozhodli vyřešit, se už zabýval někdo před vámi. Pokud už jej vyřešil a svůj kód zveřejnil se svobodnou licencí, pak je z vaší strany zcela zbytečné zkoušet objevovat Ameriku. Existují samozřejmě výjimky – pokud chcete zahájit projekt proto, abyste se něco nového naučili, pak vám zdrojový kód někoho jiného nijak nepomůže; zrovna tak se může stát, že projekt, který nosíte v hlavě, je natolik specializovaný, že je šance, že by jej někdo řešil před vámi, zcela nulová. Obecně ale platí, že není žádný důvod se neporozhlédnout; může se vám to navíc velmi vyplatit. Pokud standardní internetové vyhledávače nic nenajdou, zkuste hledat na **http://freshmeat.net/** (server s novinkami o open source projektech, o němž si více povíme později), na **http://www.sourceforge.net/** a v adresáři svobodného softwaru, který spravuje Free Software Foundation na adrese **http://directory.fsf.org/**.

I kdybyste nenašli přesně to, co hledáte, můžete najít něco natolik podobného, že bude rozumnější se k tomuto projektu připojit a přidat požadovanou funkčnost, než začínat sami zcela od začátku.

Začněte od toho, co máte k dispozici

Porozhlédli jste se kolem, nenašli jste nic, co by splňovalo vaše požadavky, a rozhodli jste se založit nový projekt.

Co teď?

Nejtěžší částí spouštění projektu svobodného softwaru je najít způsob, jak své představy zveřejnit. Vy nebo vaše organizace můžete sice velmi dobře vědět, co přesně chcete, ale vyjádřit to tak, aby to bylo srozumitelné i ostatním, vyžaduje celkem dost práce. Přesto je velmi podstatné, abyste si na to našli čas. Vy a ostatní zakladatelé se musíte rozhodnout, co přesně bude cílem projektu – to znamená, že si musíte určit nějaké meze a stanovit nejen to, co váš software bude dělat, ale také co dělat nebude; své závěry pak sepište do jednoho dokumentu vašich závazně stanovených cílů. Tato část většinou není příliš obtížná, ačkoliv někdy může odhalit nevyřčené předpoklady a dokonce rozdílné názory na charakter projektu, což je ale naprosto v pořádku. Je lepší, když se tyto věci začnou řešit teď než později. Dalším krokem je zabalení projektu tak, aby byl stravitelný pro veřejnost, což je po pravdě řečeno čistá otročina.

Tato práce je tolik únavná proto, že spočívá zejména v uspořádávání a zdokumentování věci, které už dávno všichni vědí – tedy všichni, kteří už jsou v projektu zapojeni. Takže pro ty, kteří tuto práci dělají, z ní neplyne žádný bezprostřední užitek. Nepotřebují soubor README, který podává přehled projektu, nepotřebují ani dokument popisující jeho návrh nebo uživatelskou příručku. Nepotřebují mít

pečlivě uspořádaný strom zdrojového kódu, mají svou neformální strukturu, ale jejich zdrojový kód by měl odpovídat rozšířeným standardům distribuce zdrojového kódu. Stejně tak dobře jim poslouží i jakékoliv jiné uspořádání zdrojového kódu, protože už si na ně stejně zvykli; pokud kód lze spustit, vědí, jak jej používat. Dokonce je jim úplně jedno, že nejsou zdokumentovány základní principy architektury projektu, protože ty už také znají.

Jenže nově příchozí tyto věci naopak potřebují. Naštěstí je ale nepotřebují všechny najednou. Není nutné, abyste před zveřejněním projektu sepsali úplně všechno. V dokonalém světě by možná každý nový open source projekt zahájil svou existenci s pečlivě propracovaným dokumentem popisujícím jeho návrh, úplnou uživatelskou příručkou (kde bude zvlášť vyznačeno, které funkce se teprve plánují, ale ještě nejsou implementovány), bezvadně optimalizovaným a přenositelným kódem, který běží na všech možných platformách atd. V reálu by ale něco takového zabralo tak obrovské množství práce, že se do toho nikdo nepustí, zejména když se dá celkem rozumně očekávat, že vám s tím po rozběhnutí projektu pomohou dobrovolníci.

Prezentaci musíte zkrátka věnovat tolik času, aby se nově příchozí účastníci dokázali v projektu zorientovat. Berte to jako jakýsi první krok zaváděcího procesu, kdy je do projektu nutné vložit určitou minimální aktivační energii. Setkal jsem se u tohoto konceptu s označením *hacktivační energie* (hacktivation energy), tedy množství energie, které musí nováček vynaložit předtím, než začne získávat něco zpět. Čím je tato míra u vašeho projektu nižší, tím lépe. Vaším prvním úkolem je snížit tuto vyžadovanou vstupní energii na úroveň, která nebude ostatním bránit v tom, aby se zapojili.

Každý z následujících oddílů popisuje jeden důležitý aspekt zahájení nového projektu. Jsou uvedeny přibližně v tom pořadí, v němž se s nimi nový návštěvník bude setkávat; pořadí, v němž je budete uskutečňovat, může být samozřejmě jiné. Berte ho jako takový seznam úkolů. Při zahájení projektu projděte tyto body jeden za druhým a ujistěte se, že jsou všechny splněny, popřípadě že jste si vědomi toho, jaký dopad může mít, když nějaký přeskočíte.

Vyberte dobré jméno

Představte si, že jste v kůži někoho, kdo se o vašem projektu právě dozvěděl – třeba na něj narazil, když hledal software, jenž by vyřešil nějaký konkrétní problém. První věc, s kterou se setká, je jméno projektu.

Dobré jméno vašemu projektu úspěch samozřejmě nezajistí a asi těžko najdete projekt, který by selhal jenom proto, že měl špatný název – umím si sice představit jména, která by to patrně dokázala, ale vycházejme z předpokladu, že se nikdo nebude aktivně snažit svůj projekt potopit. Rozhodně ale platí, že špatné jméno může přijetí projektu výrazně zpomalit – buď proto, že jej lidé nebudou brát vážně, nebo proto, že budou mít problémy si ho vůbec zapamatovat.

Dobré jméno:

- Udělejte si představu o tom, co váš projekt přináší, nebo s čím nějak souvisí. Takže když víme co projekt přinese, jméno už si vybavíme mnohem snáz.
- Je snadno zapamatovatelné. Tady se nemůžete vyhnout skutečnosti, že standardním jazykem internetu je angličtina, takže „snadno zapamatovatelné“ zde znamená „snadno zapamatovatelné pro někoho, kdo čte anglicky“. Například jména tvořená slovní hříčkou opírající se o výslovnost rodilého mluvčího budou pro mnohé čtenáře, pro něž není angličtina mateřský jazyk, zcela nepochopitelná. Pokud je vaše hříčka opravdu zajímavá a zapamatovatelná, může samozřejmě stát za to ji použít, ale nezapomínejte přitom, že mnozí budou jméno projektu vyslovovat úplně jinak, než jak by jej četl rodilý mluvčí.
- Neshoduje se se jménem nějakého jiného projektu a neporušuje žádné obchodní známky. To patří k dobrým mravům, nemluvě o tom, že z právního hlediska je to rozumné. Navíc nechcete, aby se vaše jméno s něčím pletlo. Sledovat vše, co je na internetu k dispozici, je už tak dost náročné, natož kdyby se různé věci jmenovaly stejně.
- Zdroje, o kterých jsem se zmínil dříve v části **Nejdříve se ale rozhlédněte**, vám pomohou zjistit, zda už vámi zamýšlené jméno nepoužívá někdo jiný. Bezplatné hledání v obchodních ochranných známkách nabízejí servery **http://www.nameprotect.org/** a **http://www.uspto.gov/**.
- Pokud je to možné, je dostupné jako doménové jméno druhé úrovně v doménách `.com`, `.net` a `.org`. Z nich byste si měli vybrat jedno (pravděpodobně `.org`), které budete uvádět jako oficiální domácí místo projektu. Zbylá dvě doménová jména by sem měla přesměrovat – registrují se pouze proto, aby nemohly třetí strany vytvářet zmatek kolem jména projektu. Dokonce i když máte v úmyslu projekt hostovat na serveru někoho jiného (viz **Kompletní hosting**), můžete si stejně zaregistrovat domény odpovídající jménu projektu a přesměrovat je na hostitelský server. Pokud má projekt jednoduché URL, velmi to uživatelům usnadní jeho zapamatování.

Určete jasné cíle projektu

Když lidé najdou webové stránky projektu, začnou na nich hledat nějaký stručný popis, oznámení cílů projektu, aby se mohli (během ne víc než 30 vteřin) rozhodnout, zda mají zájem dozvědět se víc. Takový text by měl být umístěn na nějakém výrazném místě na hlavní stránce, ideálně hned pod jménem projektu.

Cíle projektu by měly být velmi konkrétní, přesně vymezené a především stručné. Tady máme pěkný příklad z **http://www.openoffice.org/:**

> *V rámci komunity vytvořit špičkový, mezinárodně použitelný balík kancelářských aplikací, který poběží na všech hlavních platformách a zpřístupní veškerou svou funkčnost a data prostřednictvím API založených na otevřených komponentech a s použitím formátu souborů založeného na XML.*

Několika slovy tady zachytili všechny hlavní body, přičemž do velké míry spoléhají na předchozí znalosti svých čtenářů. Tím, že napsali *„v rámci komunity“*, dávají najevo, že vývoj nebude ovládat žádná společnost; slovy *„mezinárodně použitelný“* říkají, že software lidem umožní pracovat ve více jazycích a v různých lokalizacích. Spojení *„na všech hlavních platformách“* pak znamená, že software bude spustitelný v systémech Unix, Macintosh a Windows. Zbytek naznačuje, že důležitou součást cílů tvoří použití otevřených rozhraní a snadno srozumitelných formátů souborů. Neříkají zde otevřeně, že chtějí být svobodnou alternativou k Microsoft Office, ale většina lidí to pravděpodobně vyčte mezi řádky. Ačkoliv tento popis cílů projektu vypadá na první pohled dost rozmáchle, ve skutečnosti je celkem dobře vymezený – slova *„balík kancelářských aplikací“* představují pro ty, kteří tento typ softwaru znají, něco velmi konkrétního. Zde se opět využívají předpokládané dřívější znalosti čtenáře (v tomto případě pravděpodobně získané v MS Office) k tomu, aby cíle projektu zůstaly co nejstručnější.

Povaha takovýchto deklarací závisí částečně na tom, kdo je sepsal, a ne pouze na softwaru, který popisují. Například pro OpenOffice.org má smysl použít slova *„v rámci komunity“*, protože projekt byl zahájen a je stále do značné míry sponzorován firmou Sun Microsystems. Přidáním těchto slov firma Sun ukazuje, že si uvědomuje, že mezi ostatními mohou panovat obavy o to, zda se nebude pokoušet vývoj řídit. Ovšem tím, že tuto výtku předvídá už zde na samém začátku, podniká významný krok k tomu, aby se projekt nutnosti tento problém řešit úplně vyhnul. Na druhou stranu projekty, které nejsou sponzorovány žádnou společností, pravděpodobně nic takového říkat nepotřebují – vývoj probíhající komunitním způsobem je v open source světě normální, takže obvykle není žádný důvod jej zde výslovně zmiňovat.

Uveďte, že jde o svobodný projekt

Ti, u nichž zájem přetrvá i po přečtení cílů projektu, se budou chtít dozvědět další podrobnosti, třeba si prohlédnout uživatelskou nebo vývojářskou dokumentaci; možná si budou chtít i něco stáhnout. Ale ještě předtím se budou chtít ujistit, že jde o open source.

Hlavní stránka musí dávat jednoznačně a jasně najevo, že jde o open source projekt. Může vám to připadat jako samozřejmost, ale byli byste překvapeni, kolik projektů na to zapomíná. Viděl jsem webové prezentace projektů svobodného softwaru, na jejichž hlavní stránce nejen že jste nenašli vůbec nic o tom, jakou konkrétní svobodnou licencí se distribuce softwaru řídí, ale kde navíc nebylo ani zmínky o tom, že jde o svobodný software. V některých případech byly tyto zásadní informace vykázány na stránku s odkazy ke stažení či na stránku pro vývojáře, popřípadě na jiné místo, kam se dalo dostat až po dalším kliknutí myši. V extrémních případech nebyla licence na webu k nalezení vůbec – jediný způsob, jak se k ní dostat, bylo software stáhnout a podívat se dovnitř.

Tuto chybu nedělejte. Podobné opomenutí může vést ke ztrátě mnoha potenciálních vývojářů a uživatelů. Hned pod cílem projektu rovnou uveďte, že jde o „svobodný software“ nebo „open source software“ a jakou licenci přesně používá. Stručného průvodce výběrem licence najdete v sekci označené **„Výběr licence a její uplatnění“** dále v této kapitole. Problémy s licencováním se budeme podrobněji zabývat v kapitole **9. Licence, autorská práva a patenty**.

V tento moment už se náš hypotetický návštěvník rozhodl (a trvalo mu to zatím zhruba minutu nebo méně), že by měl zájem věnovat prozkoumání tohoto projektu řekněme dalších pět minut. Následující oddíl popíše, s čím by se během těchto pěti minut měl setkat.

Seznam funkcí a požadavků

Měli byste také uvést krátký seznam funkcí a vlastností, které váš software má (pokud něco ještě není dokončené, uveďte to také, ale připište k tomu *„plánuje se“* nebo *„ve vývoji“*), a jaké výpočetní prostředí ke svému běhu potřebuje. Tento seznam berte jako souhrn informací, které byste poskytli někomu, kdo vás požádá o stručnou charakteristiku vašeho softwaru. Často bude logicky vyplývat z cílů projektu. Řekněme, že je cíl projektu formulován například takto:

> *Vytvořit fulltextový indexovací a vyhledávací nástroj s bohatým API, který by mohli programátoři využít při poskytování vyhledávacích služeb ve větším množství textových souborů.*

Seznam funkcí a požadavků by pak uváděl podrobnosti, které tento cíl popíšou trochu podrobněji:

Vlastnosti:
- *Prohledávání souborů ve formátu prostého textu, HTML a XML*
- *Vyhledávání slov a frází*
- *(plánuje se) Vyhledávání částečných shod (Fuzzy matching)*
- *(plánuje se) Inkrementální aktualizace indexů*
- *(plánuje se) Indexování vzdálených webových serverů*

Požadavky:
- *Python 2.2 nebo vyšší*
- *Dostatečně velký diskový prostor pro uložení indexů (přibližně dvojnásobek velikosti zdrojových dat)*

Na základě těchto informací mohou čtenáři rychle zjistit, zda by jim váš software vyhovoval, nebo zda by se do něj nechtěli zapojit i jako vývojáři.

Stav vývoje

Lidé se vždy zajímají o to, jak si projekt vede. U nových projektů chtějí vědět, jaký je rozdíl mezi tím, co projekt slibuje do budoucna a co už v současnosti umí. U vyzrálých projektů pak chtějí vědět, jak aktivně je projekt udržován, jak často jsou zveřejňovány nové verze, jak pružně asi bude reagovat na hlášení chyb atd.

Jako odpověď na tyto otázky byste měli poskytnout webovou stránku popisující stav vývoje, na níž bude uveden seznam krátkodobých cílů a potřeb projektu (můžete například právě hledat vývojáře

s nějakou specializací). Stránka může obsahovat i historii minulých verzí se seznamy jejich funkcí, takže si návštěvníci mohou udělat představu o tom, jak si váš projekt pro sebe definuje „pokrok“ a jak rychle takové pokroky dělá.

Nebojte se toho, že váš projekt bude vypadat nehotově, a odolejte pokušení stav vývoje přechválit. Všichni vědí, že software se vyvíjí postupně, a není vůbec žádná ostuda říct „Toto je alfa verze a je v ní několik známých chyb. Většinu času sice běží a funguje dobře, ale používejte ji jen na vlastní riziko.“ Vývojáře, které v této fázi vývoje potřebujete, tím neodradíte. A co se týká uživatelů, jednou z nejhorších věcí, které můžete v projektu udělat, je přilákat nové uživatele ještě předtím, než na ně software bude připraven. Jak jednou váš software získá pověst, že je nestabilní a plný chyb, bude se jí jen těžko zbavovat. Z dlouhodobého hlediska se vyplatí být spíš opatrnější. Pro software je vždy lepší, když se ukáže jako stabilnější, než uživatelé čekali – pokud je váš produkt příjemně překvapí, budou jej tím spíše doporučovat i ostatním.

Alfa a beta

Pojmem *alfa* se obvykle označuje první verze, s níž už mohou uživatelé plnohodnotně pracovat a která už obsahuje všechny zamýšlené funkce, ale ve které jsou i známé chyby. Hlavním účelem verze alfa je získání zpětné vazby, aby vývojáři věděli, na co se mají zaměřit. Další etapa, *beta*, popisuje takový stav, kdy už byly všechny závažné chyby opraveny, ale kdy ještě nebyl software dostatečně otestován na to, aby mohl být schválen pro oficiální vydání. Účelem betaverze je buď to, aby se v případě, že se v ní už žádné chyby nenajdou, stala oficiálním vydáním (release), nebo aby mohli vývojáři získat podrobnou zpětnou vazbu a chyby rychleji opravit. Rozdíl mezi verzemi alfa a beta ale není stanoven příliš pevně a konkrétní projekty si jej mohou určit do značné míry samy.

Stahování

Software by mělo být možné stáhnout v podobě zdrojového kódu ve standardních formátech. Pokud byl projekt založen teprve nedávno, není nutné, aby byly k dispozici i binární (spustitelné) balíčky, s výjimkou případů, kdy má software tak složité požadavky na sestavení nebo tak komplikované závislosti, že by většině lidí dalo spoustu práce, než by jej do spustitelného stavu dostali. (Takový projekt ale bude mimochodem shánět nové vývojáře jen velmi těžko.)

Mechanismus distribuce by měl být pohodlný, standardní a co nejméně komplikovaný. Pokud byste se snažili vymýtit nějakou nemoc, také byste léky nedistribuovali tak, aby k jejich aplikaci byla potřeba řekněme neobvykle velká injekční stříkačka. Podobně by se měl i software podřídit standardním metodám sestavení a instalace. Čím víc se bude od standardů odchylovat, tím víc potenciálních uživatelů a vývojářů to po chvíli vzdá a raději odejde.

Může to znít jako samozřejmost, ale mnohé projekty se tím, že by instalační postup nějak standardizovaly, vůbec nezatěžují a celou práci odkládají až na poslední chvíli s tím, že to je přece něco, co se

dá dodělat kdykoliv: *„Všechny tyhle věci vyřešíme, až bude náš kód před dokončením.“* Neuvědomují si ale, že když tuto nezáživnou práci s dokončováním sestavovacích a instalačních procedur odkládají, prodlužují tím i dobu, než bude kód dokončen, protože odrazují vývojáře, kteří by jinak k vývoji přispěli. Nejzáludnější na celé věci je, že si přitom vůbec neuvědomují, že o potenciální vývojáře přicházejí, protože celý tento proces sestává z událostí, které nejsou nikde zaznamenány: někdo navštíví webové stránky, stáhne si software, zkusí ho sestavit, nepovede se mu to, vzdá to a jde pryč. To se ale vůbec nikdo nedozví – kromě dotyčné osoby samotné. Nikdo z účastníků projektu nezjistí, že zde byl někdo, kdo měl o projekt zájem a dobrou vůli se přidat, ale kdo zase potichu zmizel.

Nudná práce, která se může hodně vyplatit, by se vždy měla udělat co nejdřív. Významné snížení vstupní bariéry projektu spočívající ve vytváření dobrých instalačních balíčků je přesně případ něčeho, co se vám oplatí velmi bohatě.

Pokud vydáte stažitelný balíček, je důležité, abyste mu přidělili unikátní číslo verze, aby mohl každý snadno srovnat dva různé releasy a poznat, který z nich je novější. Podrobnosti o číslování verzí naleznete v **Číslování verzí**; více o tom, jak vypadá standardní postup sestavování a instalace, najdete v části **Vytváření balíčků** – obojí viz kapitola **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji.**

Zpřístupnění systémů správy verzí a sledování chyb

Možnost stažení zdrojových balíčků bude vyhovovat těm, kdo software chtějí jen nainstalovat a používat, ale nebude stačit těm, kteří by jej chtěli ladit nebo přidávat nové funkce. Zde sice mohou pomoci pravidelné noční snímky (Nightly source snapshots) zdrojového kódu, ale ty jsou pro rozvíjející se vývojářskou komunitu pořád ještě příliš hrubé. Vývojářům musíte poskytnout přístup k nejnovějšímu zdrojovému kódu v reálném čase, což se dá zajistit použitím systému pro správu verzí (version control system). Tím, že svůj kód budete spravovat systémem pro řízení verzí a zpřístupníte jej nejširší veřejnosti, vysíláte jak uživatelům, tak vývojářům jasný signál, že se snažíte dát jim vše, co je k účasti na projektu třeba. Pokud nemůžete systém pro správu verzí nabídnout hned, alespoň někde umístěte oznámení, že ho hodláte brzy zavést. Infrastrukturou správy verzí se podrobně zabývá **Správa verzí** v kapitole **3. Technická infrastruktura**.

Totéž platí pro systém sledování chyb projektu (bug tracker). Význam systému pro sledování chyb nespočívá jen v jeho užitečnosti pro vývojáře, ale také v tom, co naznačuje vnějším pozorovatelům projektu. Pro mnoho lidí je zpřístupnění databáze chyb jednou z nejvýraznějších známek toho, že by projekt měl být brán vážně. Kromě toho, čím větší počet chyb v databázi je, tím lépe projekt vypadá. Tohle tvrzení sice na první pohled nedává moc smysl, ale začne, pokud si uvědomíte, že počet zaznamenaných chyb závisí na třech faktorech: na absolutním počtu chyb, které se v softwaru nacházejí, na počtu jeho uživatelů a na tom, jak obtížné je zapsat do systému novou chybu. Z těchto tří faktorů jsou ty druhé dva významnější než ten první. Jakýkoliv software, který je dostatečně veliký a složitý, obsahuje určité, v zásadě libovolné množství chyb, jež čekají na to, až budou odhaleny. Otázkou tedy

spíše je, jak dobře projekt zvládá jejich zaznamenávání a jak přiděluje prioritu jejich řešení. Projekt, který má velkou a dobře udržovanou databázi chyb (tím rozumím to, že na nové chyby rychle reaguje, že duplicitně nahlášené chyby slučuje dohromady atd.), proto působí lepším dojmem než projekt, který žádnou databázi chyb nemá nebo ji má téměř prázdnou.

Pokud byl váš projekt založen teprve nedávno, pak bude samozřejmě databáze obsahovat jen velmi málo chyb, s čímž toho pochopitelně moc nenaděláte. Ale pokud na stránce o stavu vývoje zdůrazníte, že jde o velmi mladý projekt, a pokud lidé při pohledu do databáze chyb uvidí, že většina záznamů byla vložena teprve nedávno, mohou si z toho odvodit, že jsou nové chyby přesto hlášeny tempem, které ukazuje na zdravý růst – a tím, že je zaznamenaných chyb zatím jen málo, se nebudou rozrušovat.

Nezapomínejte, že systémy pro sledování chyb se často používají nejen k zaznamenávání softwarových chyb jako takových, ale také ke sledování požadavků na zlepšení, změn v dokumentaci, nevyřízených úkolů a k dalším účelům. Podrobnosti o tom, jak provozovat systém pro sledování chyb, naleznete v části **Systém pro sledování chyb** v kapitole **3. Technická infrastruktura**, takže zde se jimi dále zabývat nebudeme. Z hlediska prezentace je důležité, aby projekt nějaký systém pro sledování chyb vůbec měl a aby tato informace byla na jeho hlavní stránce uvedena.

Komunikační kanály

Návštěvníci obvykle chtějí vědět, jak se dostanou k lidem, kteří jsou do projektu zapojení. Poskytněte jim tedy adresy mailing listů, chatovacích místností, kanálů IRC a všech dalších fór, kde se mohou spojit s těmi, kteří se softwarem mají něco společného. Nezapomeňte zdůraznit, že vy osobně i všichni ostatní autoři projektu jste do těchto mailing listů přihlášeni, aby každý věděl, že zpětná vazba, kterou sem napíše, se k vývojářům vždy dostane. Z vaší přítomnosti v seznamu adresátů ale nevyplývá závazek odpovídat na všechny otázky nebo implementovat všechny požadované funkce. Z dlouhodobého hlediska se většina uživatelů do fóra stejně nikdy nezapojí, ale ocení, když budou vědět, že tuto možnost mají, pokud by ji někdy potřebovali.

V raných etapách projektu není nutné zřizovat oddělená fóra pro uživatele a pro vývojáře. Mnohem lepší je, když spolu budou všichni zúčastnění mluvit v jedné „místnosti“. Ze začátku je rozdíl mezi vývojáři a uživateli často dost nejasný; i pokud by se dala mezi těmito skupinami jednoznačně vést hranice, pak bývá v počátcích projektu poměr vývojářů k uživatelům obvykle mnohem vyšší než později. I když nemůžete předpokládat, že by byl každý raný účastník programátorem, který by se chtěl na vývoji softwaru podílet, můžete očekávat, že mají přinejmenším zájem sledovat diskuzi kolem vývoje a zjistit, kam bude projekt nasměrován.

Protože se tato kapitola zabývá pouze spuštěním projektu, řekneme si prozatím jen to, že by tato komunikační fóra měla existovat. Později (v části **Jak se vyrovnat s růstem** v kapitole **6. Komunikace**) se podíváme i na to, kde a jak taková fóra vytvořit a zda budou potřebovat moderátory nebo nějakou jinou formu správy. Probereme i to, jakým způsobem a kdy nejlépe oddělit uživatelská fóra od vývojářských, aniž bychom přitom vytvořili nepřekonatelnou propast.

Pokyny pro vývojáře

Pokud někdo bude uvažovat o tom, že by se mohl na projektu podílet, poohlédne se nejprve po pokynech pro vývojáře. Tyto pokyny nejsou ani tak technického jako spíš společenského rázu. Vysvětlují, jakým způsobem vývojáři komunikují mezi sebou a s uživateli a také jak vlastně celá práce probíhá.

Tímto tématem se podrobně zabývá **Jak to všechno zapsat** v kapitole **4. Společenská a politická infrastruktura**, ale už zde můžeme uvést, že tyto pokyny by měly obsahovat:

- odkazy na fóra pro komunikaci s ostatními vývojáři
- instrukce, jak ohlašovat chyby a zasílat záplaty (patche)
- alespoň rámcový popis toho, jakým způsobem probíhá vývoj – tedy zda je projekt řízen formou benevolentní diktatury, demokracie nebo něčeho jiného

Slovo „diktatura“ zde mimochodem není myšleno nijak hanlivě. Označuje se tím druh tyranie, kde má jeden konkrétní vývojář na všechny změny právo veta, což se považuje za naprosto korektní postup. Tímto způsobem funguje celá řada úspěšných projektů. Důležité ale je, aby to projekt oznámil hned na začátku a na rovinu. Diktatura, která předstírá, že je demokracií, lidi odradí; tyranie, která o sobě otevřeně přiznává, že je tyranií, bude fungovat, pokud je tyran schopný a důvěryhodný.

Jako příklad mimořádně pečlivých pokynů pro vývojáře si prohlédněte **http://subversion.apache.org/docs/community-guide/**; obecnější pokyny, které se zaměřují více na vedení projektu a na celkový charakter spoluúčasti než na technické věci, najdete na **http://www.openoffice.org/dev_docs/guidelines.html.**

To, jak software co nejlépe představit novým programátorům, probereme v části **Dokumentace pro vývojáře** dále v této kapitole.

Dokumentace

Dokumentace je naprosto nezbytná. Vždy musí existovat něco, co si lidé mohou přečíst, i kdyby to mělo být jen stručné a neúplné. Práce na dokumentaci je přesně takovou nepopulární činností, o jakých jsme mluvili dříve, a je to často ta první oblast, v níž nový open source projekt pohoří. Sepsání cílů projektu a seznamu jeho funkcí, výběr licence a poskytování informací o stavu vývoje jsou všechno poměrně jednoduché úkoly, které lze dostat do definitivního stavu, na nějž už obvykle není nutné dále sahat. Dokumentace je ale něco, co vlastně nikdy tak úplně hotové není, což může být jeden z důvodů, proč její sepisování lidé často odkládají.

Nejzákeřnější věcí je, že užitečnost dokumentace pro ty, kteří ji píší, je přesně opačná než pro ty, kteří ji budou číst. Pro uživatele-začátečníky jsou v dokumentaci nejdůležitější úplné základy: jak lze software rychle uvést do provozu, přehled toho, jak funguje, možná pár návodů, jak provést běžné úkoly. To jsou ale přesně ty věci, které tvůrci dokumentace znají až příliš dobře – natolik dobře, že pro ně může být

obtížné podívat se na ně očima čtenáře a pracně popisovat kroky, které jim (tedy autorům dokumentace) připadají natolik samozřejmé, že nemá cenu se o nich vůbec zmiňovat.

Pro tento problém neexistuje žádné kouzelné řešení – někdo si zkrátka musí sednout a dokumentaci napsat. Její kvalitu je pak ještě vhodné prověřit tím, že ji předložíte typickým novým uživatelům. Použijte jednoduchý formát, který se snadno upravuje, jako například HTML, prostý text, Texinfo nebo nějakou variantu XML – něco, co je možné jednoduše a rychle pozměnit, kdykoliv to bude potřeba. To proto, abyste zbytečně nekomplikovali život ani původním autorům textu, kteří jej budou chtít postupně vylepšovat, ani těm, kteří se k projektu připojí později a chtěli by na dokumentaci pracovat.

Jeden ze způsobů, jak zajistit, aby alespoň ta nejzákladnější dokumentace vznikla včas, je předem vymezit její rozsah. Díky tomu alespoň nebudou mít její autoři pocit, že je jejich práce nekonečná. Osvědčeným pravidlem je, že by dokumentace měla splňovat alespoň následující kritéria:

- Čtenářům jasně řekněte, jak velké odborné znalosti se u nich očekávají.
- Srozumitelně a podrobně popište, jak se software uvede do provozu; někde na začátku dokumentace uživatelům řekněte, jak provést diagnostické testy nebo spustit jednoduché příkazy, kterými by si mohli ověřit, že vše nastavili správně. Dokumentace týkající se uvedení softwaru do provozu je svým způsobem důležitější než ta, v níž se popisuje, jak software používat. Čím více úsilí někdo bude věnovat instalaci a zprovoznění softwaru, tím vytrvalejší bude při prozkoumávání pokročilých funkcí, které nejsou moc dobře zdokumentované. Pokud to vaši potenciální uživatelé vzdají, bude to někdy ze začátku; právě proto byste měli nejvíc podpory soustředit na počáteční fáze, jako je například instalace.
- Uveďte příklad, jak provést nějakou běžnou úlohu krok za krokem. Samozřejmě čím víc příkladů, tím lépe, ale pokud jste omezeni časem, vyberte si jen jednu věc a popište ji do posledních podrobností. Jakmile někdo zjistí, že se váš software dá použít pro jednu věc, začne sám zkoumat, co dalšího ještě zvládne. A pokud máte štěstí, začnou vaši uživatelé dokumentaci doplňovat sami. Což nás přivádí k dalšímu bodu…
- Označte části dokumentace, o kterých víte, že jsou neúplné. Když přiznáte, že jste si nedostatků dokumentace vědomi, ukážete čtenářům, že chápete jejich situaci. To je uklidní, protože pak budou vědět, že není potřeba někoho přesvědčovat o tom, co je v projektu důležité. V takto označených částech nemusíte nutně slíbit, že nedostatky k určitému datu napravíte – můžete je oprávněně považovat za žádost o pomoc ze strany dobrovolníků.

Tento poslední bod je mnohem důležitější, než na první pohled vypadá, protože se dá se uplatnit na celý projekt, nejen na dokumentaci. To, že uvedete přesný seznam známých nedostatků, je ve světě open source standardem. Na tyto nedokonalosti nemusíte nějak přehnaně poukazovat – prostě je na vhodném místě zaznamenejte (tedy v dokumentaci, v databázi chyb (bug trackeru), při diskuzi v mailing listu a podobně), svědomitě a bez jakýchkoliv emocí. Nikdo to nebude brát ani jako přiznání prohry, ani jako závazek vyřešit problém k určitému datu – pokud tedy takový závazek nebude explicitně vysloven. Protože každý, kdo software používá, přijde na tyto nedostatky časem sám, bude mnohem lepší, když na ně bude psychicky připravený. Projekt pak navíc působí dojmem, že si dobře uvědomuje, jak na tom přesně je.

Údržba FAQ

Dokument FAQ („Frequently Asked Questions“ čili často kladené otázky) může být pro šíření informací o projektu jednou z těch vůbec nejlepších investic. FAQ jsou přesně zacíleny právě na ty otázky, které uživatelé a vývojáři opravdu pokládají (tedy ne na otázky, jejichž výskyt byste možná očekávali). Dobře vedený dokument FAQ proto těm, kteří do něj nahlédnou, často poskytne přesně to, co hledají. Právě FAQ je často tím prvním místem, kam se uživatelé dívají, když narazí na nějaký problém, obvykle ještě dříve, než zkusí hledat v oficiálním manuálu; ve vašem projektu to pravděpodobně bude i tento dokument, na nějž budou jiné servery nejčastěji odkazovat.

FAQ ale bohužel nelze vytvořit hned na začátku projektu. Opravdu dobrý dokument FAQ nemůže být jen tak napsán, musí se vypěstovat. Už z definice jde o text, který na něco reaguje; vytváří se s postupem času jako odezva na každodenní používání softwaru. Protože není možné zcela přesně předpovědět, jaké otázky budou lidé skutečně pokládat, není ani možné si jednoduše sednout a napsat užitečný FAQ jen tak z ničeho.

Takové pokusy jsou jen ztrátou času. Může být užitečné, když ze začátku vytvoříte kostru pro FAQ, která bude na většině míst prázdná; zajistíte tím, že až se projekt rozběhne, bude existovat místo, kam lze připisovat otázky a odpovědi. V této etapě není nejdůležitější úplnost, ale jednoduchost. Pokud půjde do dokumentu FAQ snadno něco přidat, poroste časem sám. (Kvalitní údržba FAQ je úkolem nelehkým a značně spletitým; více se mu budeme věnovat v části **FAQ Manager (manažer sekce s nejčastěji kladenými otázkami)** v kapitole **8. Řízení dobrovolníků.**)

Dostupnost dokumentace

Dokumentace by měla být dostupná na dvou místech: on-line (přímo z webových stránek) a ve stažené distribuci softwaru (viz **Vytváření balíčků** v kapitole **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji**). On-line a v podobě zobrazitelné webovým prohlížečem musí být dostupná proto, že dokumentaci uživatelé často čtou ještě předtím, než software poprvé stáhnou – až podle ní se rozhodnou, zda o něj vůbec stojí. Měla by ale být přibalena i k softwaru samotnému na základě principu, že stažením jediného balíčku byste měli získat (a mít lokálně přístupné) vše, co budete potřebovat.

On-line verze by měla obsahovat i odkaz na úplnou dokumentaci v podobě jediné HTML stránky (připojte k odkazu ke stažení i nějakou poznámku jako například „vše v jednom“ nebo „jedna velká stránka“, abyste upozornili na to, že její stahování může nějakou dobu trvat). Tato forma je užitečná proto, že uživatelé často chtějí v celé dokumentaci vyhledat konkrétní slovo nebo frázi. Obvykle je to proto, že už vědí, co přesně hledají, jenom si nemohou vzpomenout, v které části to je. Pro takové uživatele je velmi frustrující, když najdou jednu HTML stránku s obsahem, pak další, která obsahuje úvod, další s pokyny pro instalaci atd. Pokud je celá dokumentace takto rozkouskovaná, bude jim vyhledávací funkce jejich prohlížeče k ničemu. Rozdělení do samostatných stránek je užitečné pro ty, kdo už vědí, jakou část chtějí číst, nebo pro ty, kdo si chtějí celou dokumentaci přečíst od začátku do konce. To ale není ten nejběžnější způsob, jakým se dokumentace používá. Daleko častěji ji čtou ti, kteří už daný

software v podstatě znají, ale potřebují si něco konkrétního ověřit nebo najít. Pokud jim nedáte k dispozici jediný dokument, ve kterém lze jednoduše vyhledávat, zbytečně jim zkomplikujete život.

Dokumentace pro vývojáře

Vývojářská dokumentace je psaná proto, aby programátorům pomohla porozumět kódu, díky čemuž jej pak budou moci snadno opravovat a rozšiřovat. Je to něco jiného než pokyny pro vývojáře, o kterých jsme hovořili dříve – ty jsou spíš společenského než technického charakteru. Pokyny pro vývojáře programátorům říkají, jak se mají chovat k sobě navzájem; vývojářská dokumentace jim říká, jak se chovat ke zdrojovému kódu samotnému. Oba zmíněné dokumenty se pro zjednodušení často slučují do jednoho (jako tomu je v příkladu **http://subversion.apache.org/docs/community-guide/**, který jsme zmínili dříve), ale není to nutné.

Dokumentace pro vývojáře může být velmi užitečná, ale není žádný důvod k tomu, abychom odkládali zveřejnění softwaru jen kvůli ní. Dokud jsou původní autoři dostupní a ochotní odpovídat na otázky týkající se kódu, pak to pro začátek úplně stačí. Nejčastější motivací k sepsání dokumentace je ostatně právě to, že vývojáři musí neustále odpovídat na stejné dotazy. Přispěvatelé, kteří jsou odhodláni se k projektu přidat, si ale obvykle najdou způsob, jak si s kódem poradit i bez dokumentace. Důvod, proč lidé tráví svůj čas studiem zdrojového kódu, je to, že tento kód dělá něco, co je pro ně užitečné. Pokud budou věřit, že je k něčemu dobrý, najdou si čas na to, aby zjistili, jak funguje; pokud jim tato víra schází, pak je žádná vývojářská dokumentace nepřesvědčí ani neudrží.

Takže pokud máte čas na to, sepsat dokumentaci jen pro jednu skupinu, napište ji raději pro uživatele. Koneckonců veškerá uživatelská dokumentace je zároveň i dokumentací pro vývojáře. Každý programátor, který má na softwaru začít pracovat, musí vědět, jak jej používat. Když později uvidíte, že vaši programátoři kladou pořád dokola stejné otázky, vyhraďte si čas na to, abyste napsali zvláštní dokumentaci i pro ně.

Některé projekty používají pro počáteční dokumentaci stránky spravované systémem wiki; někdy je stejným způsobem vedena veškerá dokumentace projektu. Podle mých zkušeností tento způsob opravdu funguje jen tehdy, když tyto wiki stránky vytváří pár lidí, kteří se shodnou na tom, jak by měla být dokumentace organizována a jaký styl by měla používat. Další informace najdete v části **Wiki** v kapitole **3. Technická infrastruktura**.

Vzorový výstup a snímky obrazovky

Pokud má projekt grafické uživatelské rozhraní nebo pokud produkuje grafický nebo jinak výrazný výstup, umístěte na webové stránky pár příkladů. V případě rozhraní to budou snímky obrazovky (screenshoty), v případě výstupu to mohou být rovněž snímky obrazovky nebo prostě soubory. Obojí nasytí lidskou potřebu po okamžitém uspokojení. Jediný snímek obrazovky může být přesvědčivější než celý odstavec textového popisu nebo klábosení v mailing listu, a to čistě proto, že snímek obrazovky je

nezvratným důkazem, že váš software funguje. Může obsahovat chyby, může být obtížné jej nainstalovat, může mít neúplnou dokumentaci, ale snímek obrazovky pořád ukazuje, že pokud uživatel vyvine patřičné úsilí, je možné software zprovoznit.

Snímky obrazovky

Těm, kteří to nikdy nedělali, může pořizování snímků obrazovky připadat složité, takže uvedu stručný návod. Při použití programu Gimp (**http://www.gimp.org/**) otevřete Soubor->Vytvořit->Snímek obrazovky, vyberte Zachytit jedno okno nebo Zachytit celou obrazovku a klikněte na OK. Podle vašeho výběru se buď zachytí celá obrazovka nebo zvolené okno a výsledek se v Gimpu objeví jako obrázek. Podle potřeby obrázek ořežte a upravte jeho velikost podle pokynů, které najdete na **http://www.gimp.org/tutorials/Lite_Quickies/#crop.**

Na webový server projektu můžete umístit i celou řadu dalších věcí, pokud na to máte čas nebo pokud je to z nějakého důvodu zvlášť vhodné: stránku s novinkami, stránku s historií projektu, stránku se souvisejícími odkazy, funkci pro vyhledávání na celém serveru vašeho projektu, odkazy pro dárce atd. V době zahájení projektu nejsou tyto věci nezbytné, ale v budoucnosti se vám můžou hodit.

Kompletní hosting

Existuje několik serverů, které zdarma nabízejí kompletní hosting a infrastrukturu pro open source projekty: webový prostor, systém pro správu verzí, systém pro sledování chyb, prostor pro stahování, diskuzní fóra, pravidelné zálohování atd. V podrobnostech se od sebe liší, ale základní služby poskytují všechny zhruba stejné. Když jednoho z těchto serverů využijete, získáte bez práce spoustu věcí; to, čeho se ale musíte vzdát, je plný vliv na to, jak budete působit na své uživatele. Je to správce hostingové služby, kdo rozhoduje, jaký software na serveru poběží, což může zásadním způsobem ovlivnit celkový dojem z webových stránek projektu.

Podrobnější diskuzi o výhodách a nevýhodách kompletního hostingu a seznam serverů, které jej nabízejí, najdete v části **Kompletní hosting** v kapitole **3. Technická infrastruktura**.

Výběr licence a její uplatnění

Tato podkapitola má sloužit jako velmi rychlý orientační návod pro výběr licence. Pokud chcete porozumět podrobným právním důsledkům různých licencí a tomu, jak může výběr licence ovlivnit možnost mísení vašeho softwaru s jiným svobodným softwarem, přečtěte si kapitole **9. Licence, autorská práva a patenty.**

Existuje velké množství licencí pro svobodný software, z nichž si můžete vybírat. Většinou z nich se zde nemusíme zabývat, protože byly napsány tak, aby vyhověly právním potřebám konkrétních společností nebo osob, a pro váš projekt by nebyly vhodné. Omezíme se pouze na ty nejběžněji používané licence, protože ve většině případů pro vás bude nejlepší vybrat si jednu z nich.

Licence typu „vše je povoleno“

Pokud vám nevadí, že by kód vašeho projektu mohl být případně využit v proprietárních programech, pak použijte licenci ve stylu *MIT/X*. Je to ta vůbec nejjednodušší z řady minimalistických licencí a nedělá prakticky nic jiného, než že jmenuje držitele autorských práv, aniž by nějak omezovala možnost kopírování; výslovně uvádí, že na kód není poskytována žádná záruka. Podrobnosti najdete v **MIT / X Window System License**.

GPL

Pokud nechcete, aby byl váš kód použit v proprietárních programech, použijte GNU General Public License (**http://www.gnu.org/licenses/gpl.html**). GPL je v současnosti pravděpodobně nejznámější licencí pro svobodný software na světě. To už samo o sobě znamená velkou výhodu, protože ji už mnoho vašich potenciálních uživatelů a přispěvatelů bude znát a nebudou potřebovat věnovat další čas tomu, aby si vaši licenci přečetli a porozuměli jí. Další informace najdete v **GNU General Public License** v kapitole **9. Licence, autorská práva a patenty.**

Pokud uživatelé s vašimi programy pracují především prostřednictvím sítě, a váš software je tedy obvykle součástí hostované služby, pak místo GPL zvažte použití *GNU Affero GPL*. Podrobnosti najdete v části **GNU Affero GPL: Verze GNU GLP pro kód na straně serveru** v kapitole **9. Licence, autorská práva a patenty.**

Jak licenci aplikovat

Až si licenci vyberete, nezapomeňte ji uvést na hlavní stránce projektu. Nemusíte tam vkládat celý text licence – stačí jen její jméno a odkaz na plné znění nacházející se na jiné stránce.

Tím veřejnosti řeknete, s jakou licencí máte v úmyslu software zveřejnit, ale z právního hlediska tento krok nestačí. Je potřeba, aby tuto licenci obsahoval i software samotný. Obvykle se to dělá tak, že licenci v plném znění přiložíte do souboru nazvaného `COPYING` (nebo `LICENSE`) a na začátek každého zdrojového souboru připojíte krátké oznámení, v němž uvedete datum copyrightu, jeho držitele a typ licence, včetně odkazu na její plné znění.

To se dá dělat celou řadou různých způsobů, takže si uvedeme jen jeden příklad. GNU GPL doporučuje na začátek každého souboru uvést následující oznámení:

```
Copyright (C) <year>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>
```

Není zde výslovně řečeno, že se kopie licence obdržené spolu s programem nachází v souboru `COPYING`, ale obvykle se vkládá právě tam. (Citované upozornění samozřejmě můžete upravit tak, aby to uvádělo přímo.) Tato šablona uvádí i odkaz na webovou stránku, která obsahuje text licence. Další běžnou metodou je uvedení poštovní adresy, na které si můžete kopii licence objednat. Podle svého uvážení uveďte odkaz na jakékoliv místo, kde si myslíte, že je k nalezení nejtrvanlivější kopie licence – což může být jednoduše například na vašich webových stránkách. Obecně se dá říct, že poznámka přidávaná na začátek každého souboru nemusí nutně vypadat úplně stejně jako ta, kterou jsme si zde ukázali; důležité je, aby měla na začátku oznámení o držiteli copyrightu a datum, uvedla jméno licence a jednoznačný odkaz na to, kde nalézt její plné znění.

Udávání tónu

Zatím jsme se zabývali jednorázovými úkony, které se při zahájení projektu provádějí: výběr licence, uspořádání webových stránek atd. Ty nejdůležitější aspekty zahájení nového projektu jsou ale dynamické. Zvolit adresu mailing listu je snadné, ale zajistit, aby se v něm konverzace držely tématu a byly produktivní, to už je něco úplně jiného. Pokud svůj projekt otevíráte po několika letech uzavřeného, interního vývoje, bude potřeba radikálně změnit podobu jeho dosavadních vývojových procesů, na což musíte stávající členy týmu připravit.

První kroky jsou nejtěžší, protože ještě nejsou ustanoveny precedenty a nikdo zatím přesně neví, jak bude vše probíhat. Stabilita projektu nemá svůj původ v nějaké sadě předpisů, ale ve sdílené kolektivní moudrosti, která se sama časem vyvíjí a již lze jen těžko nějak zachytit. Často existují i psaná pravidla, ale ta obvykle bývají spíše jakýmsi zjednodušením těch nepsaných a neustále se proměňujících dohod, které projekt skutečně řídí. Psané zásady kulturu projektu ani tak nedefinují, jako spíš popisují – a i to jen přibližně.

Pro to, proč věci fungují právě takto, existuje několik důvodů. Růst a vysoká fluktuace zúčastněných nejsou pro vytváření společných norem zas tak velkou překážkou, jak bychom se mohli domnívat. Pokud se změny nedějí až příliš rychle, mají nově příchozí vždy dost času se seznámit s tím, jak věci fungují; až se vše naučí, budou sami dodržování stejných pravidel prosazovat. Funguje to trochu podobně jako způsob, kterým se šíří dětské říkanky. Dnešní děti používají stále zhruba stejná říkadla jako před stovkami let, ačkoliv tehdejší děti už dávno nežijí. Mladší děti je znají z podání starších dětí; až vyrostou, naučí je zase ty mladší. Tohle samozřejmě děti nedělají cíleně, aby zajistily jejich předání dalším generacím, ale ve skutečnosti dělají právě to: celý pravidelně se opakující mechanismus znamená, že říkadla přežívají dál. Časová rozpětí projektů svobodného softwaru se možná nebudou pohybovat v řádu staletí (na dělání takových závěrů je ještě trochu brzo), ale dynamika přenosu je do značné míry stejná. Výměna účastníků je zde ovšem podstatně rychlejší, což je potřeba kompenzovat tím, že celý přenos bude probíhat aktivněji a cílevědoměji.

Tomu pomáhá fakt, že když většina lidí někam přijde, bude předpokládat, že tam existují nějaké společenské normy, a začne se po nich pídit. Lidé jsou zkrátka takoví. V jakékoliv skupině, kterou sdružuje společný cíl, noví účastníci instinktivně hledají způsoby chování, jež by sdělovaly, že sem patří. Cílem vytvoření precedentů co nejdříve je zajistit, aby se tímto standardním chováním stalo něco, co projektu prospívá, protože jak se normy jednou podaří vytvořit, budou už se do značné míry udržovat samy.

V dalších oddílech ukážu pár příkladů konkrétních věcí, které můžete pro vytvoření dobrých precedentů udělat. Jejich seznam není vyčerpávající, má spíše ukázat, jak může projektu ohromně pomoct, když se v něm už na začátku vytvoří atmosféra spolupráce. V reálném světě může sice každý vývojář pracovat sám a zcela odděleně, ale můžete v něm zkusit vyvolat pocit, jako kdyby pracoval společně s ostatními v jedné veliké místnosti. Čím silněji se takto bude cítit, tím víc času bude chtít projektu věnovat. Tyto konkrétní příklady jsem si vybral, protože se objevily v projektu Subversion (**http://subversion.tigris.org/**), kterého jsem se účastnil a který jsem sledoval od úplného začátku. Rozhodně ale nejsou jedinečné pro Subversion – s podobnými situacemi se setkáte u většiny open source projektů a měli byste je vnímat jako příležitosti pro vykročení tou správnou nohou.

Vyhněte se soukromým diskuzím

I po zveřejnění projektu se spolu s jeho dalšími zakladateli často přistihnete při tom, že byste raději obtížné otázky vyřešili při soukromých rozhovorech v úzkém kruhu. Platí to zvlášť na začátku projektu, kdy se musí učinit řada důležitých rozhodnutí a kdy obvykle máte jen hrstku dobrovolníků, kteří by k tomu byli způsobilí. V takových situacích budete mít před očima všechny nevýhody, které diskuze ve veřejných mailing listech mají: zpoždění spojené s konverzacemi prostřednictvím e-mailu, nutnost ponechat dostatek času na zformování konsenzu, občas velmi únavné jednání s naivními dobrovolníky, kteří si myslí, že všem problémům rozumí, ale ve skutečnosti nerozumí (takoví se vyskytují v každém projektu; někdy se z nich časem stanou vaši nejlepší přispěvatelé, někdy zůstanou naivními napořád), nebo těmi, kdo zkrátka nedokážou pochopit, proč chcete řešit jen problém X, když se zjevně jedná o podmnožinu většího problému Y atd. Pokušení rozhodnout za zavřenými dveřmi a výsledek pak předložit jako hotovou věc nebo přinejmenším jako silná doporučení jednotné a vlivné skupiny lidí bude ve skutečnosti značné.

Ale nedělejte to.

I když mohou být veřejné diskuze někdy velmi pomalé a neohrabané, z dlouhodobého hlediska jsou téměř vždy tím nejlepším řešením. Pokud budete vést důležité rozhovory pouze v soukromí, mnoho přispěvatelů tím odradíte. Žádný normální dobrovolník nezůstane dlouho někde, kde všechna velká rozhodnutí dělá jakýsi tajný výbor. Kromě toho mají veřejné diskuze i pozitivní vedlejší účinky, které jsou z dlouhodobého hlediska významnější než ten konkrétní problém, který se zrovna řešil:

- Diskuze pomáhá školit a vychovávat nové vývojáře. Nikdy nevíte, kolik očí konverzaci sleduje – většina lidí se jí vůbec nemusí účastnit, jen ji tiše sledovat, a získávat tak informace o softwaru.
- Diskuze školí i vás samotné – učí vás vysvětlovat technické problémy lidem, kteří se softwarem nejsou tak obeznámení jako vy. Tato dovednost vyžaduje praxi a tu nezískáte, když budete mluvit s lidmi, kteří už ví všechno, co víte i vy.
- Diskuze a její závěry zůstanou už napořád uchovány ve veřejných archivech projektu, díky čemuž se budoucí debaty budou moct vyhnout prošlapávání stejných cest. Viz **Nápadné využívání archivů** v kapitole **6. Komunikace.**

Konečně existuje i možnost, že někdo z účastníků přispěje do konverzace myšlenkou, která vás vůbec nenapadla. Je obtížné říci, jaká je pravděpodobnost, že toto nastane – závisí to na složitosti kódu a na stupni požadované specializace. Ale ze svých zkušeností bych si dovolil tvrdit, že se to stává častěji, než by si většina lidí myslela. V projektu Subversion jsme (my zakladatelé) věřili, že před námi stojí několik zásadních a složitých problémů, o kterých jsme usilovně přemýšleli několik měsíců; zcela upřímně jsme pochybovali, že by do této diskuze byl někdo z čerstvě založeného mailing listu schopen nějak smysluplně přispět. Takže jsme si vybrali tu snadnější cestu a vyměňovali jsme si své technické nápady přes soukromé maily. Brzy si ale jeden pozorovatel projektu[10] domyslel, co se děje, a požádal nás, abychom diskuzi přenesli do veřejného mailing listu. Pomysleli jsme si své, ale udělali jsme to. To množství promyšlených, věcných komentářů a návrhů, které se záhy začaly objevovat, nás zcela ohromilo. Hned několikrát přišel někdo s návrhem, který nás vůbec nenapadl. Ukázalo se, že v mailing listu bylo celou dobu několik velmi chytrých lidí, kteří jen čekali na tu správnou návnadu. Je tedy pravda, že následná debata zabrala delší čas, než kdybychom zůstali u soukromé konverzace, ale byla tak produktivní, že za ten čas navíc rozhodně stála.

Nechci tady vyslovovat obecná moudra jako „skupina je vždy chytřejší než jednotlivec“ (každý z nás už se v životě setkal se skupinami, u nichž by se o tom dalo vážně pochybovat), ale určitě platí, že existují činnosti, které skupiny dělají mnohem lépe. Jednou z nich je vzájemná kontrola nových příspěvků; rychlé generování velkého počtu nových nápadů je další. Kvalita těchto nápadů samozřejmě závisí na těch, kdo s nimi přišli; na druhou stranu ale nikdy nezjistíte, jak chytré vaše publikum vlastně je, dokud mu nepředložíte nějaký zajímavý problém.

---

[10] K části věnované oceňování zásluh jsme se zatím nedostali, ale stejně bych měl dodržet to, co budu později kázat: tímto pozorovatelem byl Brian Behlendorf a byl to právě on, kdo poukázal na to, jak je důležité vést všechny debaty na veřejnosti, pokud neexistuje nějaký dobrý důvod držet je v soukromí.

Existují samozřejmě i debaty, které musí být vedeny soukromě, a v této knize si několik takových případů popíšeme. Základním principem by ale vždy mělo být: *Pokud není důvod k soukromé debatě, měla by být veřejná.*

Pro to je ale potřeba něco udělat. Nestačí jen zajistit, aby se všechny vaše příspěvky objevily na veřejném mailing listu. Musíte do něj dostat i ostatní, kteří dál vedou soukromé rozhovory, aniž by k tomu měli důvod. Pokud se někdo pokusí zahájit soukromou diskuzi na téma, které to podle vašeho názoru nevyžaduje, považujte za svou povinnost na to okamžitě upozornit. K původnímu tématu se vůbec nevyjadřujte, dokud se vám buď nepovede konverzaci úspěšně přesměrovat na veřejnost, nebo dokud vás ostatní nepřesvědčí, že tato diskuze opravdu musí zůstat soukromá. Pokud v tom budete důslední, ostatní se toho rychle chytí a začnou používat veřejné fórum přednostně.

Nezdvořilé chování potlačte hned v zárodku

Od prvních chvil veřejné existence projektu byste měli na jeho fórech dodržovat zásadu nulové tolerance vůči hrubému a urážlivému chování. Nulová tolerance nemusí nutně znamenat použití technických prostředků. Pokud se někdo naváží do jiných účastníků, nemusíte jej hned odstraňovat z mailing listu; zrovna tak nemusíte odebírat právo přispívat do projektu těm, kdo píší hanlivé poznámky. (Teoreticky můžete být časem k něčemu takovému přinuceni, ale vždy až poté, co všechny ostatní způsoby selhaly, což se už z definice na začátku projektu stát nemůže.) Nulovou tolerancí se jednoduše rozumí to, že špatné chování nikdy neprojde bez povšimnutí. Pokud například někdo napíše technický komentář smíchaný s útokem *ad hominem* na jiného vývojáře projektu, pak musíte nejprve zareagovat na tento útok jako na zcela samostatný problém a až pak přejít k technickému obsahu.

Bohužel se velmi snadno může stát (a není to nijak neobvyklé), že se konstruktivní diskuze zvrhnou ve zbytečné válčení. Do e-mailu někdy lidé napíší věci, které by někomu do očí nikdy neřekli. Tento efekt často zesiluje téma, které se právě probírá. Při řešení technických problémů se často objevuje pocit, že na většinu otázek existuje jediná správná odpověď a že nesouhlas s touto odpovědí lze vysvětlit jen jako výraz nevědomosti nebo hlouposti. Od toho, že někdo prohlásí předložené technické řešení za pitomé, už je jen krůček k tomu, aby byl za pitomce prohlášen jeho autor. Ve skutečnosti je někdy velmi těžké určit, kde končí technická debata a začíná osobní útok, což je také jeden z důvodů, proč není rozumné reagovat drastickými prostředky nebo tresty. Když něco podobného zpozorujete, je vhodné místo toho zareagovat příspěvkem, který zdůrazní důležitost zachování přátelského tónu diskuze, aniž by kohokoliv obviňoval z pokusů ji rozvrátit. Tyhle příspěvky psané ve stylu „hodný policajt“ naneštěstí často znějí, jako když učitelka v mateřské školce dětem vysvětluje, jak se chovat slušně:

> *Za prvé prosím omezte komentáře, které lze vztáhnout k nějaké osobě. Neříkejte tedy například, že návrh bezpečnostní vrstvy, který předložil J., je „naivní a prokazuje základní neznalost toho, jak počítačová bezpečnost funguje“. Možná to pravda je, možná není, ale rozhodně to není způsob, jak vést diskuzi. J. svůj návrh napsal v dobré víře. Pokud má nějaké nedostatky, upozorněte na ně a společně je buď opravíme, nebo vymyslíme něco jiného. M. určitě neměl v úmyslu*

J. urazit, ale jeho formulace byla nešťastná a my se tady pokoušíme zůstat konstruktivní. Teď tedy k tomu návrhu: myslím, že M. má pravdu v tom, že…

I když takové odpovědi mohou znít trochu prkenně, jsou celkem účinné. Pokud budete na špatné chování soustavně poukazovat, ale přitom nebudete po nikom vyžadovat omluvu nebo přiznání vlastní chyby, dáváte diskutujícím čas trochu vychladnout a pro příště ukázat svou lepší stránku v podobě uhlazenějšího chování – což se také obvykle stane. Jedno z tajemství úspěchu této strategie spočívá v tom, že z této vedlejší diskuze nikdy neuděláte hlavní téma. Vždy by to měla být jen krátká poznámka bokem, která předchází hlavní části vaší odpovědi. Upozorněte jaksi mimochodem na to, že „tohle se tady nedělá“, ale pak se věnujte skutečnému obsahu, abyste lidem dali něco, co se týká tématu a na co by mohli reagovat. Pokud někdo začne protestovat, že si vaše pokárání nezasloužil, rozhodně se odmítněte nechat zatáhnout do dalšího sporu. Buď na to vůbec nereagujte (pokud usoudíte, že si prostě potřeboval ulevit a žádná odpověď není třeba), nebo napište, že se za svou přehnanou reakci omlouváte a že se v e-mailech některé jemnější významové odstíny rozlišují jen obtížně. Pak se vraťte k hlavnímu tématu. V žádném případě netrvejte na tom, aby dotyčný uznal, ať už veřejně nebo soukromě, že se choval nevhodně. Pokud se sám od sebe omluví, bude to jen dobře, ale pokud byste to po něm vyžadovali, akorát vyvoláte odpor.

Vaším hlavním cílem je to, aby bylo dodržování pravidel slušného chování vnímáno jako jedna z vnitřních zásad skupiny. Projektu to pomůže, protože podobné hašteření může vést až k odchodu některých vývojářů (a to dokonce i z projektů, které se jim líbí a které chtějí podporovat). To je opět něco, co se nemusíte vůbec dozvědět – může to být někdo, kdo do mailing listu nikdy nepřispívá, ale pečlivě jej sleduje. Pokud dojde k závěru, že účast na projektu vyžaduje hroší kůži, může se rozhodnout, že mu to za to nestojí. Udržování přátelského charakteru fóra je z dlouhodobého hlediska důležité pro přežití projektu a je to mnohem snazší, dokud v něm není mnoho lidí. Jakmile se to stane součástí kultury projektu, nebudete už tím jediným, kdo to bude prosazovat. Pomůžou vám s tím všichni ostatní.

Kontrolujte kód veřejně

Jeden z nejlepších způsobů, jak zajistit, že vaše komunita vývojářů bude výkonná, je zavést pravidlo, že si svůj kód všichni navzájem prohlížejí. Aby se to dalo provádět efektivním způsobem, je potřeba zajistit příslušnou technickou infrastrukturu, především zapnout rozesílání informací o změnách kódu e-mailem (podrobnosti najdete v části **E-maily oznamující commit**). Pokaždé když někdo do zdrojového kódu promítne změnu (commit), rozešle se e-mail, v němž se objeví příslušná zpráva z protokolu a rozdíly, v nichž tato změna spočívá (viz **diff**, v části **Slovníček pojmů správy verzí**). *Revizí kódu* se rozumí praxe, kdy se každý z těchto e-mailů při přijetí pečlivě kontroluje, přičemž se v něm hledají chyby a zkoumá se, zda a jak by se dal zlepšit.[11]

---

[11] Tak se to alespoň dělá u většiny open source projektů. U centrálněji řízených projektů se „revize kódu“ provádí i tak, že se několik lidí společně sejde, projde vytištěný zdrojový kód a pokusí se v něm najít konkrétní problémy a identifikovat časté nedostatky.

Revize kódu slouží několika účelům současně. Je to dobrý příklad toho, jak si v open source světě všichni kontrolují svou práci navzájem, což pomáhá udržet kvalitu softwaru. Každá chyba, která se ve vydané verzi vašeho programu objeví, se tam dostala tak, že byla při zapsání kódu přehlédnuta. Z toho vyplývá, že čím víc očí nově zapsané změny prohlédne, tím méně chyb se dostane na veřejnost. Revize kódu ale zároveň dělá ještě něco jiného – pro účastníky projektu znamená potvrzení, že někomu na jejich práci a na tom, jaký bude mít výsledek, záleží, protože jinak by jen těžko věnovali svůj čas tomu, aby ji kontrolovali. Když lidé vědí, že si ostatní najdou čas na to jejich práci zhodnotit, odvádějí ji, jak nejlépe dokážou.

Revize by měly být veřejné. Dokonce i v případech, kdy jsme seděli s vývojáři v jedné místnosti a někdo z nás zapsal změnu, jsme si dávali pozor na to, abychom revizi neprováděli jen ústně, ale poslali ji do vývojářského mailing listu. Z provádění revizí před očima všech plynou různé výhody. Ti, kteří revizní komentáře čtou, v nich můžou najít nedostatky; v případech, že žádné nenajdou, jim tato činnost bude alespoň připomínat, že revize kódu je něco běžného, co se musí dělat pravidelně, stejně jako třeba umývání nádobí nebo sekání trávníku.

V projektu Subversion jsme ze začátku kód nijak pravidelně nekontrolovali. Neexistovala žádná záruka, že každou změnu kódu někdo zkontroluje; pokud se někdo o příslušnou oblast programu zvlášť zajímal, mohl si občas změny prohlédnout. Tak se stávalo, že se do zdrojového kódu dostaly i celkem zbytečné chyby, které opravdu měly být zachyceny dříve. Vývojář jménem Greg Stein, který už z dřívějška věděl, jak cenné revize mohou být, se rozhodl, že půjde všem příkladem a bude recenzovat každičký řádek každého nového commitu, který se v úložišti kódu objeví. Na každou změnu, kterou někdo zapsal, brzy Greg zareagoval komentářem ve vývojářském mailing listu, v němž ji rozpitval, analyzoval možné problémy a někdy i vyjádřil své uznání nad zvlášť rafinovaným programátorským kouskem. Ihned zachytil chyby a nestandardní postupy, které by jinak prošly bez povšimnutí. Ani jednou si nepostěžoval, že je tím jediným, kdo každou změnu kontroluje, ačkoliv mu to muselo zabrat spoustu času; při každé vhodné příležitosti ale zdůrazňoval, jak je revize kódu důležitá. Poměrně brzy ji začali pravidelně provádět i ostatní, mne nevyjímaje. Jakou jsme měli motivaci? Nebylo to proto, že bychom se před Gregem začali stydět. Dokázal nám ale, že čas strávený revizí kódu opravdu není ztracený a že touto činností můžete projektu prospět zrovna tak jako psaním nového kódu. Když jsme si to všichni uvědomili, stalo se revidování kódu standardním postupem – až do té míry, že pokud nějaká změna prošla bez reakce, začal si její autor dělat starosti a dokonce se pak v mailing listu vyptával, jestli se na ni opravdu zatím nikdo nepodíval. Greg později dostal jinou práci, která znamenala, že projektu Subversion už nemohl věnovat tolik času a musel s pravidelnou kontrolou kódu přestat. Tou dobou už ale byl tento zvyk natolik zakořeněný, že se nám zdálo, že tu byl už od nepaměti.

Začněte revidovat kód hned u prvních příspěvků. Mezi problémy, které se při revizi rozdílů nejsnadněji zachytí, patří bezpečnostní nedostatky, úniky paměti, nedostatečné komentování nebo dokumentace API, cykly provedené s jedním opakováním navíc/méně, nedodržení standardní komunikace mezi volajícím a volaným (caller/calee) a další problémy, které se dají zpozorovat i v minimálním kontextu. Může ale zachytit i jiné nedostatky, které jsou mnohem rozsáhlejší, jako například možnost sloučit podobné části kódu na jediné místo, protože pokud děláte revize pravidelně, pak si při prohlížení jedné části snadno vybavíte, co jste kontrolovali předtím.

Nedělejte si starosti s tím, že možná nenajdete nic, co by se dalo okomentovat, nebo že toho o některých částech kódu víte jen málo. Skoro ke každé změně je co říct. Když v ní nenajdete žádný problém, možná najdete alespoň něco, co si zaslouží pochvalu. Důležité je dát každému přispěvateli najevo, že si jeho příspěvku někdo všiml a že mu porozuměl. Revize kódu samozřejmě nezbavují programátory zodpovědnosti za to, aby si své změny před zapsáním prohlédli a otestovali sami. Nikdo by neměl spoléhat na to, že revize jeho kódu odhalí věci, kterých si měl všimnout sám.

Když otvíráte dosud uzavřený projekt, uvědomte si rozsah změn

Pokud otvíráte existující projekt, který má své aktivní vývojáře zvyklé na práci v uzavřené skupině, ujistěte se, že všichni rozumí tomu, že to přinese velké změny – a také že se na věc dokážete podívat jejich očima.

Zkuste si představit, jak se celá situace jeví jim: dříve psala kód a o projektu rozhodovala skupina programátorů, kteří daný software znali víceméně stejně dobře, kteří byli pod stejným tlakem stejného vedení a kteří vzájemně znali své silné a slabé stránky. Teď po nich chcete, aby svůj kód vydali napospas zkoumavému pohledu úplně cizích lidí, kteří je budou soudit výhradně na základě kódu, aniž by tušili, že některá rozhodnutí mohla být výsledkem tlaku shora. Tito noví lidé budou pokládat řadu otázek, díky nimž si původní vývojáři uvědomí, že dokumentace, se kterou se tolik nadřeli, na svůj úkol nestačí (to je něco, čemu se nedá vyhnout). A aby toho nebylo málo, všichni tito nováčci jsou zcela neznámí jedinci bez tváře. Pokud si některý z vašich vývojářů už předtím nebyl úplně jist svými schopnostmi, představte si, jak se jej dotkne, až někdo přicházející zvenku poukáže na chyby v jeho kódu, a co hůř, ještě před jeho kolegy. Pokud váš tým nesestává výhradně z dokonalých programátorů, tak se něčemu takovému nevyhnete; pravděpodobně se to ze začátku stane každému z nich. Ne proto, že by jejich práce byla špatná, ale proto, že každý dostatečně složitý program chyby zkrátka obsahuje a kontrola kódu někým jiným než původním autorem některé z nich může objevit (viz **Kontrolujte kód veřejně** výše v této kapitole). Zároveň bude platit, že tyto nováčky samotné nebude kritizovat prakticky nikdo, protože se ještě neseznámili s projektem natolik, aby do něj mohli sami přispět. Vaši vývojáři tak mohou snadno získat pocit, že veškerá kritika směřuje zvenku na ně a nikdy ne opačným směrem; hrozí, že si začnou připadat jako v obležení a budou se i bránit.

Nejlepší způsob, jak tomu předejít, je každého upozornit, co nastane, vysvětlit to, sdělit jim, že je naprosto normální, když to na začátku bude poněkud nepříjemné, a ujistit je, že se to zlepší. Některá z těchto varování by měla proběhnout soukromě ještě předtím, než je projekt otevřen. Může ale být užitečné všem účastníkům ve veřejných mailing listech čas od času připomenout, že vývoj projektu teď probíhá novým způsobem a že nějakou dobu potrvá, než se vše přizpůsobí. To nejlepší, co můžete udělat, je jít sám příkladem. Pokud uvidíte, že vaši vývojáři neodpovídají dostatečně na otázky nováčků, pak moc nepomůže, když jim řeknete, aby odpovídali víc. Možná zatím nemají vyvinutý smysl pro to, co si odpověď zaslouží a co ne; možná si ještě neumí pořádně uspořádat priority mezi programováním samotným a novou zátěží v podobě komunikace s okolím. K účasti je přimějete tak, že se zúčastníte sami. Sledujte veřejný mailing list a na otázky, které se v něm objeví, odpovídejte. Pokud nějaký dotaz přesahuje vaše odborné znalosti, pak jej viditelně přihrajte jinému vývojáři, který dané problematice

rozumí – a ujistěte se, že na něj pak skutečně odpoví nebo alespoň zareaguje. Dlouholetí vývojáři budou v pokušení důležité otázky řešit v soukromých diskuzích, protože jsou na to zvyklí. Nezapomeňte tedy ani sledovat dění v interních mailing listech, na kterých k tomu může docházet, abyste mohli případně požádat o přenesení celé diskuze na nějaké veřejné fórum.

S otevřením dříve uzavřených projektů jsou spojeny i další, dlouhodobé problémy. V kapitole **5. Peníze** se podíváme na to, jak úspěšně řídit spolupráci placených a neplacených vývojářů, a v kapitole **9. Licence, autorská práva a patenty** si řekneme, proč je třeba postupovat opatrně při zpřístupňování soukromých zdrojových kódů, které mohou obsahovat software napsaný nebo „vlastněný“ dalšími stranami.

Oznamování

Až bude projekt ve stavu, který se dá předvést na veřejnosti (nemusí být dokonalý, stačí, když bude přijatelný), je čas jeho existenci oznámit. To je až překvapivě jednoduchá záležitost: jděte na **http://freshmeat.net/**, klikněte na tlačítko Submit v horní navigační liště a vyplňte oznamovací formulář. Freshmeat všichni sledují jako to místo, kde se nové projekty ohlašují. Stačí, aby si oznámení vašeho projektu všimlo pár lidí; pak už se tato informace začne šířit sama.

Pokud víte o mailing listech nebo diskuzních skupinách, kam by oznámení vašeho projektu tematicky zapadalo a kde by mohlo vyvolat zájem, napište tam; dejte ale pozor na to, abyste do každého fóra poslali jen jedno oznámení a abyste případnou následující diskuzi přesměrovali do fóra vyhrazeného vašemu projektu (nastavením hlavičky Reply-to). Zpráva by měla být krátká a měla by jít rovnou k věci:

```
Komu: diskuzni@list.priklad.org
Předmět: [ANN] Projekt fulltextového indexovacího nástroje Scanley
Reply-to: dev@scanley.org

Toto je jednorázové oznámení vzniku projektu Scanley, což je
open source fulltextový indexovací a vyhledávací nástroj s bohatým
API, jenž by mohli využívat programátoři při poskytování vyhledávacích
služeb ve větším množství textových souborů.  Scanley má v současnosti
podobu funkčního kódu, je aktivně vyvíjen a hledá nové vývojáře
i testery.

Domovská stránka: http://www.scanley.org/
```

```
Vlastnosti:
   - Prohledávání souborů ve formátu prostého textu, HTML a XML
   - Vyhledávání slov a frází
   - (plánuje se) Vyhledávání částečných shod (Fuzzy matching)
   - (plánuje se) Inkrementální aktualizace indexů
   - (plánuje se) Indexování vzdálených webových serverů

Požadavky:
   - Python 2.2 nebo vyšší
   - Dostatečně velký diskový prostor pro uložení indexů (přibližně
     dvojnásobek velikosti zdrojových dat)

Více informací naleznete na scanley.org.

Děkuji,
-J. Novák
```

(Rady, jak oznamovat nové verze softwaru a jiné události projektu, naleznete v sekci **Publicita** v kapitole **6. Komunikace.**)

Ve světě svobodného softwaru už dlouho probíhá debata, zda je nutné začínat fungujícím programem, nebo zda může být pro projekt výhodné, když je otevřený už během etapy návrhu a diskuzí. Dříve jsem si myslel, že začít s fungujícím programem je ten vůbec nejdůležitější faktor, který odděluje úspěšné projekty od pouhých hraček; měl jsem za to, že vážné zájemce z řad vývojářů přitáhne jen software, který už něco konkrétního dělá.

Ukázalo se ale, že to není tak úplně pravda. V projektu Subversion jsme začali s dokumentem popisujícím návrh, s několika vývojáři, kteří měli o věc zájem a kteří spolu dobře komunikovali, s velkolepou reklamou a bez kousku běžícího kódu. K mému velkému překvapení projekt získal aktivní účastníky hned od samého začátku a v době, kdy už jsme měli alespoň něco, co by se dalo spustit, už s ním úzce spolupracovalo poměrně velké množství dobrovolníků. A Subversion není jediným příkladem – například projekt Mozilla byl také ohlášen, aniž by měl jakýkoliv spustitelný kód, a dnes je z něj úspěšný a populární webový prohlížeč.

Ve světle těchto důkazů jsem musel svůj předpoklad, že pro spuštění projektu je nutné mít běžící program, poněkud přehodnotit. Stále platí, že funkční program je tím nejlepším základem úspěchu a že je v zásadě dobrý nápad počkat s ohlášením projektu až do doby, než ho budete mít. Nicméně existují situace, kdy má smysl zveřejnit oznámení už dřív. Pořád si myslím, že přinejmenším dobře propracovaný dokument s návrhem nebo alespoň nějaká kostra celého programu jsou nezbytné. Samozřejmě je možné jej na základě zpětné vazby od veřejnosti poněkud upravit, ale vždy musíte mít něco konkrétního, něco hmatatelnějšího než jen pár dobrých nápadů, aby se toho někdo mohl chytit.

Ať už ale vydáte oznámení kdykoliv, nečekejte, že se k projektu ihned připojí hromada dobrovolníků. Výsledkem oznámení většinou je, že dostanete několik nezávazných dotazů, k mailing listu se připojí pár lidí a kromě toho všechno pokračuje v zásadě tak jako předtím. Časem si ale všimnete, že se objevuje čím dál více nových přispěvatelů i uživatelů. Oznámení je jen jakési zasazení semínka. Než se zpráva rozšíří, může to trvat dlouho. Pokud projekt bude důsledně odměňovat ty, co se do něj zapojí, šířit se bude, protože když někdo najde něco dobrého, chce se o to obvykle také podělit. Pokud vše půjde dobře, promění časem dynamika sítí s exponenciálně narůstající komunikací váš projekt v komplexní komunitu, ve které už nebudete znát každého člena jménem a nebudete schopni sledovat každou konverzaci. Následující kapitoly se zabývají tím, jak v takovém prostředí pracovat.

3. Technická infrastruktura

3. Technická infrastruktura — 73

Co je třeba pro projekt zajistit — 74

Mailing listy — 75
Ochrana před nežádoucími zprávami — **77**
Filtrování příspěvků — **77**
Skrývání adres v archivech — **79**
Identifikace a správa hlaviček — **80**
Velká debata o Reply-to — **82**
Dvě ideální řešení — **84**
Archivace — **85**
Software — **86**

Správa verzí — 87
Slovníček pojmů správy verzí — **87**
Výběr systému pro správu verzí — **91**
Používání systému pro správu verzí — **92**
Sledujte verze u všeho — **92**
Možnost prohlížení — **93**
E-maily oznamující commit — **93**
Používejte větve, abyste nezpomalovali vývoj — **95**
Jedinečnost informace — **96**
Autorizace — **97**

Systém pro sledování chyb — 99
Interakce s mailing listy — **102**
Předběžné filtrování v bug trackeru — **103**

IRC a jiné systémy pro diskuzi v reálném čase — 104
Roboti — **106**
Archivace IRC — **107**

RSS — 107
Wiki — 108

Webové stránky — 110
Kompletní hosting — **110**
Jak si vybrat kompletní hosting — **111**
Anonymita a zapojení se do projektu — **112**

3. Technická infrastruktura

Projekty svobodného softwaru spoléhají na technologie, které podporují selektivní zachycování a sdružování informací. Čím lépe umíte tyto technologie používat a čím spíše dokážete ostatní přesvědčit, aby je používali také, tím bude projekt úspěšnější. Jak projekt roste, stává se celá věc ještě důležitější. Dobrá správa informací je tím, co open source projekty chrání před podlehnutím Brooksovu zákonu,[12] který říká, že připojením dalších lidí ke zpožděnému softwarovému projektu dosáhnete jen dalšího zpoždění. Fred Brooks zpozoroval, že složitost projektu roste s druhou mocninou počtu účastníků. Pokud je do něj zapojeno jen pár lidí, mohou mezi sebou komunikovat celkem snadno; pokud má ale nějaký projekt stovky členů, pak není možné, aby měl každý z nich přehled, co dělají ti ostatní. Pokud má dobrá správa projektu svobodného softwaru zajistit, aby měl každý pocit, jako by pracoval se všemi ve stejné místnosti, vyvstane samozřejmě otázka, co se stane, když se v nějaké přeplněné místnosti pokusí všichni najednou něco říct.

Tento problém není ničím novým. V reálném prostředí spočívá řešení v ustanovení určité, řekněme, *parlamentní procedury*, souboru formálních pravidel pro řízení rozsáhlých diskuzí v reálném čase, která zajistí, že se důležité námitky neztratí v záplavě komentářů typu „já taky“, určí, jak se ustanoví užší výbory, jak rozpoznat, že byla učiněna rozhodnutí atd. Důležitou součástí takových procedur je také stanovit, jak bude skupina zacházet se svým systémem pro správu informací. Některé poznámky jsou určeny „k zaprotokolování“, jiné nikoliv. Takový zápis je vždy nějak upraven – není brán jako doslovný záznam toho, co se událo, ale jako reprezentace toho, na čem se skupina shodla, že se událo. Tyto záznamy nejsou jednolité; pro různé účely na sebe berou různé podoby. Patří sem zápisy z jednotlivých jednání, soubor všech zápisů ze všech jednání, jejich shrnutí, programy a poznámky k nim, zprávy výborů, zprávy od nepřítomných přispěvatelů, seznamy bodů jednání atd.

Protože internet není skutečná místnost, můžeme rovnou vynechat ty části parlamentních procedur, které zajistí, že když někdo mluví, jsou všichni ostatní zticha. Co se ale technik správy informací týče, jsou dobře vedené open source projekty o několik kvalitativních úrovní výše než jakákoliv parlamentní procedura. Protože se v open source projektech vede téměř veškerá komunikace písemně, vyvinuly se propracované systémy pro přesměrovávání a označování dat, pro minimalizaci opakování, které má zabránit falešnému větvení, pro ukládání a získávání dat, pro opravování špatných nebo zastaralých údajů a pro sdružování různě roztroušených informací dohromady, pokud se mezi nimi odhalí nové souvislosti. Aktivní účastníci open source projektů už přijali mnohé z těchto technik za své a často ručně provádějí poměrně složité úkony, aby zajistili, že se každá informace dostane na správné místo. Ale veškeré úsilí závisí hlavně na důmyslné softwarové podpoře. Komunikační média by měla zajišťovat správné pracovní postupy, označování a zaznamenávání informací; měla by také být schopna tyto informace zpřístupnit co nejužitečnějším způsobem. V praxi samozřejmě pořád bude potřeba, aby do celého procesu na různých místech zasahovali lidé; je tedy pochopitelně také důležité, aby

---

[12] Poprvé popsaném v knize *The Mythical Man-Month* (Mýtus zvaný člověkoměsíc), 1975.
Viz **http://en.wikipedia.org/wiki/The_Mythical_Man-Month** a **http://en.wikipedia.org/wiki/Brooks_Law.**

takové zásahy šlo dělat pokud možno jednoduše. Obecně ale platí, že pokud si někdo při zanášení dat do systému dá záležet na tom, aby je správně označil a nasměroval, pak by měl být software nakonfigurován tak, aby tato metadata dokázal co nejvíc využít.

Rady v této kapitole jsou velmi praktické a zakládají se na zkušenostech s konkrétním softwarem a na tom, jak se skutečně používá. Nechci vám tady ale jen ukázat několik osvědčených postupů. Tato kapitola by také měla pomocí řady dílčích příkladů ilustrovat celkový přístup, který vašemu projektu zajistí dobrou správu informací. Tento přístup v sobě zahrnuje kombinaci technických a lidských dovedností. Technické dovednosti jsou nezbytné, protože software pro správu informací je vždy nejprve potřeba nakonfigurovat a pak tato nastavení průběžně upravovat vždy, když se objeví nějaké nové požadavky (viz například diskuze týkající se zvládání růstu projektu v části **Předběžné filtrování v bug trackeru** dále v této kapitole). Schopnost pracovat s lidmi je také důležitá, protože i lidská komunita vyžaduje „údržbu“. Není totiž vždy zřejmé, jakým způsobem lze příslušné nástroje využít co nejefektivněji, a v některých případech používají různé projekty různé, navzájem nekompatibilní standardy (viz například diskuze o nastavování hlaviček `Reply-to` u zpráv zasílaných do mailing listu – viz **Mailing listy**). Každý, kdo je do projektu zapojen, musí být ve správný čas a správným způsobem vyzván, aby pomohl udržet informace projektu v dobře organizovaném stavu. Čím více je přispěvatel do projektu zapojen, tím komplexnější a specializovanější techniky by se měl naučit.

Správa informací není něco, co by se dalo řešit podle nějaké příručky. Na to je v ní příliš mnoho proměnných. Může se stát, že se konečně propracujete ke konfiguraci, která dělá všechno přesně podle vašich představ, a většina komunity se bude zavedenými pravidly řídit, ale při dalším růstu projektu se ukáže, že některé ze zavedených praktik byly neškálovatelné a teď už nefungují. Nebo se růst projektu může stabilizovat, vývojáři a komunita uživatelů se s technickou infrastrukturou naučí zacházet, ale pak někdo vymyslí úplně novou službu pro správu informací a nováčci se záhy začnou ptát, proč ji váš projekt nepoužívá – to se v současnosti například stává mnoha projektům svobodného softwaru, které vznikly před vynálezem systému wiki (viz **http://en.wikipedia.org/wiki/Wiki**). U mnoha otázek záleží jen na vašem uvážení, na nalezení kompromisu mezi vyhověním těm, kteří informace produkují, a těm, kteří je konzumují, popřípadě mezi časem, jenž je potřebný ke konfiguraci softwaru pro správu informací, a výhodami, které to projektu přinese.

Vyvarujte se pokušení vše přeautomatizovat, tedy pokoušet se automaticky řešit věci, které spíše vyžadují lidský zásah. Technická infrastruktura je důležitá, ale projekt svobodného softwaru udržuje v chodu především systematická činnost lidí, kteří se jej účastní. Technika je tu hlavně proto, aby tuto činnost zjednodušila.

Co je třeba pro projekt zajistit

Většina open source projektů nabízí přinejmenším určité minimum, standardní sadu nástrojů pro správu informací:

Webové stránky
Převážně centralizovaný, jednosměrný způsob, jak předat informace o projektu veřejnosti. Webové stránky mohou sloužit také jako administrativní rozhraní pro ostatní nástroje projektu.

Mailing listy
Obvykle nejaktivnější komunikační fórum projektu, které je také archivované.

Správa verzí
Umožňuje vývojářům jednoduše spravovat změny zdrojového kódu, včetně možnosti vrátit se k původnímu stavu a změny přenést jinam. Díky správě verzí mohou mít všichni přehled, co se s kódem děje.

Sledování chyb
Umožňuje vývojářům sledovat, na čem se právě pracuje, rozdělovat si práci mezi sebou a plánovat vydání nových verzí. Pomocí tohoto systému se může kdokoliv dotázat na stav chyb a zaznamenávat informace ke konkrétním chybám (například postupy, jak je vyvolat). Nemusí se používat jen k sledování chyb, ale také ke správě úkolů, vydávání nových verzí, přidávání nových funkcí atd.

Diskuze v reálném čase
Místo pro rychlou, odlehčenou diskuzi a pro výměnu otázek a odpovědí. Nebývá vždy plně archivována.

Každý z těchto nástrojů má své konkrétní použití, ale jejich funkce jsou navzájem provázané; musí tedy být navrženy tak, aby mohly spolupracovat. Níže se podíváme na to, jak toho dosáhnout a hlavně jak účastníky projektu přimět, aby je používali. Webové stránky si necháme až na konec, protože ty nejsou ani tak nástrojem samy o sobě jako spíše tím, co drží všechno ostatní pohromadě.

Při výběru a konfiguraci těchto nástrojů si můžete ušetřit spoustu starostí tím, že využijete některou ze služeb *kompletního hostingu*, tedy serveru, který nabízí v podobě šablon už připravené webové služby a všechny doprovodné nástroje, jež k provozu projektu svobodného softwaru potřebujete. Diskuzi o výhodách a nevýhodách kompletního hostingu najdete dále v této kapitole – viz **Kompletní hosting.**

Mailing listy

Mailing listy jsou tím hlavním místem, kde probíhá komunikace projektu. Pokud uživatel kromě webových stránek zavítá i na nějaké fórum, pak to budou s největší pravděpodobností mailing listy. Ale ještě předtím, než se setká se samotným mailing listem, musí nejprve najít jeho rozhraní, tedy ten mechanismus, pomocí nějž se do mailing listu přihlásí (subscribe). To nás přivádí k pravidlu číslo jedna pro mailing listy:

Nepokoušejte se spravovat mailing listy ručně. Pořiďte si na to software.

Možná se vám do toho nebude chtít. Ze začátku se zavádění softwaru pro správu mailing listu může zdát jako zbytečně přehnané. Ruční správa mailing listu, který má jen hrstku členů a celkem malý objem zpráv, je zrádně jednoduchá. Ustanovíte nějakou adresu pro přihlašování, která bude přesměrovaná na vás, a pokud na ni někdo napíše, jednoduše otevřete textový soubor se seznamem členů mailing listu a jeho e-mailovou adresu do něj přidáte, popřípadě ji z něj smažete. Nic složitého v tom není.

Zádrhel spočívá v tom, že dobrá správa mailing listu, což je to, co budou všichni očekávat, není vůbec jednoduchá záležitost. Nejsou to jen požadavky uživatelů na přihlášení a odhlášení. Je také potřeba mailing list moderovat, aby se do něj nedostávaly nežádoucí e-maily (spam), nabízet obsah mailing listu jak ve formě přehledu příspěvků, tak jako seznam jednotlivých zpráv, poskytovat standardní informace o mailing listu a o projektu prostřednictvím automatických odpovědí a různé další náležitosti. Pokud přihlašovací adresu spravuje pouze člověk, je schopen vykonávat z těchto funkcí jenom některé, a to ani tak spolehlivě, ani tak rychle, jako to dokáže software.

Moderní software pro správu mailing listu obvykle nabízí přinejmenším následující funkce:

Přihlašování jak přes e-mail, tak přes webové rozhraní
Pokud se uživatel přihlásí do mailing listu, měl by obratem obdržet automatickou odpověď s uvítací zprávou, která mu oznámí, kam se přihlásil, jak se se softwarem mailing listu zachází a (což je nejdůležitější) jak se může odhlásit. Tato automatická odpověď může být samozřejmě upravena tak, aby obsahovala také informace specifické pro daný projekt, jako je jeho webová stránka, umístění dokumentu FAQ atd.

Přihlášení v režimu přehledu nebo zobrazení jednotlivých zpráv
V režimu přehledu (digest mode) obdrží účastník každý den jeden e-mail, který zachycuje aktivitu v mailing listu za posledních 24 hodin. Pro ty, kteří mailing list sledují jen zpovzdálí a nepřispívají do něj, bývá přehledový režim často vhodnější, protože jim umožní rychlé prohlédnutí všech zpráv najednou a nebudou je rozptylovat e-maily přicházející v náhodných intervalech.

Možnosti moderování
„Moderování“ spočívá v kontrole příspěvků, která prověří, zda a) nejsou spam, b) jsou k tématu, ještě předtím, než se zpráva rozešle celému mailing listu. Moderování nezbytně vyžaduje lidský zásah, ale software jej může v mnohém usnadnit. O moderování si řekneme víc později.

Rozhraní pro správu
Toto rozhraní umožňuje administrátorům mimo jiné do systému vstoupit a snadno odstranit zastaralé adresy. To je zvlášť důležité v situacích, kdy začne z adresy příjemce na každou příchozí zprávu automaticky chodit do mailing listu odpověď typu „Tato adresa se již nepoužívá“. (Některý software pro mailing listy umí tyto případy rozpoznat a dotyčnou osobu pak odhlásí automaticky.)

Úprava hlaviček
Mnozí lidé mají ve svých programech pro práci s poštou definována důmyslná pravidla pro filtrování a odpovídání. Software pro mailing list může ke každé zprávě přidávat nebo v ní měnit některé standardní hlavičky, které se k těmto účelům dají použít (podrobnosti viz níže).

Archivace
Všechny příspěvky do mailing listů by se měly ukládat a být přístupné na webu. Některé programy pro mailing listy nabízí k tomuto účelu zvláštní rozhraní pro zapojení externích archivačních nástrojů, jako je například MHonArc (**http://www.mhonarc.org/**). Jak je řečeno v části **Nápadné využívání archivů** v kapitole **6. Komunikace**, archivace je nesmírně důležitá.

Tento seznam má ukázat, že správa mailing listu je složitý problém, nad kterým se už hodně přemýšlelo a který byl do značné míry vyřešen. Není nutné, abyste se v této oblasti stali odborníky. Měli byste si ale uvědomit, že je to téma, o němž se budete učit nové věci prakticky pořád, a že si v rámci řízení projektu svobodného softwaru správa mailing listu čas od času vyžádá vaši pozornost. V dalších oddílech se podíváme na několik nejběžnějších problémů s konfigurací mailing listů.

Ochrana před nežádoucími zprávami

Během doby, která uplyne mezi tím, než tuhle větu napíšu a než bude zveřejněna, se pravděpodobně závažnost problému zvaného spam, jenž sužuje celý internet, zdvojnásobí – a jestli ne, tak vám to tak alespoň bude připadat. Byly doby, a není to zase tak dávno, kdy bylo možné provozovat mailing list, aniž byste si museli nevyžádanými zprávami lámat hlavu. Občas se sice ojedinělý kousek někde objevil, ale byl to tak řídký jev, že to nikoho moc neobtěžovalo. Tahle éra už je navždy pryč. Dnes už by se mailing list, který nepoužije žádné preventivní opatření proti spamu, začal nežádoucími e-maily plnit tak rychle, že by byl zcela nepoužitelný. Ochrana proti spamu je nezbytná.

Prevenci proti spamu dělíme na dvě kategorie: zabránění tomu, aby se ve vašich mailing listech objevoval, a zabránění tomu, aby se vaše mailing listy mohly stát zdrojem e-mailových adres pro ty, kdo spam šíří. První kategorie je důležitější, takže se podíváme nejdříve na ni.

Filtrování příspěvků

Na to, aby se zabránilo nežádoucím příspěvkům, se používají tři základní techniky a většina softwaru pro správu mailing listů nabízí všechny tři. Nejlepší je, když jsou užity současně:

1. **Automatické schvalování, které se týká výhradně příspěvků od členů mailing listu.**
   V rámci možností jde o celkem efektivní techniku, která navíc nevyžaduje prakticky žádnou údržbu, protože jde obvykle jen o jediné nastavení v konfiguraci softwaru pro správu mailing listu. Je třeba si ale uvědomit, že to, že příspěvek nebyl automaticky schválen, ještě neznamená, že by měl být automaticky zamítnut. Tyto zprávy by měly být přeposlány moderátorům, a to

ze dvou důvodů. Zaprvé chcete, aby mohli zprávy posílat i ti, kteří členy mailing listu nejsou. Pokud má někdo dotaz nebo nějaký návrh, neměl by být jen kvůli jedné zprávě nucen se do mailing listu přihlásit. Zadruhé někdy mohou chtít přispívat účastníci mailing listu z jiné adresy, než pod kterou jsou přihlášeni. E-mailová adresa není spolehlivý způsob, jak někoho identifikovat, takže ji tak ani nelze používat.

2. **Filtrování příspěvků pomocí softwaru pro identifikaci spamu.**
Pokud to váš software pro mailing listy umožňuje (a většinou tomu tak je), můžete nechat zprávy filtrovat přes software pro identifikaci spamu. Automatická filtrace proti spamu není dokonalá a nikdy ani nebude, protože ti, kdo spam rozesílají, hledají stále nové metody, jak tyto filtry přelstít. Přesto může množství nevyžádaných zpráv, které se dostanou až do fronty pro moderátory, významně redukovat; protože samozřejmě platí, že čím je tato fronta delší, tím více času musí moderátoři strávit jejím procházením, takže každý automatický zásah je usnadněním. Nemáme zde prostor pro to podrobně popsat, jak filtr proti spamu nastavit. Na to se budete muset podívat do dokumentace softwaru pro správu mailing listů, který používáte (viz **Software** dále v této kapitole). Software pro mailing listy se často dodává se zabudovanými funkcemi na ochranu proti spamu, ale může být užitečné k němu přidat ještě další externí filtry. Mám dobré zkušenosti s následujícími dvěma: SpamAssassin (**http://spamassassin.apache.org/**) a SpamProbe (**http://spamprobe.sourceforge.net/**). Tím nechci nijak kritizovat všechny ostatní open source filtry proti spamu, kterých rozhodně není málo a z nichž některé jsou podle všeho velmi dobré. Pouze říkám, že s těmito dvěma mám osobní zkušenost a mohu je doporučit.

3. **Moderování.**
Pro zprávy, které nebyly povoleny automaticky, protože nepocházejí od člena mailing listu, a které prošly softwarem pro filtraci spamu (pokud jej používáte), je poslední etapou *moderování*. V praxi to znamená, že je tento e-mail přesměrován na zvláštní adresu, kde jeho obsah prozkoumá člověk-moderátor a buď jej schválí, nebo zamítne.

Schválení zprávy může mít dvě různé podoby: buď se bude vztahovat jenom na tuto konkrétní zprávu, nebo softwaru pro mailing list přikážete, aby povolil tento a všechny další e-maily ze stejné adresy. Téměř vždy budete chtít udělat to druhé, protože tím si do budoucna práci s moderováním trochu zjednodušíte. To, jak přesně schvalování probíhá, se trochu liší systém od systému, ale obvykle to vypadá tak, že zašlete odpověď na zvláštní adresu spolu s příkazem „accept“ (přijmout, tedy schválit tuto konkrétní zprávu) nebo „allow“ (povolit, tedy schválit tuto a budoucí zprávy).

Zamítnutí se obvykle provádí tak, že příslušný e-mail jednoduše ignorujete. Dokud software pro mailing list nedostane informaci o tom, že byla nějaká zpráva schválena, do mailing listu ji neodešle, takže jako moderátor dosáhnete svého cíle prostě tak, že příslušné oznámení necháte být. Někdy také můžete využít možnost zaslat odpověď s příkazem „reject“ (zamítnout) nebo „deny“ (odepřít), čímž automaticky zamítnete budoucí zprávy od stejného odesílatele, aniž by se vůbec dostaly do fáze moderování. Tyto příkazy jsou ovšem jen málokdy k něčemu dobré; smyslem moderování je především zamezit přívalu spamu, který ale jen málokdy přijde ze stejné adresy dvakrát.

Moderování nepoužívejte na nic jiného než na filtrování nežádoucích zpráv a příspěvků, které se zjevně netýkají tématu (off-topic) – například pokud někdo omylem odešle zprávu do nesprávného mailing listu. Systém pro moderování vám obvykle nabídne ještě možnost odesílateli přímo odpovědět, ale tu byste v žádném případě neměli používat k tomu, abyste odpovídali na otázky, které plným právem patří do mailing listu – a to ani v případě, kdy znáte odpověď z hlavy. Je důležité, aby komunita projektu měla přesnou představu o tom, na co se lidé ptají; musíte dát i ostatním šanci na danou otázku zareagovat nebo si přečíst odpovědi ostatních. Smyslem moderování mailing listu je pouze to, aby se v něm neobjevoval spam a příspěvky, které se netýkají tématu – nic jiného.

Skrývání adres v archivech

Aby se váš mailing list nemohl stát zdrojem adres pro ty, kdo rozesílají spam, obvykle se při archivaci nějakým způsobem e-mailové adresy znečitelňují, například tím, že nahradíte

`jnovak@domena.com`
za
`jnovak_AT_domena.com`
nebo
`jnovakNOSPAM@domena.com`

nebo použijete nějaký podobný způsob, který je pro člověka snadno odhalitelný. Automatický sběr adres pro rozesílání nevyžádané pošty často probíhá tak, že něčí skript prochází různé webové stránky – včetně on-line archivů vašeho mailing listu – a hledá v nich řetězce obsahující znak „@“. Výše uvedené způsoby zajistí, že jsou pro takové automatické hledače e-mailové adresy buď neviditelné, nebo nepoužitelné. Tím samozřejmě nijak nezabráníte tomu, aby se do vašeho mailing listu dostával spam, ale alespoň nebudete zvětšovat objem spamu, který chodí na osobní adresy jeho členů.

Skrývání adres může být kontroverzní téma. Někteří lidé mají tento systém rádi a budou překvapeni, pokud se ve vašich archivech nebude provádět automaticky. Jiní si myslí, že jim to přidělává práci, protože před použitím takto upravené adresy je potřeba ji nejprve vrátit do původní podoby. Někteří naopak tvrdí, že to je neefektivní, protože pokud bude systém pro skrývání adres používán konzistentně, může s tím skript sbírající adresy počítat a příslušné úpravy provést automaticky. Existují ale ověřené důkazy, že skrývání adres efektivní skutečně je,
viz **http://www.cdt.org/speech/spam/030319spamreport.shtml.**

V ideálním případě by měl software pro správu mailing listu ponechat tuto volbu na každém jednotlivém přispěvateli, ať už prostřednictvím speciální hlavičky typu ano/ne nebo nastavením příslušné předvolby v uživatelském účtu. Přiznám se ale, že nevím o žádném softwaru, který by něco takového nabízel, a to ani pro jednotlivé zprávy, ani pro uživatele, takže prozatím musí tohle rozhodnutí za všechny udělat správce mailing listu (pokud tedy archivační software takovou možnost vůbec nabízí, což neplatí ve všech případech). Osobně bych hlasoval spíše pro ukrývání adres, ale úplně přesvědčen

o jeho výhodách nejsem. Někteří lidé se zveřejňování svých e-mailových adres na internetu a vůbec kdekoliv, kde by je mohly automatické sběrače najít, velmi pečlivě vyhýbají; pokud pak veškerou jejich snahu zmaříte tím, že váš archiv adresy nijak skrývat nebude, můžete je tím jenom zklamat. Je sice pravda, že tato volba komplikuje uživatelům život, ale opravdu jenom velmi málo, protože pokud se s někým potřebujete spojit, je převedení adresy do původní podoby celkem triviální záležitost.

Nezapomínejte ale na to, že celá věc jsou klasické závody ve zbrojení. Než se k vám tento text dostane, možná už se sběrače adres vyvinou natolik, že dokážou ty nejběžnější způsoby ukrývání snadno rozpoznat a my budeme muset vymyslet zase něco jiného.

Identifikace a správa hlaviček

Odběratelé mailing listu si často budou chtít jeho zprávy ukládat do zvláštní složky, odděleně od své ostatní pošty. To mohou jejich programy pro čtení e-mailů dělat automaticky na základě *hlaviček* zprávy. Jako hlavičky se označují pole na začátku e-mailu, která určují odesilatele, příjemce, předmět, datum a řadu dalších různých informací o zprávě. Některé hlavičky jsou dobře známé a v podstatě povinné:

```
Od: …
Komu: …
Předmět: …
Datum: …
```

Další jsou nepovinné, ale přesto celkem běžné. Není například nutné, aby e-maily měly také hlavičku

```
Reply-to: odesilatelova@mailova.adresa
```

Většina zpráv ji ale obsahuje, protože příjemcům umožňuje se jednoduše spojit s jejím autorem (což se zvlášť hodí v případě, kdy autor musel zprávu odeslat z jiné adresy, než na jakou by měla být směřována odpověď).

Některé programy pro čtení pošty nabízejí snadno použitelné rozhraní pro třídění zpráv na základě obsahu hlavičky Předmět. Mnozí lidé proto žádají, aby mailing list na začátek předmětu každé zprávy přidal určitý řetězec; mohou pak totiž svému softwaru pro čtení pošty přikázat, aby veškeré zprávy s tímto řetězcem zařadil do příslušné složky. Pokud by tedy původní autor napsal:

```
Předmět: Vytváření release 2.5
```

ukázala by se zpráva v mailing listu takto:

```
Předmět: [diskuzni@list.priklad.org] Vytváření release 2.5
```

Ačkoliv většina softwaru pro správu mailing listů tuto volbu nabízí, důrazně doporučuji, abyste ji nezapínali. Problém, který se tím řeší, lze zrovna tak snadno řešit i mnohem nevtíravěji a v poli předmět není zas tolik místa, aby se jím dalo plýtvat. Zkušení uživatelé mailing listu často projíždějí předměty zpráv, které během dne přišly, aby se rozhodli, co budou číst a na co budou reagovat. Pokud bude na začátku každého předmětu ještě jméno mailing listu, může být jeho konec vytlačen na pravém kraji mimo obrazovku, kde nebude vidět. Tím se zakryje informace, podle které se uživatelé rozhodují, zda zprávu otevřou, což snižuje celkovou funkčnost mailing listu.

Místo toho, abyste zasahovali do hlavičky Předmět, naučte své uživatele, jak využít ostatní standardní hlavičky, počínaje hlavičkou Komu, která by měla obsahovat jméno mailing listu:

```
Komu: <diskuzni@list.priklad.org>
```

Každý program pro čtení pošty, který je schopný filtrovat podle hlavičky Předmět, by měl být zrovna tak schopen filtrovat i podle hlavičky Komu.

U mailing listů se očekává použití ještě několika dalších nepovinných, i když celkem standardních hlaviček. Filtrování na základě těchto hlaviček je ještě spolehlivější než podle obsahu polí „Komu“ nebo „Kopie“, protože tyto zvláštní hlavičky jsou do každé zprávy automaticky přidávány softwarem pro správu mailing listu, a máte tedy jistotu, že budou přítomné vždy:

```
list-help: <mailto:diskuze-napoveda@list.priklad.org>
list-unsubscribe: <mailto:diskuze-odhlasit@list.priklad.org>
list-post: <mailto:diskuzni@list.priklad.org>
Delivered-To: mailing list diskuzni@list.priklad.org
Mailing-List: kontakt diskuze-napoveda@list.priklad.org; spravuje ezmlm
```

Většina z nich se vysvětluje celkem sama. Více podrobností najdete na **http://www.nisto.com/listspec/list-manager-intro.html.** Pokud potřebujete opravdu důkladnou, přesnou specifikaci, podívejte se na **http://www.faqs.org/rfcs/rfc2369.html.**

Všimněte si, že tyto hlavičky předpokládají, že pokud máte mailing list, který se jmenuje „list“, pak také máte administrativní adresy „list-help“ (nápověda) a „list-unsubscribe“ (odhlášení). Kromě nich obvykle existuje ještě „list-subscribe“ (přihlášení) pro zájemce o účast a „list-owner“ (vlastník) pro ty, kdo se chtějí spojit se správci mailing listu. Založení těchto a mnoha jiných adres závisí na tom, jaký software pro správu mailing listu použijete – více najdete v jeho dokumentaci. Úplný seznam všech těchto zvláštních adres i s jejich popisem je obvykle zaslán každému novému uživateli po přihlášení jako součást automatického „uvítací zprávy“. Kopii této zprávy dostanete pravděpodobně i vy sami. Pokud ji nedostanete, požádejte někoho jiného, ať vám ji přepošle, abyste věděli, co vaši uživatelé uvidí, když se do mailing listu přihlásí. Tuto kopii mějte po ruce, abyste mohli odpovídat na otázky týkající se funkcí mailing listu, nebo ji ještě lépe umístěte někam na své stránky. Pokud pak někdo svou kopii těchto instrukcí ztratí a vznese dotaz „Jak se můžu odhlásit z mailing listu?“, můžete mu jednoduše předat URL.

Některý software pro mailing listy nabízí možnost připojit na konec každé zprávy instrukce pro odhlášení. Pokud je tato volba k dispozici, zapněte ji. Ke každé zprávě se přidá jen pár řádek, navíc na místo, kde to nikomu nevadí, ale může vám to ušetřit spoustu času tím, že se zmenší počet lidí, kteří se vás (nebo v horším případě i celého mailing listu) budou ptát, jak se odhlásit.

Velká debata o Reply-to

Výše v části **Vyhněte se soukromým diskuzím** jsem upozornil na to, jak je důležité, aby diskuze probíhaly na veřejných fórech, a mluvil jsem o tom, že někdy je potřeba aktivně zasáhnout, abychom zabránili tomu, že konverzace sklouzne do podoby soukromých diskuzí; celá tato kapitola má za svůj cíl nastavit komunikační software projektu tak, aby za vás zastal co nejvíc práce. Takže pokud vám software pro správu mailing listu nabídne způsob, jak v něm veškeré diskuze automaticky udržet, mohli byste logicky usoudit, že tuhle funkci určitě chcete zapnout.

Jenže zas tak jednoduché to není. Taková možnost sice existuje, ale má i své poměrně velké nevýhody. Otázka, zda ji použít nebo ne, patří při debatách o správě mailing listů k těm nejožehavějším tématům – tedy jistě, není to zrovna něco, co by plnilo přední stránky novin, ale v projektech svobodného softwaru se to může čas od času vynořit. V následujícím textu tuto funkci popíšu a uvedu hlavní argumenty obou stran; z toho pak odvodím nejlepší doporučení, jaké vám k celé věci můžu dát.
Celá funkce je velmi jednoduchá: software pro mailing list umí, pokud si to přejete, automaticky nastavit u každé zprávy hlavičku Reply-to tak, že se odpovědi přesměrují do mailing listu. To znamená, že ať už do hlavičky Reply-to původní odesílatel vložil cokoliv (popřípadě když ji vůbec nepoužil), do mailing listu se e-mail dostane v podobě, kdy bude zmíněná hlavička obsahovat toto:

```
Reply-to: diskuzni@list.priklad.org
```

Na první pohled se to zdá jako dobrý nápad. Prakticky veškerý software pro čtení pošty se obsahem hlavičky Reply-to řídí, takže pokud kdokoli na zprávu odpoví, bude jeho reakce automaticky zaslána celému mailing listu a nejen odesílateli původní zprávy. Ten, kdo na zprávu reaguje, samozřejmě může ručně změnit, kam zpráva poputuje, ale důležité je, že pokud tak neučiní, budou odpovědi přesměrovány do mailing listu. Jde o skvělý příklad využití technologie k podnícení spolupráce.

Bohužel to ale má i své nevýhody. První z nich je problém nazývaný *„nevím, jak se dostat zpátky domů“* – původní odesílatel někdy do pole Reply-to vloží svou „opravdovou“ e-mailovou adresu, protože z nějakého důvodu odesílal e-mail z jiné adresy, než kde by chtěl přijmout odpověď. Ti, kdo e-maily čtou a odesílají stále ze stejného místa, tento problém nemají a mohou být dokonce překvapeni, že vůbec existuje. Ale pro ty, kdo používají nestandardní konfiguraci e-mailu nebo kteří nemohou ovlivnit to, jak vypadá obsah hlavičky „Od“ (třeba proto, že e-mail posílají z práce a tyto věci spravuje jejich IT oddělení), může být použití „Reply-to“ jediným způsobem, jak zajistit, aby se k nim odpovědi dostaly. Pokud takový člověk zašle zprávu do mailing listu, do nějž není přihlášen, je obsah hlavičky Reply-to velmi důležitou informací. Pokud ji software pro mailing list přepíše, nemusí se reakcí na svou zprávu dočkat nikdy.

Druhá nevýhoda souvisí s očekáváním uživatelů a je to podle mého názoru ten nejsilnější argument proti úpravám obsahu Reply-to. Většina zkušených uživatelů e-mailu už je zvyklá na dva základní způsoby odpovídání: *odpovědět všem* a *odpovědět autorovi.* Veškerý moderní software pro čtení pošty má pro obě zmíněné akce samostatná tlačítka. Uživatelé vědí, že pokud chtějí odpovědět všem (tedy včetně adresy mailing listu), měli by použít „odpovědět všem"; pokud chtějí odpovědět soukromě autorovi, měli by použít „odpovědět autorovi". I když chcete všechny vybízet, aby své odpovědi zasílali pokud možno vždy do mailing listu, existují i situace, kdy je lepší někomu odpovědět soukromě – například pokud chcete autorovi původní zprávy sdělit něco důvěrného, co by nebylo vhodné probírat prostřednictvím veřejného mailing listu.

Teď uvažte, co se stane, když mailing list přepíše Reply-to původního odesílatele. Odpovídající stiskne tlačítko „odpovědět autorovi" a bude předpokládat, že tím odešle soukromou zprávu původnímu odesílateli zprávy. Protože takové chování očekává, nemusí se obtěžovat tím, aby adresu příjemce nové zprávy zkontroloval. Sepíše soukromou, důvěrnou zprávu, která například obsahuje nepříjemné informace o někom jiném z mailing listu, a stiskne tlačítko odeslat. O několik minut později, aniž by to očekával, se ale zpráva objeví v mailing listu. Jistě, teoreticky si měl pole příjemce před odesláním zkontrolovat a o obsahu hlavičky Reply-to neměl činit předčasné závěry. Jenže do hlavičky Reply-to umisťují téměř všichni odesílatelé zpráv svou osobní adresu (respektive to za ně dělá jejich poštovní software), takže ti, kteří e-maily používají už řadu let, to pochopitelně i očekávají. Dokonce je celkem běžné, že pokud někdo záměrně nastaví Reply-to na nějakou jinou adresu (jako například adresu mailing listu), pak se o tom obvykle zmíní v textu zprávy, aby ostatní nebyli překvapeni tím, co se stane, když na zprávu odpoví.

Vzhledem k tomu, že tohle neočekávané chování může mít potenciálně velmi závažné následky, bych osobně při konfigurování softwaru pro správu mailing listu upřednostňoval, aby hlavičku Reply-to nechával v původní podobě. Jsem toho názoru, že toto je jeden z případů, kdy může mít využití technologie k podpoře spolupráce potenciálně nebezpečné vedlejší účinky. To ale neznamená, že by zastánci této funkce neměli na své straně některé velmi přesvědčivé argumenty. Ať už si vyberete jakýkoliv způsob, občas se setkáte s lidmi, kteří se v mailing listu budou ptát, proč jste nezvolili ten druhý. A protože to není zrovna něco, z čeho byste v mailing listu chtěli udělat hlavní diskuzní téma, není špatný nápad mít už předem připravenou odpověď, která celou debatu spíše ukončí, než aby ji povzbuzovala. Tato odpověď by v žádném případě neměla tvrdit, že vaše rozhodnutí (ať už je jakékoliv) je to jediné správné a že to druhé nedává smysl (i kdybyste věděli, že tomu tak je). Místo toho byste měli říct, že to je už velmi starý spor a že existují dobré argumenty pro i proti. Žádná volba neuspokojí všechny uživatele, takže jste učinili to nejlepší rozhodnutí, jaké jste mohli. Zdvořile požádejte, aby se uvedené téma znovu neprobíralo, pokud někdo nemá něco opravdu nového, co by k tomu řekl; pak už jen do tohoto vlákna nezasahujte a doufejte, že časem zanikne samo.

Někdo možná navrhne, aby se pro jednu nebo druhou možnost hlasovalo. Pokud chcete, můžete to udělat, ale osobně nemám pocit, že by to v tomto případě mohlo vést k uspokojivému řešení. Negativní následky toho, že někdo, kdo takové chování nečekal, omylem pošle soukromou zprávu do veřejného mailing listu, jsou obrovské; naproti tomu někomu občas vytknout, že má odpovídat celému mailing

listu a nejen vám osobně, je sice poněkud otravné, ale nijak zásadní problém to není. Nejsem proto přesvědčen, že by měla v této situaci většina (pokud to tedy vůbec většina je) právo vystavovat menšinu tak závažnému riziku.

Celý problém má i své další stránky, které zde ale uvádět nechci; věnoval jsem se jenom těm, které mi připadají nejdůležitější. Úplnou diskuzi naleznete ve dvou stěžejních dokumentech, které bývají obvykle citovány těmi, kdo se této debaty účastní:

- Leave Reply-to alone (Nechte Reply-to na pokoji), od *Chipa Rosenthala*
  http://www.unicom.com/pw/reply-to-harmful.html
- Set Reply-to to list (Nastavte Reply-to na adresu mailing listu), od *Simona Hilla*
  http://www.metasystema.net/essays/reply-to.mhtml

I když, jak jsem uvedl výše, se v této otázce spíše přikláním na jednu stranu, nemám pocit, že by na ni existovala nějaká „správná“ odpověď a jsem samozřejmě členem mnoha mailing listů, které obsah hlavičky Reply-to mění. Nejdůležitější je, abyste si ten či onen způsob zvolili brzy a abyste se pokud možno nikdy nenechali vtáhnout do debaty, který je lepší.

Dvě ideální řešení

Jednoho dne někdo dostane skvělý nápad implementovat v programu pro čtení pošty tlačítko *odpovědět do mailing listu*. Tato funkce využije dříve zmíněné hlavičky zprávy, aby z nich vyčetla adresu mailing listu. Odpověď pak zašle jen na tuto adresu a všechny ostatní adresy příjemců vynechá, protože většina z nich je stejně do mailing listu přihlášena. Tato funkce se časem rozšíří i do ostatních programů pro čtení e-mailů a celá debata zanikne. (Ve skutečnosti už program, který takovou funkci má, existuje; jmenuje se **Mutt**.[13])

Ještě lepším řešením by bylo, kdyby si nastavování Reply-to mohl navolit každý člen mailing listu ve svých předvolbách. Ti, kteří by dávali přednost tomu, aby jim mailing list hlavičku Reply-to upravoval (ať už u jejich vlastních příspěvků nebo u příspěvků od ostatních), by si to tak nastavili; ti, kteří mají opačný názor, by ponechali Reply-to v původním nastavení. Nicméně nevím o žádném softwaru pro správu mailing listů, který by takové nastavení pro každého člena zvlášť umožňoval. Prozatím se zdá, že to lze nastavit jen globálně.[14]

---

[13] Krátce po vydání této knihy mi napsal **Michael Bernstein** toto: „Existují i další e-mailoví klienti, nejen Mutt, kteří funkci odpovědět mailing listu mají. Jedním z nich je například Evolution, který pro ni nemá zvláštní tlačítko, ale klávesovou zkratku Ctrl+L.“

[14] Poté, co jsem tento oddíl napsal, jsem zjistil, že existuje přinejmenším jeden systém pro správu mailing listů, který tuto funkci nabízí: **Siesta**. Více podrobností naleznete v tomhle článku: **http://www.perl.com/pub/a/2004/02/05/siesta.html**

Archivace

Technické detaily nastavení archivace mailing listu jsou u každého softwaru, který se pro jeho provoz používá, trochu odlišné a přesahují rámec této knihy. Při výběru a konfiguraci archivačního nástroje se zaměřte na následující vlastnosti:

Rychlá aktualizace
Členové mailing listu budou často chtít odkázat na archivovanou zprávu, která byla zaslána jen před hodinou nebo dvěma. Pokud je to možné, měl by archivační nástroj ukládat každou zprávu okamžitě – takže když se zpráva objeví v mailing listu, měla by už být přítomna i v archivu. Pokud taková možnost není k dispozici, pak se jej alespoň pokuste nastavit tak, aby archiv aktualizoval zhruba každou hodinu. (Některé archivační nástroje standardně spouštějí aktualizační proces přes noc, ale to je pro aktivní mailing list prakticky nepoužitelné.)

Stabilita odkazů
Jakmile je zpráva archivována na určité URL, měla by tam zůstat dostupná už navždy – nebo zkrátka co nejdéle to bude možné. Dokonce i v případě, kdy jsou archivy znovu sestaveny, obnoveny ze záloh nebo nějak jinak opraveny, by měly už dříve zveřejněné URL zůstat beze změny. Stabilní odkazy umožňují internetovým vyhledávačům vaše archivy indexovat, což přináší velkou výhodu pro uživatele, kteří v nich hledají odpovědi. Stabilita odkazů je důležitá i proto, že na zprávy a diskuzní vlákna se často odkazuje ze systému pro sledování chyb (viz **Systém pro sledování chyb** dále v této kapitole) nebo z jiných dokumentů projektu.

V ideálním případě by měl software pro mailing list v okamžiku distribuce zprávy příjemcům přidat do hlavičky i URL této zprávy v archivu, popřípadě alespoň tu část URL, pod níž je možné ji najít. Díky tomu mohou ti, kteří mají kopii této zprávy, získat stabilní odkaz, aniž by museli archiv vůbec otevírat, což je pro ně užitečné, protože jakákoliv operace, která vyžaduje použití webového prohlížeče, je automaticky časově o něco náročnější. Po pravdě nevím, zda vůbec nějaký software pro správu mailing listů takovou funkci nabízí; u těch, které jsem používal já, to tak bohužel nebylo. Přesto je to ale něco, po čem byste se mohli podívat (a pokud píšete software pro mailing listy, zvažte, prosím, jestli byste takovou funkci nemohli přidat).

Zálohování
Zálohování archivů by mělo být relativně jednoduché, stejně jako jejich obnova ze zálohy. Jinými slovy archivační nástroj byste neměli považovat za černou skříňku. Někdo z vašeho projektu by měl vědět, kam se zprávy ukládají a jak se dá v případě potřeby archiv z tohoto úložiště obnovit. Záznamy v archivech jsou totiž velmi cenné; pokud je projekt ztratí, přijde o značnou část své kolektivní paměti.

Podpora třídění podle vláken
Z každé jednotlivé zprávy by mělo být možné přejít na její *vlákno* (thread), tedy skupinu navzájem souvisejících zpráv, do nichž patří. Každé diskuzní vlákno by mělo mít i své vlastní URL, jiné, než jsou URL jednotlivých zpráv ve vlákně.

Možnost vyhledávání
Nástroj pro práci s archivem, který nepodporuje vyhledávání, a to jak v obsahu zpráv, tak podle autorů a předmětů, není skoro k ničemu. Je pravda, že některé archivační nástroje podporují vyhledávání tak, že požadavek jednoduše předají externímu vyhledávači, jako například Google. To je sice přijatelné, ale přímá podpora vyhledávání je obvykle lepší, protože pak můžete například upřesnit, že chcete vyhledávat pouze v předmětu a ne v těle zprávy.

Výše uvedené body tvoří jen orientační seznam, který by vám měl pomoct při výběru a nastavení archivačního nástroje. Na to, jak přimět účastníky projektu, aby archiv skutečně používali způsobem, který bude pro všechny výhodný, se podíváme v dalších kapitolách, zejména v části **Nápadné využívání archivů.**

Software

V tomto oddílu uvedu přehled několika open source nástrojů pro správu a archivaci mailing listů. Pokud má server, na němž chcete svůj projekt hostovat, už připravené nějaké standardní nastavení, pak se možná výběrem nástroje nebudete muset vůbec zabývat. Pokud jej ale budete muset nainstalovat sami, můžete zvážit následující možnosti. Mezi ty, které jsem osobně používal, patří Mailman, Ezmlm, MHonArc a Hypermail, čímž ale neříkám, že ty ostatní nejsou dobré (a samozřejmě existují pravděpodobně ještě další nástroje, které jsem nenašel, takže následující seznam rozhodně nepokládejte za vyčerpávající).

Software pro správu mailing listů:

- Mailman – **http://www.list.org/**
  (Má zabudovaný archivační nástroj a rozhraní pro zásuvné moduly externích archivačních nástrojů.)
- SmartList – **http://www.procmail.org/**
  (Navržen pro použití spolu se systémem pro zpracování e-mailů Procmail.)
- Ecartis – **http://www.ecartis.org/**
- ListProc – **http://listproc.sourceforge.net/**
- Ezmlm – **http://cr.yp.to/ezmlm.html**
  (Navržen pro spolupráci se systémem pro doručování e-mailů Qmail.)
- Dada – **http://mojo.skazat.com/**
  (Navzdory tomu, že se to webové stránky projektu snaží z nějakého podivného důvodu skrýt, je to svobodný software vydaný pod GNU General Public License. Má i zabudovaný archivační nástroj.)

Software pro archivaci mailing listů:

- MHonArc – **http://www.mhonarc.org/**
- Hypermail – **http://www.hypermail.org/**
- Lurker – **http://sourceforge.net/projects/lurker/**
- Procmail – **http://www.procmail.org/**
  (Tento systém pro zpracování e-mailů, který byl navržen pro spolupráci s programem SmartList, lze nakonfigurovat i pro archivaci.)

Správa verzí

Systém pro správu verzí (version control system), popřípadě *systém řízení oprav* (revision control system) využívá kombinaci několika technologií a postupů pro to, aby mohl sledovat a řídit změny v souborech projektu, zejména zdrojových kódů, dokumentace a webových stránek. Pokud jste nikdy žádný systém pro správu verzí nepoužívali, pak první věc, kterou byste měli udělat, je najít někoho, kdo s ním už zkušenosti má, a přemluvit jej k zapojení do projektu. V dnešní době už budou všichni očekávat, že přinejmenším zdrojový kód vašeho projektu bude systémem pro správu verzí řízen, a pokud zjistí, že jej nepoužíváte, pravděpodobně nebudou brát celý projekt moc vážně.

Důvod, proč se použití systémů pro správu verzí stalo standardem, je to, že během projektu pomáhají téměř se vším: s komunikací mezi vývojáři, správou vydávání nových verzí, opravováním chyb, oddělováním stabilního kódu od experimentálních větví, přiřazením konkrétních změn jejich autorům a schvalováním úprav. Ve všech těchto aspektech je systém pro správu verzí jakýmsi centrálním koordinačním nástrojem. Jádrem systému pro správu verzí je *systém pro řízení změn* (change management), který provádí identifikaci všech jednotlivých změn, které byly v souborech projektu provedeny. Ke každé z nich navíc připojuje metadata, jako je datum změny a její autor, a tyto informace pak zpřístupňuje komukoliv, kdo o ně požádá, ať už jakýmkoliv způsobem. Je to komunikační mechanismus, ve kterém je základní jednotkou informace změna.

V této podkapitole nebudeme probírat všechny aspekty používání systému pro správu verzí. Je to natolik velké téma, že se jím budeme zabývat v rámci jednotlivých kapitol v průběhu celé knihy. Zde se zaměříme na to, jak systém pro správu verzí vybrat a jak jej nastavit tak, aby podporoval rozvoj vývoje.

Slovníček pojmů správy verzí

Pokud jste systém pro správu verzí nikdy nepoužívali, pak vás to tato kniha nenaučí, ale ani základní pojednání o jeho funkcích se neobejde bez uvedení alespoň několika klíčových pojmů. Tyto termíny jsou použitelné nezávisle na tom, jaký konkrétní systém pro správu verzí použijete – jsou to ta nejzákladnější slova používaná při spolupráci na síti a v dalších částech této knihy se s nimi setkáte zcela běžně. I kdyby na světě neexistoval žádný systém pro správu verzí, problém správy změn by tu zůstal; tato slova nám poskytují způsob, jak o nich jednodušeji diskutovat.

„Verze“ versus „revize“
Slovo *verze* se někdy používá jako synonymum pro „revize“, ale já ho tak v této knize používat nebudu, protože jej lze snadno zaměnit se slovem „verze“ ve smyslu verze programu, tedy s číslem releasu nebo řady – například „Verze 1.0“. Protože spojení „správa verzí“ je už ale zavedené, budu jej používat i nadále jako synonymum pro „správa revizí“ a „správa změn“.

commit
Provedení změny v projektu, tedy přesněji řečeno uložení změny v databázi systému pro správu verzí takovým způsobem, aby mohla být začleněna do budoucích releasů (tedy nových verzí) projektu. Vedle podstatného jména „commit“ (v překladu zhruba „zápis“) se lze setkat i se slovesy jako „commitnout“, „commitovat“ atd. Podstatné jméno „commit“ lze v zásadě chápat jako synonymum pro „změnu“. Například takto: „Právě jsem commitnul opravu pro tu chybu, kvůli které padaly servery pod Mac OS X. Mohl by ses na ten commit prosím podívat a zkontrolovat, jestli jsem tam správně použil alokátor?“

zpráva protokolu (log message)
Krátký komentář připojený ke každému commitu, jenž popisuje, v čem spočívá a čemu slouží. Tyto zprávy patří mezi ty vůbec nejdůležitější dokumenty celého projektu – vytvářejí jakýsi most mezi vysoce technickým jazykem jednotlivých změn v kódu a více uživatelsky orientovaným jazykem popisujícím nové funkce, opravy chyb a celkový postup projektu. Dále se v tomto oddílu ještě podíváme na to, jak zprávy protokolu dostat k příslušnému publiku. Několik způsobů, jak přimět přispěvatele k tomu, aby psali do protokolu zprávy, které budou stručné, ale výstižné, naleznete v části **Kodifikace tradic** v kapitole **6. Komunikace.**

aktualizace (update)
Požadavek na to, aby byly commity ostatních zapsány do vaší lokální kopie projektu, která tím pádem bude odpovídat nejnovějšímu stavu. To je něco, co se provádí velmi často. Většina vývojářů aktualizuje svůj kód několikrát denně, aby měli záruku, že spouštějí v zásadě totéž co ostatní vývojáři; pokud tedy najdou chybu, mohou si být celkem jisti, že ještě nebyla opravena. Například takto: „Všiml jsem si, že kód pro indexování pokaždé zahodí poslední byte. Je to nová chyba?“ „Je, ale byla opravena minulý týden. Zkus aktualizovat a měla by zmizet.“

úložiště (repository)
Databáze, ve které jsou změny uloženy. Některé systémy pro správu verzí jsou centralizované, což znamená, že existuje jediné hlavní úložiště, které uchovává všechny změny projektu. Jiné jsou naopak decentralizované; v nich má každý vývojář své vlastní úložiště, mezi nimiž lze změny libovolně posílat tam a zpět. Systém pro správu verzí sleduje závislosti mezi změnami, a když nastane doba vydání nové verze (release), je pro ni schválena určitá množina změn. Otázka, zda je lepší centralizovaný, nebo decentralizovaný systém, je jednou z těch nejdéle probíhajících svatých válek v oblasti vývoje softwaru vůbec. Zkuste se debatám na toto téma v mailing listech svého projektu úplně vyhnout.

checkout
Proces získání kopie projektu z úložiště. Při checkoutu se obvykle vytvoří adresářový strom označovaný jako „pracovní kopie“ (viz níže); změny provedené v tomto stromu mohou být promítnuty jako commit zpět do původního úložiště. V některých decentralizovaných systémech pro správu verzí je každá pracovní kopie sama o sobě úložištěm, z nějž lze změny šířit do libovolného jiného úložiště, pokud je nastaveno pro jejich přijetí.

pracovní kopie (working copy)
Soukromý adresářový strom vývojáře, který obsahuje zdrojové kódy projektu a případně i jeho webové stránky a jiné dokumenty. Pracovní kopie obsahuje navíc ještě metadata, která přidává systém pro správu verzí. Označují, z jakého úložiště pracovní kopie pochází, jaké „revize“ (viz níže) souborů se v ní nacházejí atd. Obecně platí, že každý vývojář má svou vlastní pracovní kopii, v níž provádí a testuje změny a z níž zasílá commity.

revize, změna, sada změn (revision, change, changeset)
Pojmem „revize“ se obvykle označuje konkrétní podoba určitého souboru nebo adresáře. Pokud například v projektu existuje revize 6 souboru S a někdo provede commit nějaké změny S, vznikne revize 7 souboru S. Některé systémy používají pojmy „revize“, „změna“ nebo „sada změn“ také k popsání určité množiny změn tvořících jeden ucelený commit.

V různých systémech pro správu verzí mohou být tyto tři pojmy od sebe nějakým způsobem odlišeny, ale v zásadě jde pořád o totéž: možnost, jak identifikovat konkrétní okamžik v historii souboru nebo sady souborů (dejme tomu bezprostředně před a po opravě nějaké chyby). Například takto: „Ano, to bylo opraveno v revizi 10“ nebo „to bylo opraveno v revizi 10 souboru foo.c.“ Pokud někdo mluví o souboru nebo o několika souborech najednou, aniž by uvedl jejich konkrétní revizi, pak se obecně předpokládá, že má na mysli tu nejnovější.

diff
Textová reprezentace změny. Diff ukazuje, které řádky byly změněny a jak; obvykle se v něm objevuje ještě několik předcházejících a následujících řádků kvůli kontextu. Vývojář, který už určitý úsek zdrojového kódu zná, obvykle může diff celkem snadno přečíst a pochopit, co změna způsobila, popřípadě v ní najít chyby.

tag
Jmenovka pro konkrétní revize nějaké sady souborů. Tagy se většinou používají k trvalému zachycení zajímavých momentů projektu v podobě takzvaného snímku (snapshot). Tagem se například obvykle označuje každý release uvolněný na veřejnost, aby mohl každý přímo ze systému pro správu verzí získat přesnou podobu souborů, respektive revizí, které tento release tvoří. Nejběžnější podoby tagů znějí například `Release_1_0, Delivery_00456` atd.

větev (branch)
Kopie projektu, která je řízena systémem pro správu verzí, ale je izolovaná, takže změny provedené na této větvi neovlivní zbytek projektu a naopak – s výjimkou případů, kdy jsou změny

záměrně zkopírovány z jedné strany na druhou (viz heslo **„merge“** níže). Větve jsou známy také jako „vývojové řady“. I když projekt větve nepoužívá, často uslyšíte, že vývoj probíhá v „hlavní větvi“, které se také říká „hlavní linie“ nebo „kmen“ (trunk).

Větve nabízejí způsob, jak od sebe vzájemně izolovat různé vývojové řady. Jedna větev může být například určena pro experimentální vývoj, který by mohl být pro hlavní větev příliš nestabilní. Nebo naopak můžete větev použít jako místo, kde bude stabilizován nový release. V procesu přípravy nové verze může vývoj nerušeně pokračovat v hlavní větvi úložiště, zatímco ve větvi pro vydání nejsou povoleny žádné změny s výjimkou těch, které schválí správci release. To znamená, že příprava nové verze nemusí mít vůbec žádný vliv na vývoj jako takový. Podrobnější diskuzi o práci s větvemi najdete v části **Používejte větve, abyste nezpomalovali vývoj** níže v této kapitole.

merge (někdy se používá i termín port)
Začlenění změny z jedné větve do jiné. Sem spadá i šíření změn z hlavního kmene do nějaké větve a naopak. To jsou ostatně ty vůbec nejběžnější typy merge – přenášení změn mezi dvěma vedlejšími větvemi je poměrně neobvyklé. Více se o tomto druhu merge dozvíte v části **Jedinečnost informace**.

Pojem „merge“ má ještě jeden význam, který s tím prvním souvisí. Označuje se tak to, co systém pro správu verzí provádí v situaci, kdy dva různí uživatelé změní stejný soubor, ale jejich změny se nepřekrývají. Protože jsou na sobě tyto dvě změny zcela nezávislé, získá při aktualizaci své lokální kopie tohoto souboru (která už jednu z těchto změn obsahuje) každý z nich i tu změnu, kterou provedl ten druhý. To se stává velmi často, zvlášť u projektů, kde na stejném kódu pracuje více lidí. Pokud se ovšem dvě různé změny překrývají, je výsledkem „konflikt“ – viz níže.

konflikt (conflict)
Nastává, když se dva lidé pokusí zapsat různé změny na stejném místě kódu. Všechny systémy pro správu verzí konflikty detekují automaticky a přinejmenším jednu ze zúčastněných osob pak upozorní na to, že jsou její změny v konfliktu se změnami někoho jiného. V takovém případě je na dané osobě, aby konflikt vyřešila a toto řešení předala systému pro správu verzí.

uzamčení (lock)
Způsob, jak oznámit, že určitý soubor nebo adresář chcete měnit výhradně vy. Například takto: „Na webové stránky teď nejde poslat žádný commit. Zdá se, že si je Alfréd všechny uzamkl, aby mohl v klidu opravit obrázky na pozadí.“ Možnost zamykat soubory nenabízejí všechny systémy pro správu verzí; to, že ji nabízejí, ale ještě neznamená, že ji musíte použít. To proto, že souběžně probíhající vývoj je ve světě open source považován za normální a bránění ostatním lidem v přístupu je (obvykle) s tímto ideálem v rozporu.

O systémech pro správu verzí, které vyžadují, aby byly commity prováděny na zamčených souborech, se říká, že používají model *zamknout – změnit – odemknout* (lock-modify-unlock). Pro ty, které zamykání nevyžadují, se používá termín *kopírovat – změnit – začlenit* (copy-modify-merge).

Vynikající, podrobné vysvětlení a srovnání těchto dvou modelů naleznete na **http://svnbook.red-bean.com/svnbook-1.0/ch02s02.html.**

Obecně platí, že pro vývoj open source je lepší ten druhý model, a podporují jej i všechny systémy pro správu verzí, o kterých budeme v této knize hovořit.

Výběr systému pro správu verzí

V době psaní tohoto textu byly nejpopulárnějšími systémy pro správu verzí ve světě svobodného softwaru *Concurrent Versions System* (CVS, **http://www.cvshome.org/**) a *Subversion* (SVN, **http://subversion.tigris.org/**).

Systém CVS už existuje poměrně dlouho. Většina zkušených vývojářů ho dobře zná, dělá víceméně to, co potřebujete, a protože je populární už tak dlouho, pravděpodobně se nezamotáte do dlouhých debat o tom, zda to byla správná volba nebo ne. CVS má ale i své nevýhody. Neexistuje v něm jednoduchý způsob, jak odkázat na změnu provedenou ve více souborech najednou; neumožňuje ani přejmenovat nebo kopírovat soubory, které spravuje (takže pokud potřebujete přeorganizovat adresářový strom se zdrojovými kódy poté, co byl projekt odstartován, může vás to stát spoustu práce). Jeho podpora slučování změn není úplně ideální a neumí si moc dobře poradit s velkými a binárními soubory. Některé jeho operace jsou navíc velmi pomalé v případech, kdy se týkají velkého množství souborů.

Žádný z těchto nedostatků CVS ale není úplně zásadní a nadále zůstává celkem populární volbou. Nicméně v posledních několika letech se do popředí dostává mladší systém jménem Subversion, zejména u novějších projektů.[15] Pokud zahajujete nový projekt, doporučil bych vám Subversion.

Na druhou stranu je ale pravda, že tady nejsem příliš objektivní rádce, protože jsem do projektu Subversion sám zapojen. V posledních několika letech se navíc objevila řada nových open source systémů pro správu verzí. Seznam všech, o kterých vím, seřazený zhruba podle jejich popularity, najdete v příloze **A. Svobodné systémy pro správu verzí.** Z tohoto seznamu vyplývá, že z porovnávání jednotlivých systémů pro správu verzí by se mohl snadno stát celoživotní výzkumný projekt. Nutnosti rozhodnout ale možná budete ušetřeni, protože už to za vás udělal hostingový server. Pokud se ale musíte rozhodnout sami, konzultujte to s ostatními vývojáři. Zeptejte se jich, s čím už mají zkušenosti, a pak si jeden systém zkrátka vyberte a používejte jej. Všechny stabilní systémy pro správu verzí, které jsou připravené na ostré nasazení, jsou dobré, takže se nemusíte obávat, že byste svým rozhodnutím něco zásadního pokazili. Pokud se ale nemůžete rozhodnout, zvolte Subversion. Dá se poměrně snadno naučit a je pravděpodobné, že přinejmenším po několik dalších let zůstane standardem.

---

[15] Důkazy jeho růstu naleznete na **http://cia.vc/stats/vcs** a **http://subversion.tigris.org/svn-dav-securityspace-survey.html.**

Používání systému pro správu verzí

Doporučení v této podkapitole nejsou zaměřená na konkrétní systém pro správu verzí a měla by se dát snadno realizovat v každém z nich. Podrobnosti si ověřte v dokumentaci svého konkrétního systému.

Sledujte verze u všeho

Správa verzí ve vašem projektu by se neměla omezovat jen na zdrojové kódy, ale i na jeho webové stránky, dokumentaci, FAQ, poznámky týkající se vývoje a všechno ostatní, co by někdo mohl chtít upravovat. To vše by se mělo nacházet někde v blízkosti zdrojového kódu, ve stromu stejného úložiště. Každá informace, kterou stojí za to zapsat, si zaslouží i sledování verzí – tedy přesněji řečeno, každá informace, která se může měnit. Věci, které se nemění, by se měly archivovat a ne zařazovat pod správu verzí. Například e-mailová zpráva se po odeslání už nemění, proto by její zařazení pod správu verzí nedávalo smysl (pokud se tedy nestane součástí nějakého většího, vyvíjejícího se dokumentu).

Důvod, proč by mělo být všechno umístěno na jediném místě a spravováno jediným systémem, je ten, že pak lidem stačí, aby se naučili jen jeden mechanismus pro provádění změn. Přispěvatelé například často začínají jen tím, že upravují webové stránky nebo dokumentaci, a až později začnou zasílat své drobné příspěvky do kódu. Pokud projekt používá pro všechny typy příspěvků stejný systém, pak se jej budou muset učit používat jen jednou. Spravování všech dokumentů stejným systémem také znamená, že nové funkce lze zapsat spolu s úpravami příslušné dokumentace v jednom commitu, že vytvoření nové větve programu současně povede k vytvoření větve dokumentace atd.

Pod správou verzí neudržujte *generované soubory*. Tyto soubory totiž nejsou editovatelná data v pravém slova smyslu, protože jsou vytvářeny příslušnými programy z jiných souborů. Některé systémy pro sestavování například vytvářejí soubor `configure` na základě šablony `configure.in`. Pokud něco chcete změnit v souboru `configure`, musíte upravit `configure.in` a pak znovu spustit generování; editovatelným souborem je v tomto případě pouze šablona configure.in. Z toho plyne, že pod správu verzí by měly spadat jen tyto šablony. Pokud byste pod ni zařadili i výsledné soubory, nevyhnutelně se stane, že někdo po změně šablony zapomene spustit generování a výsledkem bude naprostý zmatek, který budete rozmotávat ještě dlouho potom.[16]

Z pravidla o tom, že by veškerá editovatelná data měla mít své verze, existuje jedna celkem nešťastná výjimka – systém pro sledování chyb (bug tracker). Databáze chyb obsahují spoustu editovatelných dat, ale z technických důvodů tato data obvykle nemohou být uložena v hlavním systému pro správu verzí. (Některé systémy pro sledování chyb provádějí i vlastní, většinou celkem primitivní správu verzí; ta je ale nezávislá na hlavním úložišti projektu.)

---

[16] Jiný názor na správu verzí souborů `configure` naleznete v příspěvku Alexeje Machotkina nazvaném *„configure.in and version control“* („configure.in a systémy pro správu verzí“) a dostupném na **http://versioncontrolblog.com/2007/01/08/configurein-and-version-control/.**

Možnost prohlížení

Úložiště projektu by mělo být možné procházet na webu. To ale neznamená jen možnost prohlížení posledních revizí souborů projektu, ale také možnost jít zpět do minulosti a podívat se na předchozí revize, zobrazit rozdíly mezi jednotlivými revizemi, číst pro vybrané změny zprávy protokolu atd.

Možnost prohlížení je důležitá, protože představuje jednoduchý přístup k datům projektu. Pokud se úložiště nedá zobrazit prostřednictvím webového prohlížeče, pak musí někdo, kdo se chce podívat na konkrétní soubor (třeba proto, aby si ověřil, že se do kódu dostala oprava určité chyby), nejprve nainstalovat lokálního klienta systému pro správu verzí; z celkem jednoduchého úkolu, jenž by měl zabrat tak asi dvě minuty, se tak může stát práce na půl hodiny i víc.

Z možnosti prohlížení vyplývá i to, že budou existovat stabilní URL pro prohlížení konkrétních revizí souborů i URL pro zobrazení té nejaktuálnější revize. To se může hodit při technických diskuzích nebo při odkazování lidí na dokumentaci. Místo toho, abyste například řekli „pokyny pro správu chyb naleznete v souboru `community-guide/index.html` ve své pracovní kopii“, tedy můžete říct „pokyny pro správu chyb naleznete v **http://subversion.apache.org/docs/community-guide/**“, čímž dáte k dispozici URL, které vždy odkazuje na poslední revizi souboru `community-guides/index.html`. Takové URL je lepší, protože je zcela jednoznačné a vyhnete se otázkám, zda má dotyčný k dispozici aktualizovanou pracovní kopii.

Některé systémy pro správu verzí mají mechanismus pro prohlížení úložiště už zabudován, jiné se spoléhají na nástroje třetích stran. Takovými nástroji jsou například *ViewVC* (**http://viewvc.org/**), *CVSWeb* (**http://www.freebsd.org/projects/cvsweb.html**) a *WebSVN* (**http://websvn.tigris.org/**). První z nich dokáže pracovat jak s CVS, tak se Subversion, druhý jen s CVS a třetí jen se Subversion.

E-maily oznamující commit

Každé promítnutí změny (commit) do úložiště by mělo vygenerovat e-mail, který zachycuje, kdo změnu udělal, kdy ji udělal, které soubory a adresáře se změnily a jak. Tento e-mail by měl být zaslán do zvláštního mailing listu určeného právě pro tyto zprávy, jenž by měl být oddělen od mailing listů, do nichž přispívají lidé. Vývojářům a všem ostatním, kdo se o průběh projektu zajímají, byste měli doporučit, aby se do tohoto commitového mailing listu přihlásili, protože představuje ten nejefektivnější způsob, jak si udržet přehled o tom, co se v projektu děje na úrovni zdrojového kódu. Vedle zjevných technických výhod možnosti revidovat kód ostatních (viz **Kontrolujte kód veřejně**) pomáhají tyto e-maily vytvářet pocit soudržné komunity, protože vytvářejí sdílené prostředí, v němž mohou všichni reagovat na události (v tomto případě commity), o kterých vědí, že je vidí i ostatní.

To, jak přesně se zasílání e-mailů o provedených commitech nastavuje, je v různých systémech pro správu verzí různé, ale obvykle v nich najdete nějaký skript nebo jiný nástroj, který to dokáže. Pokud jej nemůžete najít, zkuste v dokumentaci hledat slovo *hooks* (háčky), popřípadě konkrétněji *post-commit hook*; v systému CVS se také používá termín *loginfo hook*. Obecně řečeno jsou háčky

prostředkem pro spouštění automatizovaných úloh na základě provedeného commitu. Každá operace commit spustí háček, předá mu veškeré informace, které se commitu týkají, a pak jej nechá, ať si s nimi dělá, co chce – například je pošle e-mailem.

U přednastavených systémů pro zasílání e-mailů o commitech možná budete chtít pozměnit jejich standardní chování:

1. U některých z nich e-maily o commitech neobsahují výpis rozdílů (diff), ale místo toho poskytují URL, přes které si změny lze prohlédnout v systému prohlížení úložiště na webu. Poskytnutí takového URL je sice dobrý nápad, protože jím pak můžeme na změnu odkázat, ale zároveň je velmi důležité, aby výpis rozdílů obsahoval i každý e-mail o commitu. Čtení e-mailů už patří ke každodenní rutině, takže pokud je obsah změny k vidění přímo v oznamovací zprávě, mohou vývojáři commit ihned revidovat, aniž by opustili program pro čtení pošty. Pokud k prohlédnutí změny musí kliknout na URL, většina z nich to neudělá, protože to od nich vyžaduje novou akci místo toho, aby jen pokračovali v tom, co už dělají. A kromě toho pokud se budou chtít na něco týkající se této změny zeptat, bude pro ně nesrovnatelně jednodušší kliknout na odpovědět a své poznámky připojit přímo k příslušným rozdílům, než navštívit příslušnou webovou stránku a pak pracně kopírovat části změn z webového prohlížeče do e-mailového klienta.

   (Pokud je ale diff opravdu rozsáhlý, jako například po přidání většího množství nového zdrojového kódu do úložiště, pak samozřejmě dává smysl rozdíly nevkládat a poskytnout pouze příslušné URL. Většina systémů pro posílání e-mailů o commitu umí takové omezení provést automaticky. Pokud to ten váš neumí, pak je přesto lepší změny přiložit a smířit se s tím, že čas od času přijde jeden obrovský e-mail, než změny nepřikládat vůbec. Pohodlné provádění revizí a připomínkování patří k základním kamenům vývoje ve skupině; jsou příliš důležité na to, aby se bez nich bylo možné obejít.)

2. E-maily o commitech by měly mít nastavenou hlavičku Reply-to na vývojářský mailing list a ne na mailing list commitů. To znamená, že pokud někdo příspěvek reviduje a napíše reakci, dostane se tato reakce automaticky do vývojářského mailing listu, kde se takové technické záležitosti obvykle probírají. Je k tomu několik důvodů. Za prvé chcete, aby všechny technické diskuze probíhaly v jednom mailing listu, protože tam je všichni očekávají a protože pak existuje jen jeden archiv, v kterém je potřeba hledat. Za druhé mohou existovat lidé, kteří se o projekt zajímají, ale do mailing listu pro commity přihlášeni nejsou. Za třetí mailing list pro commity slouží právě jen ke sledování commitů a ne ke sledování commitů a občasným technickým diskuzím. Ti, kteří se k mailing listu pro commity přihlásili, tím vyjádřili, že mají zájem získávat zprávy o commitech. Pokud bychom jim v tomto mailing listu posílali ještě něco jiného, tuto nepsanou dohodu bychom porušili. Za čtvrté si vývojáři často píšou vlastní jednoduché programy, které mailing list pro commity čtou a nějakým způsobem zpracovávají (například jej pak zobrazují na webové stránce). Takové programy jsou napsané tak, aby dokázaly číst zprávy o commitech naformátované konkrétním, očekávaným způsobem; na zprávy napsané lidmi připravené nejsou.

Tato rada změnit hlavičku Reply-to nijak neporušuje zásady, které jsem doporučoval v části **Velká debata o Reply-to** výše v této kapitole. Nastavení hlavičky Reply-to odesílatelem zprávy se vždy považuje za zcela korektní. V tomto případě je odesílatelem samotný systém pro správu verzí a hlavičku Reply-to nastavuje proto, že vhodným místem pro odpovídání je vývojářský mailing list a nikoliv mailing list pro commity.

CIA: Další mechanismus pro zveřejňování změn

E-maily o commitech nejsou jediným způsobem, kterým se mohou zprávy o změnách šířit. Nedávno byl vyvinut ještě další mechanismus nazvaný CIA (**http://cia.navi.cx/**). CIA je systém, který v reálném čase sbírá informace o commitech a šíří je dál. Nejoblíbenější použití CIA spočívá v rozesílání oznámení o commitech do IRC kanálů, takže ti, kdo jsou k těmto kanálům připojeni, mohou sledovat provádění commitů v reálném čase. Ačkoliv je technický význam tohoto nástroje poněkud menší než u e-mailů o commitech jednoduše proto, že když se toto oznámení na IRC objeví, nemusí kanál zrovna nikdo číst, má velký význam společenský. Vytváří pocit, že je každý vývojář součástí něčeho živého a aktivního; máte díky němu dojem, že vám projekt roste přímo před očima.

Funguje to tak, že program CIA pro zasílání oznámení vyvoláte pomocí háčku reagujícího na commit. Tento program pak naformátuje informaci o commitu do podoby XML zprávy, kterou odešle na centrální server (obvykle `cia.navi.cx`). Tento server pak informaci o commitu přepošle do ostatních fór.

Program CIA může být také nakonfigurován tak, aby zasílal tato data ve formátu **RSS**. Podrobnosti hledejte v dokumentaci na **http://cia.navi.cx/**.

Pokud chcete vidět, jak použití CIA vypadá v praxi, připojte svého IRC klienta na `irc.freenode.net`, kanál `#commits`.

Používejte větve, abyste nezpomalovali vývoj

Nezkušení uživatelé systémů pro správu verzí se někdy trochu bojí vytváření a slučování větví. To je pravděpodobně vedlejší efekt oblíbenosti systému CVS – jeho rozhraní pro vytváření větví a slučování totiž není příliš intuitivní, takže si mnozí lidé navykli se těmto operacím raději úplně vyhýbat.

Pokud k nim patříte také, dejte si závazek, že se celé věci přestanete obávat a naučíte se, jak se to dělá. Až si tyto operace osaháte, zjistíte, že vlastně zas tak složité nejsou, a čím více vývojářů se do projektu zapojí, tím je jejich použití důležitější.

Větve jsou cenné, protože se díky nim něco, čeho bylo dříve jen omezené množství, totiž místo ve zdrojovém kódu, v němž se dá bez obav pracovat, stává najednou hojně dostupným. Při klasickém vývoji pracují všichni společně a na stejném pískovišti si stavějí stejný hrad. Pokud někdo chce přidat nový padací most, ale nikdo jiný si nemyslí, že by to byl příliš dobrý nápad, tak se díky možnosti vytvořit větev může jednoduše přestěhovat někam stranou a vyzkoušet si to sám pro sebe. Když se mu to podaří,

může ostatní vývojáře pozvat, ať se na výsledek jeho práce podívají. Pokud se bude všem líbit, mohou systému pro správu verzí přikázat, aby padací most z tohoto experimentálního hradu přidal k tomu hlavnímu.

Je zřejmé, že něco takového společnému vývoji pomáhá. Lidé potřebují mít možnost vyzkoušet si nové věci, aniž by měli pocit, že přitom ostatním překážejí v práci. Stejně důležité jsou i situace, kdy je potřeba nějaký kód izolovat od hlavního vývoje, aby v něm bylo možné opravit nějakou chybu nebo stabilizovat nový release (viz **Stabilizace release** a **Udržování více release řad** v kapitole **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji**), aniž by se vám přitom kód nekontrolovaně měnil pod rukama.

Nebojte se vytvářet větve; používejte je často a nabádejte k tomu i ostatní. Zajistěte ale, aby byla každá větev aktivní jen tak dlouho, dokud je to potřeba. Každá aktivní větev trochu rozptyluje pozornost komunity. Dokonce i ti, kteří na větvi nepracují, pořád po očku sledují, co se v ní děje. To je samozřejmě zcela v pořádku; e-maily o commitech by měly být odesílány pro commity ve větvi stejně jako pro jakékoliv jiné. Systém větví by se ale neměl stát mechanismem pro štěpení vývojářské komunity. Až na vzácné výjimky by mělo být konečným cílem všech větví začlenění změn zpět do hlavní linie a jejich následný zánik.

Jedinečnost informace

Možnost slučování s sebou nese jedno důležité pravidlo: nikdy neposílejte stejný commit dvakrát. Každá jednotlivá změna by měla do systému pro správu verzí vstoupit právě jednou. Revize (nebo sada revizí), v níž tato změna vznikla, se od toho okamžiku stává jejím jedinečným identifikátorem. Pokud je potřeba změnu aplikovat i na jiné větve, než je ta, do níž byla vložena, pak by tam měla být přenesena ze svého původního umístění – jinými slovy není dobrý nápad provést další commit zcela identické změny. V kódu samotném by to sice vypadalo naprosto stejně, ale sledování změn a vydávání nových verzí by se tím nesmírně zkomplikovalo.

Způsoby, jak tuto radu aplikovat v praxi, se liší od jednoho systému pro správu verzí k druhému. V některých systémech se operace typu merge považují za zvláštní události, které se principiálně liší od commitů a které nesou svá vlastní metadata. V jiných systémech se výsledky operací merge zaznamenávají stejným způsobem jako commity, takže „merge commit“ a „commit nových změn“ se od sebe odlišují především zprávou v protokolu. Pokud provádíte merge, neopakujte v protokolu zprávu, která doprovázela původní změnu. Místo toho napište, že jde o merge, a uveďte identifikaci revize původní změny; můžete přidat i popis toho, co změna dělá, ale nejvýš jednou větou. Pokud někdo bude chtít vidět plnou zprávu z protokolu, měl by ji hledat u původní revize.

Důvod, proč je důležité vyhnout se opakování, je ten, že se zprávy v protokolech občas upravují i poté, co byly zaznamenány. Kdyby se taková zpráva objevovala na každém místě, kde došlo k merge dané změny, pak by v případě, že někdo její první výskyt upraví, zůstala všechna další opakování v původní podobě, což by vedlo akorát ke zmatení.

Stejný princip platí pro návrat ke stavu před změnou. Pokud je změna z kódu odstraněna, pak má příslušná zpráva v protokolu pouze oznámit, že došlo k návratu k určité revizi, a ne popsat, jak přesně byl kód změněn, aby se k tomuto předchozímu stavu vrátil – to lze kdykoliv odvodit z textu původní zprávy v protokolu a z původní změny samotné. Zpráva oznamující návrat k dřívějšímu stavu by samozřejmě měla uvádět i důvod, proč je daná změna nežádoucí, ale neměla by duplikovat nic z původní zprávy o změně. Pokud je to možné, k této původní zprávě se vraťte a připište, že změna byla odvolána.

Ze všeho, co jsme uvedli výše, vyplývá, že byste měli mít ustálený systém pro odkazování na konkrétní revize. Je to užitečné nejen v protokolech, ale také v e-mailech, v bug trackeru i jinde. Pokud používáte CVS, doporučuji „`cesta/k/souboru/ve/stromu/projektu:REV`“, kde REV je číslo revize v CVS, tedy například „1.76“. Pokud používáte Subversion, pak se pro revizi 1729 používá standardní zápis „r1729“ (cesty k souborům nepotřebujeme, protože Subversion používá pro revize globální čísla). V ostatních systémech obvykle existuje standardní zápis pro identifikaci konkrétní sady změn. Ať už je příslušný zápis pro váš systém jakýkoliv, nabádejte ostatní, aby jej při odkazování na změny používali. Při systematickém používání takových označení jsou organizační práce celého projektu mnohem jednodušší (jak uvidíme v kapitole **6. Komunikace** a v kapitole **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji**); vzhledem k tomu, že značnou část této práce budou provádět dobrovolníci, měla by být tak snadná, jak je to jen možné.

Viz také **Vydávání releasů a každodenní práce** v kapitole **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji.**

Autorizace

Většina systémů pro správu verzí nabízí funkci, díky níž lze některým uživatelům umožnit nebo naopak zakázat promítání změn do konkrétních částí úložiště. Na základě principu, že když dáte lidem do ruky kladivo, začnou hledat hřebíky, využívají tuto funkci mnohé projekty velmi náruživě; pečlivě přidělují lidem přístup jen do těch oblastí, kam smějí zapisovat změny, a zajišťují, že nikam jinam přispívat nemůžou. (Rady, jak v projektu rozhodnout, kdo a kam může přispívat, naleznete v části **Committeři** v kapitole **8. Řízení dobrovolníků.**)

Takhle pečlivým dohledem asi nic nezkazíte, ale vůbec nevadí, pokud se celou věcí zas tolik zabývat nebudete. Některé projekty spoléhají na to, že jejich vývojáři budou držet slovo, a pokud někomu přidělí právo na zapsání změn, i když to bude jen pro nějakou dílčí část celého úložiště, pošlou mu heslo, které umožňuje přispívat kamkoliv. Pouze jej požádají, aby své příspěvky omezil jen na danou oblast. Uvědomte si, že zde vlastně žádné riziko nehrozí – v aktivním projektu se stejně všechny nové commity revidují. Pokud někdo zašle commit někam, kam by neměl, ostatní si toho všimnou a něco k tomu řeknou. Pokud je nutné tento commit odstranit, nic se neděje – vše je zařazeno pod správu verzí, takže se jednoduše provede návrat ke stavu před změnou.

Takový uvolněnější přístup má několik výhod. Za prvé když pak tito vývojáři expandují i do jiných oblastí, což se, pokud u projektu zůstanou, děje celkem běžně, nemusíte řešit to, že jim je potřeba přidělit širší oprávnění. Jakmile je učiněno příslušné rozhodnutí, můžou hned začít posílat své commity do nové oblasti.

Za druhé lze takové rozšiřování působnosti provádět jemněji. Obecně to probíhá tak, že když někdo, kdo přispívá do oblasti X, chce rozšířit svou působnost i do oblasti Y, začne posílat své záplaty k oblasti Y s žádostí o revizi. Pokud si někdo, kdo má právo zapisovat do oblasti Y, takovou záplatu prohlédne a schválí ji, může jejímu autorovi říct, aby ji poslal přímo jako commit (přičemž by samozřejmě měl zmínit ve zprávě do protokolu jméno toho, kdo mu to povolil). Díky tomu přijde commit od toho, kdo příslušnou změnu skutečně napsal, což je lepší jak z pohledu správy informací, tak z hlediska ocenění zásluh.

Poslední a možná nejdůležitější výhoda používání tohoto systému je to, že posiluje atmosféru důvěry a vzájemného respektu. Když někomu dáte možnost posílat commity k nějaké konkrétní oblasti, říkáte tím něco o jeho technických schopnostech: „Vidíme, že jsi natolik schopný, abys mohl do téhle oblasti přispívat, takže s chutí do toho.“ Pokud ale budete přísně hlídat, kam dotyčný ještě může a kam už ne, vyjadřujete tím něco trochu jiného: „Nejen že pochybujeme, že bys mohl přijít s něčím užitečným ještě v nějaké jiné oblasti, ale také si nejsme úplně jisti, že nemáš nekalé úmysly.“ To není zrovna něco, co byste chtěli ostatním říkat, pokud nemusíte. Když někoho k projektu přiberete jako přispěvatele, je to dobrá příležitost uvést jej do kruhu lidí, kteří si navzájem důvěřují. To se dá udělat například právě tak, že mu svěříte větší pravomoc, než jakou bude potřebovat, a řeknete mu, že je jen na něm, aby stanovená omezení dodržel.

Projekt Subversion v době psaní tohoto textu už používal takový systém více než čtyři roky; v tento moment je v něm 33 přispěvatelů s plným přístupem a 43 s částečným. Systém, který tato práva řídí, ale rozlišuje jenom mezi těmi, kteří přístup pro commit mají, a těmi, kteří ne. Veškeré další dělení dělají jen a jen lidé. Nikdy jsme ale neměli problém s tím, že by někdo záměrně zasílal commity mimo svou oblast. Jednou nebo dvakrát se stalo, že někdo (zcela bez vedlejších úmyslů) špatně pochopil, kam zasahovat může a kam ne, ale vždy se to vyřešilo rychle a přátelsky.

Je jasné, že v situacích, kdy by bylo spoléhání se na sebekázeň ostatních nepraktické, musíte spoléhat na přísné přidělování přístupových práv. Ale takové situace jsou vzácné. I kdyby váš projekt měl miliony řádků kódu a stovky nebo i tisíce vývojářů, stále by měl být každý commit revidován těmi, kdo na daném modulu pracují, a ti měli být schopni poznat, že změny zaslal někdo, kdo by neměl. A pokud váš projekt nově zapsané commity nereviduje, pak má mnohem větší problémy, než je hlídání něčích oprávnění.

Celkově se tedy dá říct, že je zbytečné ztrácet čas tím, že si budete s nastavováním práv v systému pro správu verzí příliš hrát, pokud k tomu tedy nemáte konkrétní důvod. Výhod, které to přináší, není zas tolik a spoléhání se na členy projektu samotné je většinou lepší.

Tím ale rozhodně neříkám, že omezení jako taková nejsou důležitá. Pro projekt by rozhodně nebylo dobré vyzývat jeho účastníky, aby zasílali commity do oblastí, pro něž nemají kvalifikaci. V mnoha projektech má navíc plný (neomezený) přístup do úložiště ještě zvláštní význam: vyplývá z něj právo hlasovat o otázkách týkajících se projektu jako takového. Tento v zásadě politický aspekt věci podrobněji probereme v části **Kdo hlasuje?** v kapitole **4. Společenská a politická infrastruktura.**

Systém pro sledování chyb

Téma sledování chyb je velmi široké a budeme se s ním průběžně setkávat v celé této knize. V této části se pokusím soustředit hlavně na to, jak takový systém nastavit, a na další technické otázky, ale než se k nim dostaneme, musíme začít u toho nejzákladnějšího – jaký druh informací bychom v něm vlastně měli sledovat?

Samotný pojem *systém pro sledování chyb* (bug tracker) je poněkud zavádějící. Tyto systémy se často používají i pro další věci, jako je správa požadavků na nové funkce, jednorázových úkolů, nevyžádaných záplat a vlastně čehokoliv jiného, u čeho lze odlišit nějaký počáteční a koncový stav (a případně stavy mezi nimi) a u čeho v průběhu existence dochází k nabalování různých informací. Z tohoto důvodu mohou mít systémy pro sledování chyb ještě mnoho dalších jmen – *systémy pro sledování problémů* (issue trackers), *systémy pro sledování závad* (defect trackers), *systémy pro sledování artefaktů* (artifact trackers), *systémy pro sledování požadavků* (request trackers), *systémy pro řešení potíží* (trouble ticket systems) atd. **Seznam softwaru** naleznete v příloze **B. Volně přístupné bug trackery (záznamníky chyb)**.

V této knize budu pro takový software používat pojem „systém pro sledování chyb“, respektive „bug tracker“, protože tak jej nazývá i většina lidí; pro konkrétní položku v databázi takového systému budu používat pojem *záznam o problému* (issue). Díky tomu od sebe bude možné odlišit správné či nesprávné chování, se kterým se uživatel setkal (tedy chyby jako takové), a záznamy bug trackeru, které zachycují, kdy a jak byla chyba objevena, kde byla nalezena její příčina a jak byla nakonec vyřešena. Nezapomínejte, že ačkoliv se většina takových záznamů týká skutečných chyb, lze je využít i pro sledování jiných typů úkolů.

Klasický životní cyklus záznamu o problému vypadá asi takto:

1. Někdo záznam o problému vytvoří. Napíše shrnutí, stručný popis (pokud možno včetně návodu pro to, jak chybu vyvolat – o tom, jak zajistit kvalitní hlášení o chybách, najdete více v části **Ke všem uživatelům se chovejte jako k potenciálním dobrovolníkům** v kapitole **8. Řízení dobrovolníků**) a jakékoliv další informace, které systém vyžaduje. Osoba, která záznam vytvořila, může být v rámci projektu zcela neznámá – zprávy o chybách a žádosti o nové funkce přicházejí zrovna tak často od uživatelů jako od vývojářů.

Jakmile je záznam o problému vytvořen, dostává se do takzvaného *otevřeného stavu*. Protože zatím nebyla provedena žádná další činnost, přidělují mu některé systémy navíc nálepku *neověřeno* (unverified), případně *nezahájeno* (unstarted). Problém zatím nebyl nikomu přidělen, respektive v některých systémech byl přidělen fiktivnímu uživateli, což v praxi znamená totéž: že za něj dosud nikdo není zodpovědný. Záznam se prozatím nachází v jistém provizoriu – problém byl sice ohlášen, ale projekt si jej ještě pořádně nevšiml.

2. Další lidé si záznam o problému přečtou, přidají k němu své komentáře a možná jeho autora požádají o vyjasnění některých bodů.

3. Chyba je *reprodukována*. To může být ten nejdůležitější okamžik v celém jejím životním cyklu. Chyba sice pořád ještě není opravena, ale to, že ji byl schopen vyvolat i někdo jiný než autor záznamu, dokazuje, že skutečně existuje; neméně důležité je i to, že autorovi potvrzuje, že svým oznámením projektu pomohl.

4. Chyba je *diagnostikována* – je nalezena její příčina, a pokud je to možné, je proveden odhad toho, jak pracné bude ji opravit. Nezapomeňte, že tyto věci by se měly v záznamu objevit. Pokud by se náhodou člověk, který diagnostiku chyby prováděl, musel najednou od projektu na nějakou dobu vzdálit (což se u dobrovolných vývojářů stává poměrně často), měl by být někdo jiný schopen pokračovat tam, kde práce přestala.

   V této fázi, popřípadě v té předchozí, se může některý vývojář rozhodnout „převzít vlastnictví“ celého problému a v systému si jej *přidělit* (podrobnosti o tom, jak přidělování funguje, naleznete v části **Jasně rozlišujte mezi poptáváním a zadáváním** v kapitole **8. Řízení dobrovolníků**). V této etapě může být u problému také nastavena jeho *priorita*. Pokud je například natolik závažný, že by měl vést k odkladu vydání další verze, měla by tato skutečnost být zjištěna co nejdřív a systém pro sledování by to měl být schopen nějakým způsobem zaznamenat.

5. Je naplánováno řešení problému. To nemusí nutně znamenat, že se určí konkrétní datum, ke kterému bude chyba opravena. Někdy se tím myslí jen rozhodnutí, ve které příští verzi (nemusí to být hned ta následující) by chyba měla být opravena, popřípadě to, že by tato chyba neměla vydání nových verzí blokovat. Pokud lze chybu opravit rychle, můžete se klidně obejít i bez plánování.

6. Chyba je opravena (nebo je dokončen úkol, aplikována záplata nebo něco podobného). K problému by měl být přidán komentář, který odkáže na změnu, popřípadě sadu změn, která chybu opravuje. Celý problém je *uzavřen*, případně označen jako *vyřešený*.

Uvedený životní cyklus může mít i různé varianty, z nichž některé jsou poměrně běžné. Někdy je záznam o problému uzavřen už záhy po svém vzniku, protože se ukáže, že vůbec nejde o chybu, ale spíše o neporozumění na straně uživatele. S tím, jak projekt získává více uživatelů, přichází podobných

záznamů o neexistujících problémech stále víc a víc a vývojáři, kteří je musí uzavírat, k nim přidávají stále jedovatější komentáře. Tomu se ale pokuste zabránit. Nikomu to neprospěje, protože žádný konkrétní uživatel za ty předchozí nesprávné záznamy nemůže; to, že už jich bylo hodně, vědí sice vývojáři, ale uživatelé ne. (V části **Předběžné filtrování v bug trackeru** dále v této kapitole se podíváme na techniky, kterými lze počet neplatných záznamů o problémech redukovat.) Pokud různí uživatelé opakovaně dospívají ke stejnému nedorozumění, může to také znamenat, že by tato část softwaru potřebovala poněkud upravit. Něco takového nejsnadněji zjistíte, pokud budete mít určeného správce problémů (issue manager), který bude databázi chyb systematicky sledovat; viz **Issue manager (manažer problémů)** v kapitole **8. Řízení dobrovolníků.**

Další běžnou variantou je záhy po provedení prvního kroku celý záznam uzavřít, protože se ukáže, že jde o *duplikát*. Za duplikát se považuje to, když někdo vytvoří záznam o problému, který už je v projektu známý. Duplikáty se neobjevují jen u otevřených záznamů – může se také stát, že se nějaká chyba znovu objeví poté, co už byla opravena (říká se tomu *regrese*). V takovém případě se dává obvykle přednost znovuotevření původního záznamu a uzavření nových oznámení jako duplikátů. Systém pro sledování chyb by měl vztah mezi duplikáty udržovat v obou směrech, takže například informace o tom, jak chybu vyvolat, která byla přidána k jednomu z duplikátů, by měla být dostupná i z původního záznamu a naopak.

Třetí varianta nastává, když vývojáři záznam o problému uzavřou, protože si myslí, že chybu opravili, ale ten, kdo jej ohlásil, opravu zamítne a problém znovu otevře. Stává se to obvykle z toho důvodu, že vývojáři nemají přístup k prostředí, ve kterém by bylo možné chybu reprodukovat, nebo proto, že opravu neotestovali přesně podle postupu pro vyvolání chyby, který autor záznamu uvedl.

Kromě uvedených možností se můžeme setkat i s dalšími menšími odlišnostmi životního cyklu, které se mohou lišit v závislosti na použitém softwaru, v zásadě ale vypadá vždy zhruba podobně. Ačkoliv tento životní cyklus jako takový není specifický jen pro open source software, má dopad na to, jak se v open source projektech systémy pro sledování chyb používají.

Jak vyplývá z kroku 1, systém pro sledování chyb spoluutváří to, jak projekt působí navenek, stejně jako mailing listy a webové stránky. Záznam o problému může vytvořit kdokoliv; kdokoliv se na něj může také podívat a kdokoliv si může nechat vypsat seznam všech aktuálně otevřených problémů. Z toho vyplývá, že nikdy nevíte, kolik lidí čeká na to, až se u daného záznamu objeví nějaký pokrok. Je samozřejmé, že tempo, jakým mohou být problémy řešeny, je ovlivněno velikostí a schopnostmi vašeho vývojářského týmu, ale projekt by se měl přinejmenším pokusit co možná nejrychleji potvrdit, že vytvoření nového záznamu vzal na vědomí. I pokud se celý problém trochu povleče, může taková zpráva autora záznamu povzbudit, protože jej ujišťuje, že se na něj nezapomnělo a že si jeho práce někdo všiml (nezapomínejte, že vyplnění záznamu o problému obvykle vyžaduje větší úsilí než řekněme odeslání e-mailu). Navíc jakmile si problému všimne nějaký vývojář, dostane se tím do povědomí celého projektu – čímž myslím to, že tento vývojář může od toho momentu začít sledovat, jestli se neděje ještě něco jiného, co s tímto problémem souvisí, může o něm hovořit s ostatními atd.

Potřeba včasné reakce vede k dvěma věcem:

- Systém pro sledování musí být napojený na mailing list, aby každá změna záznamu o problému, včetně jeho vytvoření, způsobila odeslání e-mailu popisujícího, co se stalo. Tento e-mail obvykle nebude posílán do hlavního vývojářského mailing listu, protože ne všichni vývojáři chtějí automatické zprávy o chybách dostávat, ale jeho hlavička Reply-to by měla být (stejně jako u e-mailů o commitech) nastavena právě tam.
- Formulář pro zadávání záznamu o problému by měl obsahovat i pole pro e-mailovou adresu ohlašujícího, aby mohl být kontaktován v případě, že budou potřeba ještě další informace. (Vyplnění této e-mailové adresy by ale nemělo být povinné, protože někteří lidé dávají při ohlašování problémů přednost anonymitě. Více o tom, jak je anonymita důležitá, si povíme v části **Anonymita a zapojení se do projektu** dále v této kapitole.)

Interakce s mailing listy

Zajistěte, aby se systém pro sledování chyb nezvrhl v diskuzní fórum. Je sice důležité, aby lidé bug tracker sledovali a přispívali do něj, ale už z principu není tento systém vhodný pro diskuzi v reálném čase. Berte jej spíše jako archivační nástroj a způsob, jak shromažďovat informace a odkazy na diskuze, které probíhají jinde, zejména v mailing listech.

K tomu existují dva důvody: Za prvé jsou systémy pro sledování chyb na rozdíl od mailing listů (a chatovacích místností) pro tento účel dost neohrabané. Není to proto, že by měly špatné uživatelské rozhraní, ale proto, že toto rozhraní bylo navrženo pro zachycení a popis jednotlivých ohraničených stavů a ne pro volně plynoucí diskuze. Za druhé ne každý, kdo by měl být do diskuze kolem daného problému zapojen, bug tracker sleduje. Dobrá správa problémů (viz **Podělte se o řídící i technické úkoly** v kapitole **8. Řízení dobrovolníků**) mimo jiné znamená zajištění, aby byli na každý problém upozorněni ti správní lidé, spíš než aby museli všichni sledovat všechno. V části **Nediskutujte v bug trackeru** v kapitole **6. Komunikace** se podíváme na způsoby, jak zajistit, aby se diskuze z příslušného fóra nepřelévala do bug trackeru.

Některé systémy pro sledování chyb umí monitorovat mailing listy a automaticky si vést záznamy o všech e-mailech, které se nějakého známého problému týkají. Obvykle to dělají tak, že hledají identifikační číslo příslušného záznamu, přidávané do předmětu e-mailu ve zvláštním řetězci. Vývojáři se časem naučí tyto řetězce do svých e-mailů přidávat, aby si jich bug tracker všiml. Systém pro sledování chyb může buď celý e-mail uložit, nebo (což je ještě lepší) může do svých záznamů přidat odkaz na tento e-mail v archivu mailing listu. V obou případech je to velmi užitečná funkce; pokud ji váš bug tracker podporuje, nezapomeňte ji zapnout a ostatním připomínat, že by ji měli využívat.

Předběžné filtrování v bug trackeru

Většina databází se záznamy o problémech časem začne trpět stejným neduhem, jímž je záplava duplikátů nebo záznamů o neexistujících problémech vytvořených uživateli, kteří to sice mysleli dobře, ale neměli dost zkušeností nebo informací. Pokud se tomuto problému chcete postavit, první krok obvykle spočívá v tom, že někam na hlavní stránku bug trackeru umístíte výrazné upozornění, v němž vysvětlíte, jak se pozná, že je nějaká chyba opravdu chybou, jak hledáním zjistit, zda už náhodou nebyla zaznamenána, a nakonec jak by měl uživatel, který je i nadále přesvědčen, že našel novou chybu, tuto věc oznámit.

Na nějakou dobu se tím příval špatných záznamů trochu sníží, ale s nárůstem uživatelů se problém dřív nebo později vynoří znovu. Nemůžete to ale mít žádnému konkrétnímu uživateli za zlé. Každý z nich se upřímně snaží přispět k dobrému zdraví projektu; dokonce i když bude něčí první oznámení chyby zbytečné, měli byste ho přesto povzbudit k tomu, aby zůstal do projektu zapojen a v budoucnu zaznamenal opravdové problémy. To ale nic nemění na tom, že váš projekt potřebuje udržet svou databázi záznamů o problémech v co nejlepším stavu.

Předcházení celému problému nejvíce pomohou dvě věci: první z nich je zajištění, že systém pro sledování chyb prohlížejí lidé, kteří mají dostatečné znalosti na to, aby mohli duplicitní a neplatné záznamy uzavřít hned, jak se objeví; druhou pak je od uživatelů vyžadovat (nebo je důrazně nabádat), aby si domnělou chybu nechali před zapsáním do systému potvrdit ještě někým jiným.

První z těchto technik se používá celkem často. Dokonce i v projektech s obrovskými databázemi problémů (jako je například bug tracker projektu Debian na **http://bugs.debian.org/**, který v době psaní tohoto textu obsahoval 315 929 záznamů) lze věci uspořádat tak, aby se na každý příchozí problém někdo skutečně podíval. Může to třeba být pro každou kategorii problému někdo jiný. Zmíněný projekt Debian je například kolekcí softwarových balíčků, takže je každý problém automaticky přesměrován na správce příslušného balíčku. Uživatelé samozřejmě někdy mohou určit kategorii problému chybně, takže se jejich záznam dostane k nesprávné osobě a ta jej pak musí přeposlat jinam. To ale pořád znamená, že se celá zátěž trochu rozloží. Ať už se uživatel při vytváření záznamu trefí nebo ne, prohlížení nových záznamů je mezi vývojáři rozděleno víceméně rovnoměrně, takže ke každému problému může přijít včasná reakce.

Druhá technika už tolik rozšířená není, pravděpodobně proto, že se obtížněji automatizuje. Základní myšlenka spočívá v tom, že se každý nový problém dostává do databáze přes někoho dalšího. Pokud si uživatel myslí, že našel nějaký problém, bude požádán o jeho popsání v jednom z mailing listů nebo prostřednictvím IRC kanálu, kde mu někdo potvrdí, že jde skutečně o chybu. Když je k celému problému už na začátku přidán další pár očí navíc, předejde se velkému množství falešných hlášení. Druhá strana někdy správně pozná, že dané chování není chybou, nebo že je to něco, co bylo v posledním vydání opraveno. Může si také vzpomenout, že se stejné symptomy objevily už dříve, a předejít duplicitě tím, že uživatele odkáže na starší záznam. Často také stačí, když se někdo zeptá „Zkoušeli jste

v bug trackeru hledat, zda tato chyba už nebyla nahlášena?“ Mnoho lidí hledání v trackeru samo od sebe vůbec nenapadne, ale pokud zjistí, že se to od nich očekává, rádi to udělají.

Takový systém opravdu dokáže udržet celou databázi v poměrně čistém stavu, ale má i své nevýhody. Mnoho lidí záznam o problému stejně vytvoří bez jakéhokoliv ptaní, protože si buď instrukcí, které uživatele vyzývají, aby si svůj problém nejprve ověřili u někoho jiného, nevšimli, nebo je ignorovali. Takže stejně potřebujete dobrovolníky, kteří budou databázi sledovat. Většina nových oznamovatelů navíc vůbec netuší, jak obtížné spravování databáze se záznamy vlastně je, takže není úplně fér, když je budete kvůli ignorování pokynů plísnit. Dobrovolníci musí být sice pečliví, ale na druhou stranu by měli být při vracení neověřených problémů zpátky k oznamovatelům spíše opatrní. Cílem je, aby se každý z nich do budoucna naučil tento ověřovací systém používat, protože se tak zvýší počet lidí, kteří si jsou celého problému filtrování záznamů v trackeru vědomi. Pokud zpozorujete nové hlášení, které bylo zapsáno bez ověření, pak by měly být v ideálním případě provedeny následující kroky:

1. Okamžitě na záznam zareagujte. Uživateli zdvořile poděkujte za jeho vyplnění, ale odkažte jej na pokyny pro ověřování (které by samozřejmě měly být popsány na nějakém nápadném místě vašich stránek).
2. Pokud je ovšem jeho záznam oprávněný a nejde o duplikát, schvalte jej a uveďte do standardního životního cyklu. O tom, že by měl být problém nejdřív ověřen, už jste autora informovali, a bylo by zcela zbytečné mrhat jeho časem tím, že byste jinak platný záznam zavírali.
3. V opačném případě, tedy pokud není jasné, zda jde o skutečný problém, záznam uzavřete, ale oznamujícího požádejte, aby jej znovu otevřel, pokud najde někoho, kdo mu jej potvrdí. Pokud se tak stane, měl by uvést odkaz na diskuzní vlákno, ve kterém k tomuto potvrzení došlo (například URL archivu mailing listu).

Nezapomínejte, že ačkoliv tento systém může v databázi bug trackeru časem poměr signál/šum výrazně zlepšit, nežádoucích záznamů se nikdy úplně nezbavíte. Jediný způsob, jak zcela zabránit vzniku falešných záznamů, je zakázat přístup do bug trackeru všem kromě vývojářů; taková léčba ale není o nic lepší než onemocnění samotné, dokonce spíš naopak. Je lepší se smířit s tím, že vymycování falešných záznamů patří k rutinní údržbě projektu, a snažit se pro tuto činnost získat co nejvíc pomocníků.
Viz také **Issue manager (manažer problémů)** v kapitole **8. Řízení dobrovolníků**.

IRC a jiné systémy pro diskuzi v reálném čase

Mnohé projekty spravují místnosti pro diskuzi v reálném čase, které využívají protokol *Internet Relay Chat* (IRC). Na těchto fórech si mohou uživatelé a vývojáři navzájem klást otázky a získávat na ně okamžitou odpověď. Je sice možné provozovat IRC server v rámci vašeho vlastního webu, ale není to úplně jednoduché a za ty potíže to nestojí. Místo toho to řešte tak jako všichni ostatní: provozujte své IRC kanály na Freenode (**http://freenode.net/**). Freenode poskytuje administrátorské nástroje, které pro správu IRC kanálů svého projektu potřebujete,[17] aniž byste museli provozovat vlastní IRC server.

První věcí, kterou musíte udělat, je zvolit si jméno kanálu. Nejvíc se nabízí jméno vašeho projektu; pokud je na Freenode ještě volné, použijte jej. Pokud ne, pak vyberte něco, co se bude jménu vašeho projektu podobat a bude dobře zapamatovatelné. Existenci kanálu oznamte na stránkách projektu, aby si této informace všiml každý návštěvník, který se jen chce na něco rychle zeptat. Například na hlavní stránce projektu Subversion se nachází úplně nahoře celkem nápadný rámeček, v němž stojí toto:

> *Pokud používáte Subversion, doporučujeme, abyste se přihlásili k mailing listu* `users@subversion.tigris.org` *a přečetli si* ***manuál Subversion*** *a* ***FAQ****. Dotazy můžete také pokládat na našem IRC kanálu* `#svn` *na* `irc.freenode.net`.

Některé projekty mají pro různá témata více různých kanálů. Například tedy mají kanál pro potíže při instalaci, další pro otázky uživatelů, další pro diskuzi vývojářů atd. (více informací o tom, jak rozdělit diskuzi do více kanálů, najdete v části **Jak se vyrovnat s růstem** v kapitole **6. Komunikace**). Ze začátku by měl projekt mít jen jeden kanál, kde se probírá všechno dohromady. Časem, až vzroste poměr uživatelů k vývojářům, se může ukázat, že potřebujete více různých kanálů.

Jak se lidé dozví, jaké kanály vlastně existují a ke kterému by se měli připojit? A až se připojí, jak zjistí, jaké zvyklosti v daném kanálu platí?

Odpověď je jednoduchá – napište tyto informace do *tématu kanálu*.[18] Téma kanálu je krátká zpráva, která se uživateli vypíše pokaždé, když do něj vstoupí. Nováčkům poskytuje pár stručných rad a slouží i jako rozcestník na místa, kde lze získat další informace. Například takto:

```
Nacházíte se v #svn

Téma #svn je Fórum pro dotazy uživatelů Subversion, viz také
http://subversion.tigris.org/. || Vývojářská diskuze probíhá v
#svn-dev. || Prosím, nekopírujte sem dlouhé kusy textu; místo toho
použijte např. http://pastebin.ca/ || NOVINKY: vydána verze
Subversion 1.1.0, podrobnosti naleznete na http://svn110.notlong.com/
```

To je sice dost strohé, ale obsahuje to všechno, co potřebují nově příchozí vědět. Dozví se, k čemu kanál přesně slouží, kde se nacházejí hlavní stránky projektu (pokud se někdo dostal na IRC, aniž by na těchto stránkách byl) a že existuje ještě sesterský kanál; jsou tu i informace o tom, jak citovat delší text.

---

[17] Pro využívání služeb Freenode není nutné ani se automaticky neočekává, že jim pošlete nějaký finanční příspěvek, ale pokud si to můžete dovolit, pouvažujte o tom, prosím. Společnost Freenode je registrována v USA jako dobročinná organizace, která je osvobozena od daně, a její služby jsou velmi cenné.

[18] Téma kanálu se nastavuje příkazem `/topic`. Všechny příkazy v IRC začínají „/“. Pokud nevíte, jak se používá a spravuje kanál IRC, podívejte se na **http://www.irchelp.org/**; zejména stránka **http://www.irchelp.org/irchelp/irctutorial.html** vám může hodně pomoct.

Jak na delší citáty

IRC kanál je sdílený prostor – každý zde vidí všechno, co ostatní napíšou. To je samozřejmě dobrá věc, protože to znamená, že se k probíhající konverzaci může přidat každý, kdo má pocit, že k tomu má co říct, a zároveň ji mohou sledovat i jiní a dozvídat se něco, co nevěděli. Problém ale nastává, když někdo potřebuje zveřejnit větší množství informací najednou, jako například výpis z debugovacího prostředí, protože pokud do kanálu vložíte tak velké množství textu, přerušíte tím veškerou ostatní konverzaci.

Řešením je využít služeb serverů zvaných *pastebin* nebo *pastebot*. Když po někom budete chtít velký objem dat, požádejte jej, ať je nevkládá přímo do IRC, ale navštíví například **http://pastebin.ca/**, data vloží tam a pošle jen výsledné URL. Na něj se pak může kdokoliv podívat a data si prohlédnout.

Takových služeb dostupných zdarma je na internetu celá řada, takže se nebudu pokoušet sestavit úplný seznam, ale mezi ty nejlepší, co znám, patří tyto: **http://www.nomorepasting.com/**, **http://pastebin.ca/, http://nopaste.php.cd/, http://rafb.net/paste/, http://sourcepost.sytes.net/, http://extraball.sunsite.dk/notepad.php** a **http://www.pastebin.com/.**

Roboti

Mnoho technicky orientovaných IRC kanálů má i jednoho člena, který ale ve skutečnosti není člověk, nýbrž robot (popřípadě zkráceně bot), který umí ukládat informace a pak je na základě nějakého příkazu zopakovat. Obvykle se na robota obracíte stejně jako na kohokoliv jiného v kanálu, takže mu příkazy dáváte tak, že s ním mluvíte. Například takto:

```
<kfogel> ayita: learn diff-cmd =
http://subversion.tigris.org/faq.html#diff-cmd
<ayita> Thanks!
```

Tím jsem řekl robotovi (který se v kanálu jmenuje ayita), aby si zapamatoval, že odpověď na dotaz „diff-cmd“ je zmíněná URL. Díky tomu pak můžeme ayitu poprosit, aby tuto informaci předal jinému uživateli:

```
<kfogel> ayita: tell jnovak about diff-cmd
<ayita> jnovak: http://subversion.tigris.org/faq.html#diff-cmd
```

Stejný příkaz má i kratší verzi:

```
<kfogel> !a jnovak diff-cmd
<ayita> jnovak: http://subversion.tigris.org/faq.html#diff-cmd
```

To, jak přesně takové příkazy vypadají a co dělají, závisí na konkrétním robotovi. Uvedené příklady se týkají robota jménem `ayita` (**http://hix.nu/svn-public/alexis/trunk/**), který obvykle běží v `#svn` na freenode. Další populární roboti jsou třeba Dancer (**http://dancer.sourceforge.net/**) a Supybot (**http://supybot.com/**). Na to, abyste mohli robota spustit, nepotřebujete žádné zvláštní oprávnění. Robot je klientský program; nastavit jej a pustit na nějaký server a kanál může kdokoliv.

Pokud ve svém kanálu musíte často odpovídat na pořád stejné otázky, silně doporučuji využít služeb nějakého robota. Jenom poměrně malé procento uživatelů IRC se naučí s ním zacházet, ale tito uživatelé pak budou schopni rychle odpovědět na mnoho otázek, protože použití robota je výrazně efektivnější, než muset psát pořád totéž.

Archivace IRC

Ačkoliv je možné všechno, co se v IRC kanálu stane, archivovat, není to běžné. IRC konverzace jsou sice technicky vzato veřejné, ale mnoho lidí je považuje za neformální, tak trochu soukromé rozhovory. Uživatelé obvykle příliš nedbají na správnou gramatiku a často se podělí i o své názory, například týkající se jiného softwaru či ostatních programátorů, které by archivovat určitě nechtěli.

Samozřejmě někdy proběhnou konverzace, jejichž útržky bude stát za to zachovat, což je pochopitelně zcela v pořádku. Většina IRC klientů dokáže na požádání uložit záznam konverzace do souboru, ale pokud to neumí, vždy máte ještě možnost text z IRC ručně zkopírovat a vložit na nějaké trvalejší fórum, často například bug tracker. Pokud byste ale archivovali úplně všechno, mohli byste tím uživatele dost znervóznit. Jestli se tedy pro tuto možnost rozhodnete, nezapomeňte to uvést v tématu kanálu a poskytnout i URL archivu.

RSS

RSS (Really Simple Syndication) je mechanismus, který umožňuje zasílat shrnutí novinek (včetně řady metadat) libovolné skupině odběratelů – tedy těch, kdo se k získávání těchto oznámení přihlásili. Zdroj RSS se obvykle nazývá *feed*; uživatelé s nimi pracují prostřednictvím *RSS čtečky* (feed reader nebo feed aggregator). Takovými open source čtečkami jsou například **RSS Bandit** nebo **Feedreader.**

V této knize nemáme dost prostoru na to, abychom si popisovali, jak přesně RSS funguje,[19] ale měli byste mít na paměti dvě důležité věci. První z nich je to, že software pro čtení RSS si volí odběratelé sami a používají jej pro všechny feedy, které sledují – což je koneckonců největší výhoda celého RSS, tedy že všechno lze spravovat prostřednictvím stejného rozhraní, takže feed samotný se může soustředit pouze na poskytnutí obsahu. Druhou je pak to, že RSS už je natolik rozšířené, že většina lidí, kteří

[19] Tyto informace naleznete na **http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html.**

tento systém využívají, o tom ani nevědí. Vědí, že na některých stránkách naleznou tlačítko popsané nějak jako „Odebírat novinky“ nebo „Feed“ a že když na něj kliknou, začne jejich čtečka (což může být třeba součást domácí stránky jejich prohlížeče) automaticky sledovat, jestli se na této stránce neobjevilo něco nového.

Z toho vyplývá, že váš open source projekt by měl RSS poskytovat také (většina služeb kompletního hostingu – viz **Kompletní hosting** – jej nabízí automaticky). Dejte si pozor na to, abyste svůj feed nezavalovali každý den hromadou novinek, aby nemuseli odběratelé trávit čas oddělováním zrna od plev. Pokud se bude feed aktualizovat příliš často, začnou jej odběratelé ignorovat, popřípadě se z něj raději odhlásí. Ideálně by měl projekt nabízet RSS feedů hned několik – jeden pro velká oznámení, další například pro novinky v issue trackeru, další pro každý mailing list atd. Přitom je ale potřeba postupovat celkem opatrně – hrozí totiž, že pak budou vaše stránky natolik komplikované, že se v nich ani návštěvníci, ani jejich správci sami moc nevyznají. Přinejmenším jeden RSS feed byste na hlavní stránce ale mít měli, konkrétně ten určený pro významná oznámení, jako je vydání nové verze nebo bezpečnostní upozornění.[20]

Wiki

Slovem wiki (pochází původně z havajštiny a znamená „rychle“) označujeme webové stránky, které může každý návštěvník upravovat a rozšiřovat, potažmo i software, jenž takový provoz umožňuje. Koncept wiki byl vynalezen už v roce 1995, ale jeho popularita začala strmě růst až zhruba po roce 2000 nebo 2001, částečně díky úspěchu Wikipedie (**http://www.wikipedia.org/**), encyklopedie se svobodným obsahem, která je na tomto systému založena. Wiki jsou tak na půl cesty mezi IRC a webovými stránkami. Nemění se v reálném čase, takže přispěvatelé mají šanci své příspěvky upravovat a vylepšovat, jejich editace je velmi jednoduchá, mnohem jednodušší, než upravovat klasickou webovou stránku.

Použití wiki zatím není u open source projektů standardem, ale v dohledné době zřejmě bude. Vzhledem k tomu, že je celá věc poměrně nová a teprve se zjišťuje, k čemu všemu by mohla sloužit, chtěl bych zde jenom upozornit, že je snazší poukázat na případy, kdy wiki nefungovala tak, jak by měla, než na ty, kde byla použita úspěšně.

Pokud se rozhodnete provozovat wiki, dejte si záležet na tom, aby byly její stránky přehledně organizované a dobře rozvržené, aby návštěvníci (a tedy potenciální přispěvatelé) dokázali instinktivně vytušit, kam by svůj příspěvek měli umístit. Stejně důležité je zveřejnit tyto standardy i na wiki samotné, aby bylo kde nalézt nápovědu. Až příliš často se stává, že si správci wiki představují, že pokud bude velké množství návštěvníků přidávat do wiki kvalitní obsah, bude i výsledek všech těchto dílčích snah kvalitní. Tak to ale nefunguje. Každá jednotlivá stránka nebo odstavec sám o sobě může být dobrý,

---

[20] Abych se tu nechlubil cizím peřím: v prvním vydání knihy tento oddíl chyběl. Význam RSS pro projekty open source mi připomněl zápis na blogu Briana Akera nazvaný **„Release Criteria, Open Source, Thoughts On…“** („Kritéria pro vydávání, open source a co si o tom myslím já…“).

ale pokud bude součástí neorganizovaného nebo matoucího celku, k ničemu mu to nebude. Wiki velmi často trpí těmito nešvary:

- Nejasně definovaná pravidla pro navigaci. Návštěvníci dobře organizovaných stránek vždy vědí, kde se zrovna nacházejí. Pokud jsou vaše stránky dobře navržené, pak všichni intuitivně poznají rozdíl mezi oblastí určenou pro seznam odkazů a pro obsah samotný. Přispěvatelé do wiki budou takové dělení dodržovat, ale jenom tehdy, pokud bude vůbec nějaké existovat.
- Zdvojování informací. Často se stává, že se na stejné wiki nachází několik stránek s podobným obsahem, protože si jednotliví autoři při jejich tvorbě nevšimli, že něco podobného už bylo vytvořeno dřív. To může být do určité míry způsobeno nejasně definovanými pravidly pro navigaci, která jsme zmínili výše – duplicitní stránky nenajdete, pokud nebudou tam, kde byste je čekali.
- Nekonzistentní zaměření na cílové publikum. Tento problém je vzhledem k počtu autorů wiki svým způsobem nevyhnutelný, ale lze jej výrazně omezit, pokud budou existovat písemné pokyny pro tvorbu nového obsahu. Také pomůže, když budete všechny nové záznamy už od začátku velmi agresivně upravovat, abyste tyto standardy prosadili.

Všechny tyto problémy mají společné řešení – vytvoření standardů pro tvorbu wiki a jejich dodržování. Nestačí je tedy jen zveřejnit, je také potřeba upravit stávající stránky tak, aby jim odpovídaly. Obecně se dá říct, že ve všech wiki časem dochází k zesilování nedostatků původního materiálu, protože přispěvatelé budou napodobovat to, co vidí před sebou. Nestačí jen wiki založit a doufat, že zbytek se o sebe postará sám. Musíte ji také nastartovat – naplnit ji dostatečným množstvím dobře napsaného obsahu, aby existoval nějaký standard, kterého se ostatní budou moct držet.

Zářným příkladem dobře vedené wiki je Wikipedie, ale to může být do určité míry způsobeno tím, že pro její obsah, tedy encyklopedická hesla, se formát wiki skvěle hodí. Pokud se ale na Wikipedii podíváte důkladněji, zjistíte, že pravidla pro přispívání, která stanovili její správci, jsou nesmírně podrobná. Existují v ní velmi rozsáhlé návody, jak vytvářet nové záznamy, jak při psaní udržet nestranný pohled, jaké úpravy jsou vhodné a jaké ne, jak probíhá proces diskuze nad spornými úpravami (ten může mít celou řadu dílčích kroků včetně případného odvolání na rozsudek správců samotných) atd. Kromě nich má Wikipedie ještě pojistku ve formě administrativních nástrojů, takže pokud je nějaká stránka vystavena opakovaným nevhodným úpravám, je možné ji zamknout, dokud se problém nevyřeší. Jinými slovy neudělali jen to, že by dali dohromady pár šablon a doufali, že z toho vyleze něco dobrého. Wikipedie funguje tak dobře proto, že ti, kteří ji zakládali, dlouho přemýšleli nad tím, jak přimět tisíce úplně cizích lidí k tomu, aby při psaní sledovali stejný cíl. Při vytváření wiki pro projekt svobodného softwaru tak pečlivou přípravu potřebovat zdaleka nebudete, ale přesto je to dobrý princip, kterého byste se měli držet.

Více informací o wiki naleznete na **http://en.wikipedia.org/wiki/Wiki**. Mimochodem historicky vůbec první wiki stále ještě funguje a vůbec si nevede špatně. Najdete na ní hodně informací o tom, jak wiki spravovat, psaných z různých úhlů pohledu; doporučuji zejména články **http://www.c2.com/cgi/wiki?WelcomeVisitors**, **http://www.c2.com/cgi/wiki?WhyWikiWorks** a **http://www.c2.com/cgi/wiki?WhyWikiWorksNot.**

Webové stránky

O technické stránce tvorby webových stránek projektu není moc co říct – zprovoznění webového serveru a psaní stránek je poměrně jednoduchá záležitost a o tom, jakým způsobem by měly být rozvrženy, jsme si řekli dost už v předchozí kapitole. Hlavní funkcí webových stránek je poskytnout srozumitelný, jednoduchý přehled o celém projektu a spojovat dohromady všechny ostatní nástroje (systém pro správu verzí, bug tracker atd.). Pokud nevíte, jak webové stránky vytvořit, nebude obvykle problém najít někoho, kdo to ví a kdo vám s tím ochotně pomůže. Pokud si ale chcete ušetřit čas a námahu, můžete udělat totéž co mnoho dalších lidí a využít některou ze služeb kompletního hostingu.

Kompletní hosting

Využití služby kompletního hostingu má dvě hlavní výhody. Jednou z nich je kapacita a dostupnost serverů – stroje, na nichž tyto služby běží, jsou obvykle velmi výkonné a k internetu je většinou připojuje vysoce propustná linka. I kdyby byl váš projekt sebeúspěšnější, nikdy nehrozí, že by vám došlo místo na disku nebo že by síťová konektivita byla nedostačující. Druhou výhodou je jeho jednoduchost. Už za vás vybrali bug tracker, systém pro správu verzí, software pro údržbu mailing listu, archivovací nástroj a všechno ostatní, co k provozu stránek potřebujete. Tyto nástroje jsou navíc i předkonfigurované a všechna data, která obsahují, jsou pravidelně zálohována. Není potřeba se nijak dlouze rozhodovat. Stačí vyplnit formulář, stisknout tlačítko a vaše stránky jsou hotové.

To nejsou malé výhody. Stinná stránka pak samozřejmě spočívá v tom, že musíte přijmout volbu a konfiguraci, kterou za vás udělal správce serveru, i když by vašemu projektu lépe posloužilo něco jiného. Služby kompletního hostingu vám obvykle umožní upravit některé parametry, ale nikdy vám neposkytnou tak důkladnou kontrolu, kterou získáte, pokud budete své stránky spravovat sami a mít ke svému serveru přístup administrátora.

To se dá dobře ukázat na příkladu toho, jak bude server zacházet s generovanými soubory. Některé stránky projektu mohou být generované – existují například systémy, které podklady pro FAQ udržují v nějaké podobě, jež se dá velmi snadno upravit, a z ní pak vytvářejí složitější formáty, jako je HTML nebo PDF. Jak jsme již řekli v části **Sledujte verze u všeho** dříve v této kapitole, je nežádoucí, aby byly tyto generované soubory řízeny systémem správy verzí; ten by měl spravovat pouze jejich zdrojový soubor. Pokud jsou ale vaše webové stránky umístěné na serveru někoho jiného, může být nemožné nastavit háček, který by automaticky vygeneroval HTML verzi FAQ pokaždé, když se zdrojový soubor změní. Jediný způsob, jak to obejít, je svěřit pod správu verzí i ty generované formáty, abyste jejich aktualizaci zajistili.

Tento typ hostingu ale může mít i závažnější důsledky. Můžete časem zjistit, že nad svou webovou prezentací nemáte tolik kontroly, kolik byste chtěli. Některé z těchto serverů vám umožní vaše stránky trochu upravit, ale těmito úpravami bude obvykle dál prosvítat jejich přednastavené rozložení, často dost nešikovným způsobem. Například některé projekty, které jsou umístěné na SourceForge,

si své hlavní stránky vytvořily zcela podle svého, ale pro více informací odkazují vývojáře stejně na jejich „stránku na SourceForge“. Tím myslí tu, jež by byla jejich hlavní stránkou, kdyby nepoužili vlastní. Nacházejí se na ní odkazy na bug tracker, úložiště CVS, soubory ke stažení atd. Jenže stránky SourceForge bohužel obsahují ještě spoustu dalších věcí, které jsou pro vaše návštěvníky rušivé. Úplně nahoře je reklama, často animovaná. Na levé straně najdete sloupec odkazů, které těm, kdo se zajímají o váš projekt, k ničemu nejsou. Vpravo je často další reklama. Uprostřed stránky pak najdete informace, které se skutečně týkají vašeho projektu, ale ty jsou často podány způsobem, jenž je pro běžného návštěvníka mimořádně matoucí, takže neví, kam vlastně má kliknout.

SourceForge má pro toto rozložení jistě své dobré důvody, ale jsou to důvody, které jsou dobré pro SourceForge – jako například ta reklama. Z hlediska vašeho projektu ale ideální nejsou. Tím se ale nechci dotknout SourceForge – podobné výhrady můžete mít i k mnoha ostatním serverům nabízejícím kompletní hosting. Chtěl jsem tím říct pouze to, že tyto služby mají i své nevýhody. Sice z vás spadne břemeno spravování stránek projektu, ale zaplatíte za to tím, že musíte akceptovat to, jak je spravuje někdo jiný.

Rozhodnutí, zda je použití takovéto služby pro váš projekt dobré nebo ne, musíte udělat vy sami. Pokud se rozhodnete pro kompletní hosting, ponechte si ale zadní vrátka pro to, abyste se mohli případně časem přesunout na vlastní server, tím, že si zaregistrujete doménu, kterou budete uvádět jako domovskou stránku projektu. Tato doména může jednoduše přesměrovávat na stránky hostingu; můžete na ní také mít samostatnou stránku, již jste si navrhli sami, a její návštěvníky, kteří potřebují nějakou pokročilejší funkci, odkazovat na hosting. Vždy je ale dobrý nápad uspořádat vše tak, abyste měli možnost svůj projekt v případě potřeby přesunout někam jinam, aniž byste museli měnit jeho domovskou adresu.

Jak si vybrat kompletní hosting

Největším a nejznámějším serverem, který kompletní hosting nabízí, je **SourceForge**. Stejné nebo podobné služby nabízejí ještě **savannah.gnu.org** a **BerliOS.de**. Některé další organizace, jako například **Apache Software Foundation** a **Tigris.org**,[21] nabízejí zdarma hosting open source projektům, které odpovídají jejich poslání a zapadají mezi jejich ostatní projekty.

Haggen So pečlivě analyzoval několik různých služeb kompletního hostingu při práci na své Ph.D. dizertaci, *Construction of an Evaluation Model for Free/Open Source Project Hosting (FOSPHost) sites* (Sestavení evaluačního modelu pro hosting projektů svobodného a open source softwaru). Výsledky naleznete na **http://www.ibiblio.org/fosphost/**; zvlášť užitečná je pak srovnávací tabulka na **http://www.ibiblio.org/fosphost/exhost.htm.**

---

[21] Poznámka: Jsem zaměstnancem společnosti **CollabNet**, která Tigris.org sponzoruje, a Tigris pravidelně používám.

Anonymita a zapojení se do projektu

Problémem, jenž se netýká jenom služeb kompletního hostingu, ale obvykle se tam projevuje nejčastěji, je zneužití funkce pro přihlášení uživatele. Tato funkce jako taková je jednoduchá – server umožňuje každému návštěvníkovi zaregistrovat si uživatelské jméno a heslo. Od té doby má tento uživatel na serveru svůj profil a správci projektu mu mohou přidělovat různá práva, jako například právo zapisovat do úložiště.

To může být velmi užitečné; je to ostatně i jedna z největších výhod, které kompletní hosting nabízí. Problém je ale v tom, že je někdy přihlášení se k uživatelskému účtu vyžadováno i u věcí, které by měly být povoleny i neregistrovaným návštěvníkům – konkrétně u možnosti přidávat záznamy a komentáře k záznamům do bug trackeru. Pokud budete pro tyto činnosti vyžadovat přihlášení uživatelským jménem, budete tak komplikovat něco, co by mělo být jednoduché a co nejpohodlnější. Samozřejmě je příjemné mít možnost se s tím, kdo záznam v issue trackeru vytvořil, snadno spojit, ale na to stačí mít pole, kam může, pokud bude chtít, vyplnit svou e-mailovou adresu. Pokud nový uživatel najde chybu a bude ji chtít nahlásit, pak jej povinnost nejprve si založit uživatelský účet může dost otrávit. Dokonce natolik, že se rozhodne chybu vůbec nenahlašovat.

Obecně vzato je pravda, že výhody takové správy uživatelů převažují nad jejími nevýhodami. Pokud ale máte možnost určit, které činnosti lze vykonávat i anonymně, nastavte systém tak, aby povolil anonymní přístup ke všemu, co probíhá v režimu jen pro čtení, a navíc ještě k některým úkonům, které do vašich systémů zapisují, zejména do bug trackeru a do wiki (pokud ji máte).

4. Společenská a politická infrastruktura

4. Společenská a politická infrastruktura — 115
Schopnost vytvářet odnože — **115**

Benevolentní diktátoři — 116
Kdo může být dobrým benevolentním diktátorem? — **117**

Demokracie založená na konsenzu — 118
Řízení verzí znamená, že můžete být v pohodě — **119**
Když nelze dosáhnout konsenzu, hlasujte — **119**
Kdy hlasovat — **120**
Kdo hlasuje? — **121**
Ankety versus hlasování — **122**
Veto — **123**

Jak to všechno zapsat — 123

4. Společenská a politická infrastruktura

První věc, na kterou se lidé ptají, když přijde řeč na svobodný software, je „Jak to funguje? Jak se udržuje projekt v chodu? Kdo rozhoduje?“ Nikdy jsem nebyl spokojen s vágními odpověďmi zmiňujícími meritokracii, duch spolupráce, to, že kód hovoří sám za sebe, a tak dál. Pravdou ale je, že na tuhle otázku se dost těžko odpovídá. Meritokracie, spolupráce a funkční kód k tomu patří, ale samy o sobě jen málo objasňují, jak ve skutečnosti vypadá každodenní chod projektu, nebo to, jak se řeší konflikty.

V této kapitole se pokouším ukázat strukturní základy, které jsou pro úspěšné projekty společné. Když říkám „úspěšné“, nemyslím tím jen jejich technickou kvalitu, ale i jejich provozní zdraví a schopnost přežít. Provozním zdravím rozumím trvalou schopnost projektu zapojovat do sebe nový kód a nové vývojáře a reagovat na příchozí hlášení chyb. Schopnost přežít znamená, že existence projektu není závislá na jakémkoliv jednotlivém vývojáři či sponzorovi – řekněme, že tímto termínem vyjadřujeme pravděpodobnost, že projekt bude pokračovat, i pokud jej všichni zakládající členové opustí a půjdou se věnovat něčemu jinému. Technického úspěchu není obtížné dosáhnout, ale bez solidní vývojářské a společenské základny nemusí být projekt schopen zvládnout růst vyvolaný prvotním úspěchem nebo odchod charismatických jednotlivců.

Tohoto úspěchu lze dosáhnout různými způsoby. Některé z nich zahrnují formální strukturu řízení, v jejímž rámci se řeší debaty, jsou zváni noví vývojáři (a občas opouštěni stávající), plánují se nové funkce a tak dále. V jiných je struktura méně formální, ale existují zde vědomá sebeomezení, vytvářející ovzduší spravedlnosti, na níž lze spoléhat jako na *de facto* formu sebeřízení. Oba způsoby vedou ke stejnému výsledku: jistému pocitu institucionální neměnnosti podporované zvyky a postupy, kterým všichni, kdo se projektu účastní, dobře rozumí. Tyto vlastnosti jsou důležitější v sebeřídících systémech než v těch, které mají centralizované vedení, neboť v systémech, které se řídí samy, si všichni uvědomují, že stačí jeden velký problém, aby se všechno pokazilo, přinejmenším na nějakou dobu.

Schopnost vytvářet odnože

Nezbytným prvkem, který vývojáře v projektech svobodného software sdružuje a který zajistí, že budou v případě potřeby ochotni svolit ke kompromisu, je schopnost zdrojového kódu *vytvářet odnože*: tedy to, že kdokoliv může vzít kopii zdrojového kódu a použít ji jako základ pro konkurenční projekt, kterému se říká *odnož* (fork). Paradoxem je to, že samotná možnost vytvářet odnože je obvykle mnohem významnější silou v projektech svobodného softwaru než odnože samotné, jež jsou velmi vzácné. Protože odnož je špatná pro všechny (z důvodů, jež jsou podrobněji popsány v části **Odnože (Forks)** v kapitole **8. Řízení dobrovolníků**), čím větší je riziko jejího vzniku, tím ochotnější jsou lidé svolit ke kompromisu, aby se této možnosti vyhnuli.

Odnože, respektive potenciál jejich vzniku, jsou důvodem, proč v projektech svobodného softwaru neexistují diktátoři v pravém slova smyslu. To se může zdát jako překvapující tvrzení, vzhledem k tomu, jak často uslyšíme o někom v konkrétním open source projektu, že je „diktátor“ nebo „tyran“. Ale tato

tyranie je tyranií velmi zvláštního typu, značně odlišnou od toho, jak se to slovo běžně chápe. Představte si krále, jehož poddaní by si mohli kdykoliv celé jeho království zkopírovat a přestěhovat se do této kopie, aby si tam vládli, jak se jim zlíbí. Takový král by určitě vládl podstatně jinak než ten, jehož poddaní mu musí zůstat podřízení, ať udělá cokoliv.

Proto ani ty projekty, které nejsou z formálního hlediska organizované demokraticky, se v praxi jako demokracie chovají, když je potřeba učinit závažná rozhodnutí. Následkem replikovatelnosti je schopnost vytvářet odnože; následkem schopnosti vytvářet odnože je konsenzus. Může se klidně stát, že všichni budou ochotní poslouchat jednu vůdčí osobu (nejslavnějším příkladem je Linus Torvalds při vývoji jádra Linuxu), ale jen proto, že se pro to rozhodli, a to způsobem, který není nijak cynický ani nesleduje vedlejší úmysly. Diktátor nedrží projekt v rukách nějakou kouzelnou mocí. Klíčovou vlastností všech licencí svobodného softwaru je to, že nedávají při rozhodování o tom, jak je možné zdrojový kód změnit nebo použít, některé straně víc pravomocí než jakékoliv jiné. Pokud by diktátor náhle začal dělat špatná rozhodnutí, vznikly by nepokoje, které by nakonec dospěly až k vzpouře a vytvoření odnože. K tomu ale, samozřejmě, málokdy dojde, protože dřív diktátor ustoupí ke kompromisu.

Ale to, že schopnost vytvářet odnože stanovuje horní mez moci, kterou může kdokoliv v projektu mít, neznamená, že neexistují významné rozdíly v tom, jak jsou projekty vedeny. Rozhodně nechcete, aby se každé rozhodnutí dostalo až k té úplně poslední otázce, tedy kdo uvažuje vytvořit odnož. To by všechny velmi rychle přestalo bavit a ubíralo by to energii skutečné práci. Další dva oddíly zkoumají různé způsoby, jak organizovat projekty tak, aby většina rozhodnutí prošla hladce. Tyto dva příklady jsou tak trochu idealizované extrémy; mnoho projektů bude ležet někde na ose mezi nimi.

Benevolentní diktátoři

Model *benevolentního diktátora* přesně popisuje už jeho název: pravomoc dělat konečná rozhodnutí leží v rukou jedné osoby, u níž se vzhledem k její osobnosti a zkušenostem očekává, že ji bude používat moudře.

Přestože „benevolentní diktátor“ (nebo *BD*) je pro tuhle roli zavedený termín, lépe by bylo uvažovat o ní jako o „arbitrovi schváleném komunitou“ nebo o „soudci“. Obvykle to není tak, že by benevolentní diktátoři skutečně vykonávali všechna rozhodnutí, dokonce ani většinu z nich ne. Je nepravděpodobné, že by se našla jedna osoba, která by měla tolik odborných znalostí, aby dokázala konzistentně správně rozhodovat ve všech oblastech projektu, a kromě toho dobří vývojáři nezůstanou tam, kde nebudou mít vůbec žádný vliv na to, jakým směrem se projekt ubírá. Proto toho benevolentní diktátoři obvykle moc nediktují. Místo toho nechávají běh věcí plynout, ať se řeší samy prostřednictvím diskusí a tam, kde je to možné, také experimentů. Těchto diskuzí se sami účastní, ale jako řadoví vývojáři, kteří se často podřídí osobě, která je za danou oblast zodpovědná a má víc zkušeností. Pouze tehdy, kdy je zřejmé, že konsenzu nelze dosáhnout, a kdy skupina sama vyžaduje, aby někdo učinil rozhodnutí a vývoj mohl pokračovat, zasáhnou a řeknou „Bude to takhle.“ Neochota přicházet s nařízeními je vlastnost, kterou mají téměř všichni úspěšní benevolentní diktátoři společnou. Je to také jeden z důvodů, proč se v této roli udrželi.

Kdo může být dobrým benevolentním diktátorem?

Role BD vyžaduje kombinaci několika vlastností. Nejdůležitější z nich je dobře vyvinutá schopnost posoudit vlastní vliv na projekt, což vede k tomu, že příslušná osoba dokáže ustoupit do pozadí. V raných fázích diskuze by neměl benevolentní diktátor vyjadřovat své názory a závěry s takovou jistotou, aby ostatní měli pocit, že je zbytečné odporovat. Lidé musí vždy mít svobodnou možnost podělit se o své nápady, i kdyby byly hloupé. Samozřejmě se nevyhnutelně stane, že i sám BD čas od času přijde s hloupým nápadem, a proto tato role vyžaduje i schopnost uvědomit si a uznat, že nějaké rozhodnutí bylo špatné – což je ovšem vlastnost, kterou by měl mít každý dobrý vývojář, zejména pokud je u projektu dlouho. Rozdíl je ale v tom, že BD si může dovolit čas od času udělat chybu, aniž by se musel obávat dlouhodobých následků, které situace bude mít na jeho důvěryhodnost. Vývojáři, kteří nejsou v projektu dlouho nebo jsou na nižších pozicích, si nemusí být svým postavením příliš jistí, takže by BD měl své výhrady či opačné návrhy formulovat citlivě a s ohledem na to, jakou váhu jeho slova mají, ať už po technické či psychologické stránce.

V žádném případě není potřeba, aby měl BD nejlepší technické schopnosti ze všech účastníků projektu. Musí být natolik schopný, aby dokázal na zdrojovém kódu pracovat sám a aby pochopil a dokázal okomentovat jakoukoliv změnu, ale to je všechno. Pozici BD nikdo nezískává ani si neudržuje tím, že by psal fantastický zdrojový kód. To, co je důležité, je zkušenost a celkový smysl pro návrh – ne nutně schopnost vytvořit dobrý návrh na požádání, ale schopnost jej rozpoznat, ať už přichází odkudkoliv.

Benevolentní diktátor je často zakladatelem projektu, což je ale spíše vedlejší účinek než přímý důsledek. Vlastnosti, které umožňují jednotlivci úspěšně spustit projekt – technická zdatnost, schopnost přesvědčit ostatní, aby se přidali, atd. – jsou přesně ty, které potřebuje i každý BD. A samozřejmě platí, že zakladatelé jsou automaticky na tak trochu nadřazenějších pozicích, což často stačí pro to, aby se systém benevolentní diktatury jevil všem zúčastněným jako cesta nejmenšího odporu.

Nezapomínejte, že potenciál vytvářet odnože funguje na obě strany. BD může vytvořit odnož zrovna tak snadno jako kdokoliv jiný, a už se to i párkrát stalo v případech, kdy měli pocit, že směr, kterým by chtěli, aby se projekt ubíral, je jiný než ten, který si přála většina ostatních vývojářů. Díky schopnosti vytvářet odnože nezáleží na tom, jestli má benevolentní diktátor přístup typu root (tedy pravomoci systémového administrátora) k hlavním serverům projektu nebo ne. Lidé často mluví o správě serverů, jako by to byl ten hlavní zdroj moci v celém projektu, ale ve skutečnosti je to úplně irelevantní. Schopnost přidat nebo odebrat hesla pro zápis na jednom konkrétním serveru se vztahuje pouze na tu kopii projektu, která je na tomto serveru uložena. Pokud by tuto pravomoc někdo dlouhodobě zneužíval, ať už BD nebo někdo jiný, vývoj by se jednoduše přesunul na jiný server.

To, zda by váš projekt měl mít benevolentního diktátora nebo zda by mu lépe vyhovoval nějaký jiný, méně centralizovaný systém, do velké míry závisí na tom, kdo je pro tuto roli dostupný. Obecně platí, že pokud všem připadá zcela zjevné, kdo by měl BD být, pak je to ta správná volba. Pokud se ale pro roli BD žádný kandidát automaticky nenabízí, pak by měl projekt pravděpodobně provádět svá rozhodnutí decentralizovaným způsobem, který je popsán v dalším oddílu.

Demokracie založená na konsenzu

S přibývajícím časem projekty obvykle začnou opouštět model benevolentního diktátorství a přecházejí k otevřeněji demokratickým systémům. To nemusí být nutně proto, že by byli se svým BD nespokojeni. Je to jednoduše proto, že řízení vykonávané skupinou je, pokud si mohu vypůjčit metaforu z biologie, „evolučně stabilnější“. Když benevolentní diktátor opustí svou pozici nebo se pokusí rozložit zodpovědnost za rozhodování rovnoměrněji, je to pro skupinu příležitost ustanovit nový, nediktátorský systém – dalo by se říct ústavu jistého typu. Skupina se tak nemusí rozhodnout napoprvé ani napodruhé, ale dříve nebo později to udělá a pak už je velmi nepravděpodobné, že by se k původnímu systému kdy vrátila. Na to, vysvětlit proč, stačí obyčejný selský rozum: pokud se skupina N lidí rozhodne udělit jednomu z nich zvláštní pravomoc, znamená to, že všech N – 1 lidí souhlasilo s tím, že sníží svůj vlastní vliv. A to obvykle nikdo nechce. Nicméně i kdyby to udělali, výsledná diktatura stejně bude jasně podmíněná: byla to skupina, kdo BD jmenoval, a stejná skupina jej tedy může klidně i odvolat. Jakmile tedy projekt přejde od vedení charismatickým jednotlivcem k formálnějšímu systému založeném na skupině, jen málokdy se vrátí zpět.

Způsobů, jimiž tyto systémy fungují, je velmi mnoho, ale mají dva společné rysy: za prvé, většinu času skupina dosahuje konsenzu; za druhé, pro případ, že konsenzu nelze dosáhnout, existuje i formální mechanismus hlasování.

Konsenzus znamená jednoduše dohodu, na kterou jsou ochotni přistoupit všichni. Je to jednoznačný stav: konsenzu v dané otázce je dosaženo tehdy, když někdo navrhne, že bylo dosaženo konsenzu, a nikdo proti tomuto tvrzení nic nenamítá. Osoba, která konsenzus navrhuje, by samozřejmě měla konkrétně uvést, jaký konsenzus je a jaké činnosti z něj vyplývají, pokud to není zřejmé.

Většina diskusí o projektu se týká technických témat, jako je správný způsob, jak opravit konkrétní chybu, zda přidat nebo nepřidat novou funkci, jak podrobně dokumentovat rozhraní atd. Řízení založené na konsenzu funguje dobře proto, že hladce navazuje na technickou debatu samotnou. Na konci diskuse obvykle panuje obecná shoda o tom, jakým způsobem postupovat. Někdo obvykle vystoupí se závěrečným příspěvkem, v němž zároveň shrne, co bylo rozhodnuto, a implicitně navrhne konsenzus. Tím dává poslední šanci komukoliv říct: „Počkejte, s tím já nesouhlasím. Tohle ještě musíme trochu probrat.“

U malých, nekontroverzních rozhodnutí se návrh konsenzu rozumí implicitně. Například pokud vývojář sám od sebe zapíše opravu chyby do systému, návrhem konsenzu je právě zapsání samotné: „Předpokládám, že se všichni shodneme na tom, že tuhle chybu je potřeba opravit a že tohle je ten způsob, jakým by se to mělo udělat.“ To samozřejmě vývojář přímo neříká; pouze zapíše opravu a ostatní z projektu se neobtěžují otevřeně konstatovat svůj souhlas, protože ten vyjadřují právě svým mlčením. Pokud někdo zapíše změnu a ukáže se, že kolem ní konsenzus nepanoval, výsledek je jednoduše ten, že se v projektu změna prodiskutuje, jako by zapsána nebyla. Důvod, proč tenhle systém funguje, je tématem další části.

Řízení verzí znamená, že můžete být v pohodě

To, že je zdrojový kód projektu spravován systémem řízení verzí, znamená, že většinu rozhodnutí lze snadno zvrátit. Nejčastěji se to stává tehdy, když někdo zapíše změnu, protože se mylně domnívá, že s ní všichni budou souhlasit, a protesty se ozvou až potom. V takových případech se obvykle začíná obligátní omluvou, že vývojář nezaregistroval předchozí diskusi, což je ale možné vynechat v případě, že kritizující strana nenalezne v archivech mailing listu o takové diskusi záznam. Tak jako tak, není žádný důvod pro to, aby byl tón diskuse jiný poté, co byla změna zapsána, než předtím. Jakoukoliv změnu lze vrátit, přinejmenším do té doby, než jsou zapsány změny, které na ní závisí (tedy nový kód, který by přestal fungovat, kdyby byla původní změna náhle odstraněna). Systém řízení verzí dává projektu možnost zvrátit následky špatného nebo unáhleného rozhodnutí. Díky tomu pak mohou lidé více důvěřovat svým instinktům ohledně toho, kolik zpětné vazby je potřeba, než se něco provede.

Také to znamená, že celý proces dosažení konsenzu nemusí být příliš formální. U většiny projektů se celá věc dělá instinktivně. Drobné změny lze zapsat bez diskusí, nebo jen s minimální debatou zakončenou pár souhlasnými přikývnutími. U významnějších změn, zejména těch, které mohou potenciálně destabilizovat velké množství zdrojového kódu, by měl vývojář předtím, než začne předpokládat konsenzus, den nebo dva počkat; úvaha je tady taková, že by nikdo neměl být vyjmut z důležité konverzace jen proto, že nekontroloval dost často svůj e-mail.

Když je tedy někdo přesvědčen, že ví, co je potřeba udělat, tak by se měl jednoduše sebrat a provést to. To se netýká jenom oprav softwaru, ale i aktualizací webových stránek, změn dokumentace a všeho ostatního, u čeho je jen malá pravděpodobnost, že to bude zdrojem kontroverzí. Situací, kdy je potřeba nějakou akci zvrátit, obvykle bude velmi málo, a lze je řešit případ od případu. Samozřejmě by ale lidé neměli být podporováni v tom, aby byli příliš tvrdohlaví. Pořád existuje jistý psychologický rozdíl mezi rozhodnutím, které se prodiskutovává, a rozhodnutím, které už je uvedeno do praxe, třebaže jej lze technicky snadno zvrátit. Lidé mají vždy pocit, že setrvačnost a aktivita jdou ruku v ruce, a budou vždy o něco méně ochotni změnu zvrátit, než jí zabránit. Pokud to bude nějaký vývojář zneužívat tím, že bude zapisovat potenciálně kontroverzní změny příliš rychle, ostatní si mohou (a měli by) stěžovat a brát na tohoto konkrétního vývojáře přísnější metr, dokud se situace nezlepší.

Když nelze dosáhnout konsenzu, hlasujte

Některé debaty nevyhnutelně dojdou do stavu, v němž se konsenzu vůbec nedaří dosáhnout. Pokud všechny ostatní způsoby, jak tuto patovou situaci prolomit, selžou, řešením je hlasovat. Ale předtím, než se může hlasovat, je potřeba, aby byly na hlasovacím lístku jasně definované možnosti. Tady opět dochází k tomu, že technická diskuse hladce přechází do rozhodovacího procesu v rámci projektu. Otázky, o nichž se musí hlasovat, se obvykle týkají složitých témat, jež mají rozsáhlé důsledky. V takto komplexní diskuzi se obvykle jeden nebo dva účastníci chopí role *spravedlivého moderátora*: pravidelně sestavují shrnutí jednotlivých argumentů a sledují, v čem spočívají základní příčiny neshody (a shody). Těmito shrnutími pomáhají všem sledovat, jak diskuze postupuje, a připomínají, jaká témata

je ještě potřeba probrat. Stejná shrnutí lze použít jako prototypy pro hlasovací lístek v případě, že bude potřeba hlasovat. Pokud spravedliví moderátoři odvedou svou práci dobře, budou mít dostatečnou autoritu zahájit v příslušný čas hlasování a skupina bude ochotna použít hlasovací lístek, založený na shrnutí celého tématu. Moderátoři sami se mohou debaty účastnit. Není potřeba, aby zůstali zcela mimo, musí být pouze schopni pochopit a férovým způsobem prezentovat názory ostatních a nenechat svůj vlastní pohled ovlivnit své shrnutí debaty, které musí být neutrální.

Samotný obsah hlasovacího lístku obvykle kontroverzní není. V době, kdy se skupina uchýlí k hlasování, už se z diskuse obvykle samo vydestilovalo několik klíčových bodů, které mají své vlastní obecně chápané názvy a krátký popis. Občas se stane, že vývojář vznese námitky k samotné formě lístku. Někdy je taková námitka oprávněná, například tehdy, když je vynechána důležitá možnost nebo je něco popsáno nepřesně. Občas se ale takový vývojář pouze pokouší hrát o čas a oddálit nevyhnutelné, protože cítí, že hlasování dopadne v jeho neprospěch. Rady, jak se vypořádat s podobným kladením překážek, najdete v části **Obtížní lidé** v kapitole **6. Komunikace**.

Nezapomeňte uvést, podle jakého systému se hlasuje, protože jich existuje více typů a lidé mohou chybně předpokládat jiný, než jaký bude použit. Dobrou volbou ve většině případů je volení souhlasem, v němž každý hlasuje pro všechny možnosti na lístku, které se mu líbí. Volení souhlasem se snadno vysvětluje a počítá a na rozdíl od jiných metod stačí jen jedno kolo hlasování. Více informací o systému volení souhlasem a jiných systémech naleznete na **http://en.wikipedia.org/wiki/Voting_system#List_of_systems,** ale pokuste se zamezit rozsáhlé debatě o tom, jaký bude použit (protože pak byste se samozřejmě mohli zamotat do diskuse o tom, který hlasovací systém použít pro výběr hlasovacího systému!). Jedním z důvodů, proč je volba souhlasem dobrý způsob, je to, že se proti ní jen těžko nacházejí námitky – je to asi nejspravedlivější metoda, jaká vůbec existuje.

Nakonec veřejně hlasujte. Není důvod hlasovat tajně nebo anonymně, neboť příslušné otázky byly stejně probírány na veřejnosti. Ať každý účastník pošle své hlasy do mailing listu projektu, aby si mohl výsledky každý sám přepočítat a překontrolovat a aby bylo vše zaznamenáno v archivech.

Kdy hlasovat

Nejtěžší otázka při hlasování je rozhodnout kdy. Obecně se dá říct, že hlasování by mělo být velmi vzácné – poslední útočiště pro případ, že všechny ostatní možnosti selhaly. Nepovažujte hlasování za skvělý způsob, jak uzavírat debaty. Tím rozhodně není. Ukončuje diskusi a s ní i kreativní uvažování o problému. Dokud diskuse pokračuje, existuje pravděpodobnost, že někdo přijde s novým řešením, které se líbí každému. To se stává překvapivě často; živá debata často vyvolá úplně nový způsob uvažování o problému, který pak vede k návrhu, s nímž budou všichni spokojení. I pokud se nový návrh neobjeví, pořád je obvykle lepší dopracovat se ke kompromisu než hlasovat. V situaci kompromisu jsou všichni trochu nešťastní, ale po hlasování jsou někteří nešťastní a někteří šťastní. Z politického hlediska je lepší ta první situace – alespoň má každý pocit, že za svou nespokojenost něco získal. On je sice nespokojen, ale všichni ostatní taky.

Hlavní výhoda hlasování je v tom, že uzavře celé téma a může se pokračovat. Ale tohle uzavření proběhne prostým sčítáním hlav a ne racionálním dialogem, jenž všechny přivede ke stejnému závěru. Podle toho, co jsem vypozoroval, čím zkušenější jsou lidé s open source projekty, tím méně ochotní bývají o problémech hlasovat. Místo toho se raději pokusí prozkoumat řešení, která dosud nebyla navržena, nebo se uchýlit k většímu kompromisu, než měli původně v plánu. Předčasnému hlasování lze předejít pomocí několika technik. Nejvíce se nabízí možnost jednoduše říct „myslím, že ještě nejsme připraveni hlasovat," a vysvětlit proč. Další je požádat o neformální (nezávazné) hlasování prostým zvednutím ruky. Pokud se výsledek jasně vychyluje na jednu stranu, stávají se pak někteří lidé náhle výrazně ochotnějšími svolit ke kompromisu a formální hlasování už není potřeba. Nejefektivnější ale je jednoduše navrhnout nové řešení nebo nový pohled na staré řešení, takže se lidé na problémy podívají znovu, místo aby opakovali stále stejné argumenty.

Ve vzácných případech se může stát, že se všichni shodnou na tom, že všechna kompromisní řešení jsou horší než ta nekompromisní. V těchto případech je hlasování bráno jako přijatelnější, a to jak proto, že povede k lepšímu řešení, tak proto, že nikdo nebude zvlášť nešťastný, ať už situace dopadne jakkoliv. Ani v těchto případech by se ale hlasování nemělo uspěchat. Diskuse vedoucí k hlasování je to, co hlasujícím dodává informace o problému, takže její předčasné ukončení může negativně ovlivnit kvalitu výsledku.

(POZNÁMKA: Tyto rady, které nabádají ke zdrženlivosti při zahajování hlasování, neplatí pro hlasování týkající se zahrnutí změn, popsané v části **Stabilizace release** v kapitole **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji**. V takových případech je hlasování spíše nástrojem komunikace: způsobem, jak zaznamenat svou účast v procesu revize změn, takže všichni vidí, nakolik byla daná změna revidována.)

Kdo hlasuje?

Existence hlasovacího systému vyvolává otázku o tom, kdo je voličská základna, tedy kdo bude hlasovat. Tohle může být citlivé téma, neboť to vyžaduje, aby projekt oficiálně označil některé osoby jako zapojenější nebo lépe uvažující než jiné.

Nejlepším řešením je jednoduše vzít stávající privilegium, tedy přístup k zaznamenávání změn, a podle něj přiřadit hlasovací práva. V projektech, které poskytují jak částečný, tak úplný přístup k zaznamenávání změn, závisí otázka, zda budou moct volit osoby s jen částečným přístupem, na systému, podle nějž je tento částečný přístup přidělován. Pokud projekt rozdává přístupové údaje štědře, například proto, aby v repozitáři udržel velké množství nástrojů, kterými přispívají třetí strany, pak by mělo být ujasněno, že částečný přístup k zaznamenávání se týká právě jen zaznamenávání a ne hlasování. S tím pak přirozeně souvisí opačná implikace: osoby s plným přístupem budou mít i hlasovací právo, a neměly by tedy být vybírány jen jako programátoři, ale také jako členové voličské základny. Pokud někdo v mailing listu prokazuje rušivé tendence nebo často klade překážky, měla by být skupina velmi opatrná, než mu umožní plný přístup, i pokud je to jinak osoba technicky zdatná.

Hlasovací systém samotný by měl být využit při výběru nových osob, jimž budou svěřena práva zápisu, a to jak plná, tak částečná. To je ovšem jeden ze vzácných případů, kdy je tajné hlasování na místě. O potenciálních uchazečích nemůžete hlasovat na veřejném mailing listu, protože byste mohli ublížit jejich citům (a reputaci). Obvykle se to dělá tak, že do soukromého mailing listu pro ty, kteří právo zápisu mají, někdo pošle návrh přidělit určité osobě přístup. Ostatní se k tomuto návrhu mohou svobodně vyjádřit, protože vědí, že diskuze je soukromá. Často se shodnou a hlasování nebude nutné. Po několika dnech čekání, které zajistí, že všichni měli příležitost se vyjádřit, navrhovatel pošle kandidátovi e-mail, v němž mu nabídne oprávnění zapisovat. Pokud nedojde ke shodě, zahájí se diskuse jako u jakékoliv jiné otázky, která může končit hlasováním. Aby byl tento proces otevřený a upřímný, sám fakt, že nějaká diskuse probíhá, by měl zůstat tajný. Pokud by dotyčný kandidát věděl, co se děje, a nedostal pak nabídku přístupu, mohl by dojít k závěru, že v hlasování prohrál a cítil by se poníženě. Samozřejmě pokud někdo o přístup explicitně požádá, pak nezbývá nic jiného než návrh přezkoumat a explicitně jej přijmout nebo odmítnout. V případě odmítnutí by měl být výsledek prezentován co nejzdvořileji a s jasným vysvětlením: „Vaše záplaty se nám líbily, ale ještě jsme jich neviděli dost," nebo „Vaše záplaty jsme ocenili, ale předtím, než je bylo možné aplikovat, potřebovaly ještě velké množství změn, takže dát vám práva zápisu nám zatím nepřipadá jako dobrý nápad. Doufáme ale, že se to časem změní." Nezapomeňte, že to, co říkáte, může být pro danou osobu velká rána, podle toho, jaké je její sebevědomí. Zkuste při psaní e-mailu nahlížet věc z jejich perspektivy.

Protože přidělení oprávnění zapisovat nové osobě má větší důsledky než většina jiných jednotlivých rozhodnutí, zavádějí pro něj některé projekty zvláštní pravidla. Například mohou vyžadovat, aby návrh získal alespoň n pozitivních hlasů a žádný negativní hlas, nebo aby pozitivních hlasů bylo minimálně určité procento. Přesné parametry nejsou důležité, hlavní myšlenkou je, aby byla skupina opatrná, než přibere někoho nového. Podobné a možná ještě přísnější požadavky mohou mít hlasování o odebrání těchto pravomocí, i když to snad nebude nikdy nutné. Více informací o nehlasovacích aspektech přibírání a odebírání zápisových práv naleznete v části **Committeři** v kapitole **8. Řízení dobrovolníků.**

Ankety versus hlasování

Pro některé typy hlasování může být užitečné voličskou základnu rozšířit. Například pokud vývojáři jednoduše nedokážou zjistit, zda daná změna v rozhraní odpovídá tomu, jak lidé jejich software skutečně používají, jedním z řešení je požádat všechny členy projektového mailing listu o názor. Zde se jedná spíše o ankety než o hlasování, ale jejich výsledek mohou vývojáři brát jako závazný. Jako u každé jiné ankety nezapomeňte zdůraznit, že existuje i možnost napsat vlastní návrh – pokud někdo vymyslí něco lepšího, než jsou možnosti, které anketa nabízí, může právě jeho odpověď být na celé anketě to nejcennější.

Veto

Některé projekty umožňují ještě zvláštní druh hlasu, který se označuje jako veto. Veto je způsob, jakým může vývojář pozastavit unáhlenou nebo nedomyšlenou změnu přinejmenším na tak dlouho, než bude důkladněji prodiskutována. O vetu lze uvažovat jako o nástroji někde na půl cesty mezi velmi silnou námitkou a hrou o čas. Jeho přesný význam se v jednotlivých projektech dost liší. V některých projektech je veto velmi těžké přehlasovat, jinde k tomu postačuje obyčejný většinový souhlas, před nímž je ale vynucený čas pro další diskusi. Každé veto by mělo doprovázet pečlivé odůvodnění. Pokud veto nijak odůvodněné není, mělo by být automaticky považováno za neplatné.

Právo veta s sebou nese i problém jeho zneužití. Občas se stává, že vývojáři velmi rádi debatu popíchnou vyhlášením veta ve chvíli, kdy byla pouze potřeba další diskuze. Zneužití veta lze předcházet tím, že jej budete používat jen velmi zřídka a neochotně a tento princip nenápadně zmíníte, když uvidíte, že někdo své právo veta využívá až příliš často. Pokud je to nutné, můžete také skupině připomenout, že veto je závazné jen tehdy, pokud s tím skupina souhlasí – koneckonců, pokud většina vývojářů chce X, pak se X dříve či později stane. Buď se vývojář, který veto vyhlásil, stáhne, nebo se skupina rozhodne oslabit význam veta.

Občas uvidíte, že někdo vyjadřuje veto tím, že napíše „-1“. Tento způsob má svůj původ v organizaci Apache Software Foundation, která má vysoce strukturovaný systém pro vetování a hlasování, jenž je popsán na **http://www.apache.org/foundation/voting.html**. Standardy Apache se rozšířily i do dalších projektů a v open source světě se s jejich zvyklostmi v různé míře setkáte poměrně často. Technicky vzato neznamená „-1“ vždy formální veto, a to ani podle standardů Apache, ale neformálně se obvykle chápe jako veto, nebo přinejmenším velmi silná námitka.

Stejně jako hlasy i veto lze vyhlásit retroaktivně. Není správné odmítnout veto na základě toho, že příslušná změna už byla zaznamenána nebo příslušná činnost vykonána (pokud to tedy není něco nevratného, jako je vydání tiskové zprávy). Na druhou stranu veto, které přijde se zpožděním několika týdnů nebo měsíců, nikdo brát příliš vážně nebude, což je koneckonců správně.

Jak to všechno zapsat

Po určité době se můžete dostat do situace, kdy už bude množství zvyklostí a dohod, jež ve vašem projektu existují, tak velké, že je nutné je někde zaznamenat. Aby byl takový dokument legitimní, je třeba zdůraznit, že vyplývá z diskusí v mailing listu a dohod, které již platí. Při psaní vycházejte z příslušných vláken v archivech mailing listu a kdykoliv narazíte na něco, co vám není jasné, zeptejte se znovu. V dokumentu by nemělo být nic překvapivého. Není původním zdrojem dohod, jež jdou v něm obsaženy, ale pouze je shrnuje. Samozřejmě, pokud bude tento dokument úspěšný, lidé jej začnou používat jako autoritativní zdroj, ale to jen znamená, že přesně odráží celkové názory skupiny.

O tomto dokumentu se zmiňuje část **Pokyny pro vývojáře** v kapitole **2. Zahájení projektu**. Samozřejmě když je projekt velmi mladý, musíte uvádět pravidla, aniž byste měli k dispozici historii, z níž můžete těžit. Jak ale bude vývojářská komunita dozrávat, můžete formulace upravit podle toho, jak věci budou skutečně fungovat.

Nepokoušejte se vyčerpat celé téma. Žádný dokument nemůže shrnout všechno, co by lidé o účasti v projektu měli vědět. Mnohé ze zvyklostí projektu zůstanou nikdy nevyslovené, nikdy je nikdo explicitně nezmíní, ale přesto se jimi všichni budou řídit. Další věci jsou zase příliš evidentní na to, aby je někdo vypisoval, a jenom by ubíraly pozornost tomu, co je důležité, ale ne evidentní. Je například zcela zbytečné psát rady jako „Při přispívání do mailing listů buďte zdvořilí, mějte úctu k ostatním a nevyvolávejte zbytečné hádky," nebo „Pište čistý, přehledný kód bez chyb." To jsou samozřejmě věci, které jsou pro projekt vhodné, ale vzhledem k tomu, že neexistuje myslitelná situace, v níž by vhodné nebyly, je nestojí za to zmiňovat. Pokud jsou lidé v mailing listu neurvalí nebo píšou kód plný chyb, nepřestanou jen proto, že je to v pravidlech projektu zakázáno. Takové situace je potřeba řešit, když nastanou, a ne paušálními závazky, že všichni budou hodní. Na druhou stranu ale pokud v projektu existují specifické postupy, jak psát dobrý zdrojový kód, jako jsou například pravidla o dokumentování každého API v určitém formátu, pak by tyto postupy měly být popsány tak podrobně, jak to jen bude možné.

Dobrým způsobem, jak určit, na čem by měl být dokument založen, jsou otázky, které nejčastěji kladou nováčci, a námitky, které nejčastěji vznášejí zkušení vývojáři. To nemusí nutně znamenat, že by váš dokument měl mít formu nejčastěji kladených otázek (FAQ) – bude pravděpodobně vyžadovat souvislejší strukturu, než jakou formát FAQ může nabídnout. Měl by ale sledovat stejný princip zakotvení v realitě, tedy reagovat na situace, které skutečně nastávají, spíš než na ty, jež si myslíte, že by nastat mohly.

Pokud je projekt spravován systémem benevolentní diktatury nebo v něm existují osoby se zvláštními pravomocemi (prezident, předseda, nebo něco podobného), pak je tento dokument také dobrou příležitostí kodifikovat systémy pro následnictví. Někdy to lze dělat velmi jednoduše jmenováním konkrétních lidí jako náhradníků pro případ, že BD projekt z jakéhokoliv důvodu náhle opustí. Obecně se dá říct, že pokud projekt má BD, tak námitky nevyvolá pouze to, když si svého nástupce jmenuje sám. Pokud existují volené pozice, pak by měl být v dokumentu popsán postup nominace a volby, jímž byli lidé na tyto pozice vybráni. Pokud žádný takový postup ustanoven nebyl, pak je potřeba získat v mailing listu konsenzus o tom, jak by měl vypadat, ještě předtím, než o tom začnete psát. Na hierarchické struktury jsou lidé někdy velmi citliví, takže k tomuto tématu přistupujte opatrně.

Asi nejdůležitější je zdůraznit, že pravidla lze změnit. Pokud zvyklosti popsané v dokumentu začnou projekt brzdit, připomeňte všem, že to má být reflexe záměrů skupiny a ne zdroj frustrací nebo nepřekonatelná překážka. Pokud má někdo ve zvyku, že žádá o změnu pravidel pokaždé, když se mu připletou do cesty, není nutné o tom pokaždé diskutovat – někdy je tou nejlepší taktikou zůstat zticha. Pokud s námitkami jiní lidé souhlasí, ozvou se také a bude jasné, že se něco musí změnit. Pokud nikdo jiný nesouhlasí, pak se daná osoba nedočká mnoha reakcí a pravidla zůstanou tak, jak jsou.

Dva dobré příklady projektových pravidel jsou Subversion Community Guide na **http://subversion.apache.org/docs/community-guide/** a řídící dokumenty Apache Software Foundation na **http://www.apache.org/foundation/how-it-works.html** a **http://www.apache.org/foundation/voting.html**. ASF je v zásadě jakýmsi souborem softwarových projektů, který je právně veden jako nezisková organizace, takže jejich dokumenty více popisují postupy řízení než zvyklosti vývoje. Přesto ale stojí za přečtení, neboť se v nich odrážejí nahromaděné zkušenosti mnoha open source projektů.

5. Peníze

5. Peníze — 129

Typy zapojení do projektu — 130
Pracovníky najímejte na dlouhodobý úvazek — 132
Vystupujte jako jednotlivci, nikoli jako celek — 133
Otevřeně hovořte o své motivaci — 134
Lásku si za peníze nekoupíte — 136

Dodavatelské smlouvy — 137
Kontrola a přijetí změn — **140**
Případová studie: protokol pro ověřování hesla CVS — **140**

Financování neprogramovacích činností — 141
Zajišťování kvality (tj. Profesionální testování) — **141**
Právní poradenství a ochrana — **143**
Dokumentace a použitelnost — **143**
Poskytování hostingu/síťové propustnosti — **144**

Marketing — 145
Pamatujte na to, že jste sledováni — **145**
Nekritizujte konkurenční open source produkty — **147**

5. Peníze

Tato kapitola posuzuje možnosti financování v prostředí svobodného softwaru. Nezaměřuje se pouze na vývojáře, kteří jsou za práci na projektech svobodného softwaru placeni, ale také na jejich nadřízené, kteří musí porozumět sociální dynamice vývojového prostředí. V následujících částech této knihy se předpokládá, že čtenář („vy“) je buďto placený vývojář nebo někdo, kdo tyto vývojáře řídí z pozice nadřízeného pracovníka. Rady a tipy budou pro obě skupiny čtenářů často stejné; případně bude příslušná cílová skupina jasně určitelná z kontextu.

Financování svobodného softwaru z firemních prostředků není novým fenoménem. Velká část vývojové činnosti byla vždy neformálně financována z jiných zdrojů. Když správce systému vytvoří nástroj síťové analýzy, který mu má usnadnit práci, poté jej zveřejní na internetu a ostatní správci k tomuto nástroji následně doplňují různé opravy programátorských chyb a další užitné vlastnosti, jsme vlastně svědky vzniku neoficiálního konsorcia. Takovéto konsorcium je financováno z platů správců systému, dále se na dotování tohoto konsorcia – byť nevědomky – podílejí i organizace, pro něž příslušní správci systému pracují, tomuto konsorciu totiž věnují kancelářské prostory a síťovou propustnost. Tyto organizace pak samozřejmě z této investice těží, ačkoliv si ji na institucionální rovině zprvu ani neuvědomují.

Na rozdíl od dřívějška se však dnes již mnoho těchto snah a činností formalizuje. Firmy si začaly uvědomovat výhody, které z open source softwaru plynou, a stále častěji se přímým způsobem podílejí na jeho vývoji. I vývojáři dnes očekávají, že skutečně důležité projekty přitahují alespoň dárce a v některých případech i dlouhodobé sponzory. Zatímco přítomnost peněz základní dynamiku vývoje svobodného softwaru nikterak nezměnila, značně ovlivnila rozsah, v němž celá tato činnost probíhá, jak co se týče počtu vývojářů tak i jejich časové vytíženosti. Rovněž ovlivnila způsob organizování jednotlivých projektů a interakci mezi jejich jednotlivými účastníky. Základní otázka již nespočívá v tom, jak budou určité finanční prostředky vynaloženy nebo jakým způsobem se měří návratnost investic. Hlavní problematika se přesunula do oblasti managementu a procesu: jak mohou hierarchické struktury firem produktivně spolupracovat se zpola decentralizovanými komunitami dobrovolníků zapojených do projektů svobodného softwaru? Shodnou se vůbec na významu pojmu „produktivně“?

Obecně lze říci, že komunity vývojářů open source softwaru finanční krytí vítají. Může totiž redukovat zranitelnost a citlivost projektu vůči „Silám Chaosu“, které smetou mnoho projektů dříve, než vůbec odstartují, a může tak lidi přesvědčit o tom, aby se softwarem spojili svou důvěru—, že investují svůj čas do něčeho, co bude žít déle než jen pouhých šesti měsíců od svého vzniku. Důvěryhodnost je do určité míry nakažlivá. Pokud nějaký open source projekt podporuje, řekněme, IBM, lidé předpokládají, že tento projekt jen tak nepadne, a jejich výsledná ochota věnovat tomuto projektu úsilí, může být právě tím faktorem, který tato očekávání naplní.

Financování však s sebou nese i pojem kontroly. Pokud nebude projekt veden pečlivě, mohou jej peníze rozštěpit na skupinu zasvěcených vývojářů a skupinu outsiderů. Pokud začnou mít neplacení dobrovolníci pocit, že designová rozhodnutí nebo funkční doplňky jsou jednoduše dostupné zájemcům

s nejvyšší finanční nabídkou, budou odcházet k projektům, založeným spíše na oceňování zásluh, nežli na neplacené práci v cizí prospěch. Nejspíše si ani nebudou vehementně stěžovat na mailing listech. Naopak, na externích zdrojích začne postupně ustávat pracovní šum, jednotliví dobrovolníci budou totiž pozvolna vzdávat svou snahu o to, aby je někdo bral vážně. Nadále bude pokračovat pracovní činnost menšího rozsahu, ve formě zpráv o chybách (bug reports) a příležitostných drobných oprav. Nebudou však již přicházet žádné větší příspěvky do zdrojového kódu, diskuzí o designu se nebude účastnit nikdo zvnějšku. Lidé dobře cítí, co se od nich očekává, a tato očekávání dokážou naplňovat v kladném i záporném smyslu.

Ačkoliv je s penězi nutno zacházet opatrně, neznamená to, že si za ně nelze koupit vliv. To zcela určitě lze. Vtip spočívá v tom, že si nelze tento vliv koupit přímo. V běžné, tj. přímé, obchodní transakci měníte peníze za to, co chcete. Pokud chcete doplnit nějakou užitnou vlastnost či funkci, podepíšete smlouvu, zaplatíte a zboží je vaše. U open source projektu to tak jednoduše nefunguje. S některými vývojáři sice můžete podepsat smlouvu, ale lhali by sami sobě—i vám—, kdyby garantovali, že vámi zaplacená práce bude komunitou vývojářů plně akceptována, a to jen proto, že jste za ni zaplatili. Takováto práce může být akceptována pouze na základě svých předností a podle toho, jak zapadá do představy (vize) celé komunity o daném softwaru. V této vizi sice může mít vaše slovo jistou váhu, ale nikdy o ní nebudete rozhodovat zcela sami.

Z toho vyplývá, že za peníze se sice vliv koupit nedá, ale lze za ně koupit věci, *které* k vlivu vedou. Jako nejjasnější příklad lze uvést programátory. Pokud si najmete dobré programátory a budou mít dostatek času se seznámit se softwarem a získat si důvěru komunity, pak mohou projekt ovlivnit stejným způsobem jako každý jiný účastník. Budou mít hlasovací právo nebo, pokud jich bude hodně, mohou vytvořit hlasovací blok. Pokud jsou v rámci projektu dostatečně respektováni, budou mít vliv i mimo prosté hlasování. Placení vývojáři stejně nemají důvod jakkoli maskovat svou motivaci. Každý, kdo chce provést jakoukoli změnu v softwaru, má k tomu přece určitý důvod. Důvody vaší společnosti nejsou o nic méně legitimnější než důvody kohokoli jiného. Vážnost, která bude přičítána cílům vaší společnosti, určuje pouze statut jejích zástupců v rámci projektu, nikoli velikost společnosti, její rozpočet nebo podnikatelský záměr.

Typy zapojení do projektu

Open source projekty jsou financovány z různých důvodů. Položky tohoto seznamu se vzájemně nevylučují; v mnoha případech je finanční krytí projektu výsledkem několika, nebo dokonce všech, těchto motivů:

Sdílení společného břemene
Samostatné organizace se souvisejícími softwarovými potřebami se často dostávají do situace, kdy zbytečně zdvojují svá úsilí, buďto nadbytečným vytvářením podobného kódu přímo v podniku nebo zakoupením podobných produktů od proprietární prodejců. Pokud by si tyto společnosti uvědomily, o co v takovémto případě vlastně jde, spojily by své zdroje a vytvořily

(nebo se připojily k) open source projektu, který by byl přesně přizpůsoben jejich specifickým potřebám. Výhody jsou očividné: náklady na vývoj se rozdělí mezi jednotlivé účastníky, přičemž prospěch z něj budou mít všichni. Ačkoliv se tento scénář zdá být vhodný pro neziskové organizace, může mít strategický význam i pro výdělečné společnosti.
PŘÍKLADY: **http://www.openadapter.org/, http://www.koha.org/**

Rozšiřování služeb
Pokud určitá společnost prodává služby, které jsou závislé na určitých open source programech nebo je lze těmito programy zatraktivnit, je v jejím zájmu, aby zajistila aktivní udržování a podporu těchto programů.
PŘÍKLAD: Podpora **CollabNet http://subversion.tigris.org/** (disclaimer/vyloučení odpovědnosti: má každodenní práce, ale také skvělý příklad tohoto modelu).

Podpora prodeje hardwaru
Hodnota počítačů a jejich komponentů je přímo úměrná množství softwaru, který je pro ně dostupný. Prodejci hardwaru—, nikoli jen prodejci celých počítačových sestav, ale také výrobci periferních zařízení a mikročipů—přišli na to, že pro zákazníky je důležité mít k dispozici kvalitní svobodný software, který lze na tomto hardwaru spouštět.

Podlomení konkurenta
Firmy někdy podporují určitý open source projekt jako způsob podlomení síly konkurenčního produktu, který může nebo nemusí být sám open source. Postupné ujídání z tržního podílu konkurence obvykle není jediným důvodem k podílení se na open source projektu, může však hrát svou významnou roli.
PŘÍKLAD: **http://www.openoffice.org/** (ne, tohle není jediný důvod, proč vznikl a funguje OpenOffice, tento software je však alespoň částečnou reakcí na Microsoft Office).

Marketing
Skutečnost, že je vaše společnost spojována s populární open source aplikací, může sloužit jako příklad správného vedení značky (brand managementu).

Dual licensing (Multi-licence)
Dual licensing je postup, kdy se nabízí software v rámci tradiční proprietární licence zákazníkům, kteří jej chtějí dále prodat jako součást jejich vlastní aplikace a zároveň se tento software nabízí v rámci volné licence těm, kdo jej chtějí používat dle podmínek platných pro open source software (viz **Systém dvojitých licencí** v kapitole **9. Licence, autorská práva a patenty**). Pokud je komunita vývojářů open source softwaru aktivní, software získává výhodu širokoplošného ladění, odstraňování chyb (debugging) a vývoje, firma však nadále pobírá licenční poplatky, z nichž podporuje určitý počet programátorů pracujících na plný úvazek.
DVA NEJZNÁMĚJŠÍ PŘÍKLADY JSOU: **MySQL**, tvůrci stejnojmenného databázového softwaru, a **Sleepycat**, nabízející distribuci a podporu Berkeley Database. Není náhodou, že se obě společnosti zabývají databázemi. Databázový software bývá spíše zakomponován do aplikací a neprodává se uživatelům přímo, skvěle se tak hodí do modelu dual-licensingu.

Dary
Široce používaný projekt může v některých případech získat značné příspěvky jak od fyzických, tak od právnických osob, k tomu stačí jednoduché tlačítko „online donation“ (dary online) nebo prodej značkového propagačního zboží, např. hrnků, triček, podložek pod myš apod. Upozornění: pokud váš projekt přijímá dary, dobře si rozplánujte, jak budou konkrétní peníze využity, a to ještě *dřív* než je obdržíte, tento plán pak vyvěste na své internetové stránce. Diskuze o konkrétním přidělování finančních prostředků mívají hladší průběh, pokud probíhají ještě předtím, než peníze skutečně obdržíte; a kromě toho, pokud v dané problematice existují vážné neshody, je lepší o nich vědět, dokud se celá debata odehrává na teoretické bázi.

Podnikový model financovatele není jediným faktorem, který určuje jeho vztah ke komunitě open source. Rovněž závisí na dřívějších vztazích mezi těmito dvěma subjekty: byl projekt spuštěn společností nebo se společnost jen připojila ke stávající vývojářské činnosti? V obou případech si bude muset financovatel získat důvěru, v druhém případě se bude samozřejmě muset snažit o něco více. Organizace musí mít jasně stanovené cíle stran projektu. Snaží se společnost udržet vedoucí pozici nebo chce být pouze jedním z členů celé komunity, chce celý projekt pouze usměrňovat nikoli jej řídit? Nebo chce mít jen po ruce pár vývojářů s právem potvrzování změn (committerů), kteří budou schopni opravit zákaznické chyby a tyto úpravy pak bez dalších zmatků uvést do běžné veřejné distribuce?

Při čtení následujících zásad mějte tyto otázky neustále na paměti. Platí pro jakýkoli způsob zapojení organizace do projektu svobodného softwaru, každý projekt však vytváří určité prostředí lidí a žádné dva projekty nejsou nikdy navlas stejné. Vždy budete muset do určité míry improvizovat, ale při dodržení těchto zásad zvýšíte pravděpodobnost, že se věci začnou vyvíjet vámi požadovaným způsobem.

Pracovníky najímejte na dlouhodobý úvazek

Pokud řídíte programátory při práci na open source projektu, snažte se je udržet u projektu dostatečně dlouho, aby získali řádné technické i politické dovednosti a schopnosti, —tj. nejméně po dobu několika let. Žádnému projektu, ať z oblasti open source nebo closed source, neprospívá časté obměňování programátorů. Zapracovávání nováčka je v každém prostředí brzdou. V open source projektech je však trest za fluktuaci pracovníků mnohem tvrdší, odcházející vývojáři si totiž s sebou odnášejí nejen znalost kódu, ale i svůj statut v rámci komunity a osobní vztahy, které si zde vytvořili.

Důvěryhodnost, které vývojář dosáhl, pochopitelně nelze přenášet na jiné subjekty. Nejkřiklavější příklad: nově příchozí vývojář nemůže od odcházejícího vývojáře převzít commit access (právo na potvrzení změn v kódu) (vizte dále v části **Lásku si za peníze nekoupíte** v této kapitole), takže pokud tento nový vývojář dosud commit access nemá, bude muset předkládat záplaty ke schválení, dokud jej nezíská. Commit access je však jen ten nejmarkantnější důkaz o ztrátě vlivu. Dlouhodobě zaměstnaný vývojář zná všechny starší spory, které byly znovu a znovu přetřásány na diskusních fórech. Nový vývojář, který nebyl svědkem těchto rozhovorů, může již dávno probraná témata opět nadhodit k diskuzi, čímž vaše organizace ztratí důvěryhodnost; ostatní účastníci diskuze se pak mohou s podivem ptát „Copak

si tam ti jejich vývojáři vůbec nic nepamatují?“ Nový vývojář rovněž nemá žádné politické cítění pro jednotlivé osoby v rámci projektu, nebude tedy schopen ovlivňovat směr vývoje tak rychle a hladce jako někdo, kdo se na projektu podílí již delší dobu.

Nově příchozí vývojáře zaškolujte pomocí programu s kontrolovaným zapojením uživatelů. Nový vývojář musí být od prvního dne v přímém kontaktu s veřejnou komunitou vývojářů, počínaje opravou programových chyb a čistícími úlohami, aby se mohl seznámit s kódovou bází a získal v rámci komunity určitou reputaci, aniž by vyvolával jakékoli delší diskuze o designu. Po celou dobu by měl být k dispozici jeden či více vývojářů pro zodpovídání případných otázek, měli by rovněž číst všechny příspěvky, které nový vývojář umístí na vývojářský mailing list, ačkoliv jsou tyto příspěvky ve vláknech, jimž by se zkušení vývojáři vůbec nevěnovali pozornost. Tato opatření by měly skupině pomoci odhalit skryté útesy ještě předtím, než na ně loď nového vývojáře narazí a klesne ke dnu. Soukromé zákulisní stimuly a rady mohou také značně pomoci, zejména v případě, že nový vývojář není navyklý na masivně paralelní posuzování jeho kódu ze strany kolegů z oboru.

Když chce CollabNet zaměstnat nového vývojáře pro práci na projektu Subversion, sedneme si spolu a vybereme několik otevřených chyb, na nichž se nově příchozí vývojář blíže seznámí s prací, která jej čeká. Diskutujeme o technických návrzích řešení a poté pověříme alespoň jednoho zkušeného vývojáře, aby (veřejně) zkontroloval a revidoval záplatu, kterou nový vývojář (rovněž veřejně) vyvěsil. Obvykle si danou záplatu ani neprohlížíme, dokud ji neuvidí vývojářský mailing list, ačkoliv bychom mohli, kdyby pro to byl nějaký důvod. Důležité je, aby nový vývojář prošel procesem veřejné kontroly, naučil se zdrojové kódy a přitom si zvykl přijímat kritiku od zcela neznámých lidí. Snažíme se však zkoordinovat načasování tak, aby naše vlastní kontrola a revize následoval bezprostředně po vyvěšení dané záplaty. První posudek, který se na mailing listu objeví, je tedy náš, což do jisté míry určí tón ostatních posudků. Předpokládáme, že díky tomu bude nový zaměstnanec brán vážně: pokud ostatní uvidí, že věnujeme čas vytvoření detailních posudků s důkladnými vysvětlivkami a případnými odkazy na archivy, uznají, že probíhá určitá forma školení, což pravděpodobně naznačuje dlouhodobou investici. Ostatní účastníci projektu tak budou pozitivněji naladěni vůči novému vývojáři, alespoň do té míry, že budou ochotni věnovat trochu času odpovědím na jeho případné dotazy a revizím jeho záplat.

Vystupujte jako jednotlivci, nikoli jako celek

Vaši vývojáři by se měli snažit o to, aby na veřejných fórech v rámci projektu vystupovali spíše jako jednotliví účastníci, nikoli jako jednolitá firemní skupina. Není to kvůli tomu, že by každá takováto jednolitá firemní skupina s sebou vždy nutně nesla negativní konotace (no, možná opravdu nese, ale o tom tahle kniha nepojednává). Je to kvůli tomu, že jednotlivci jsou jediným typem subjektů, s nimiž struktura open source projektů dokáže efektivně jednat. Jednotlivý přispěvatel se může zapojovat do diskuzí, předkládat záplaty ke schválení, získávat důvěryhodnost, volit atd. Firma nemůže.

Navíc, pokud se sami budete chovat decentralizovaně, nebudete k centralizaci podněcovat ani opozici. Dovolte svým vývojářům, aby na mailing listech spolu navzájem nesouhlasili. Mějte je k tomu, aby si vzájemně kontrolovali a revidovali kódy stejně často a stejně veřejně jako kódy všech ostatních

účastníků projektu. Zrazujte je od toho, aby vždy hlasovali jako jeden celek, pokud tak budou činit, ostatní mohou brzy nabýt dojmu, že je pod kontrolou udrží pouze dobře organizovaná snaha.

Mezi skutečnou decentralizací a snahou o její simulaci je velký rozdíl. Za určitých okolností je docela užitečné, aby vaši vývojáři jednali ve vzájemné shodě, v případě potřeby musí být připraveni koordinovat svou činnost v zákulisí. Například, pokud se podává určitý návrh, je vhodné mít k dispozici několik lidí, kteří od počátku vyjadřují svůj souhlas, tím vzbudí dojem postupně narůstajícího konsenzu, což pomůže daný návrh prosadit. Ostatní budou mít dojem, že předložený návrh má dostatečnou hybnou sílu a jejich případné námitky by tuto sílu zbytečně narušovaly. Lidi tak budou vznášet námitky pouze v případě, že pro ně mají opravdu dobrý důvod. Na takto zinscenované dohodě není nic špatného, pokud se nadále vážně projednávají všechny případné námitky. Veřejná oznámení o soukromých dohodách nejsou o nic méně upřímnější, ač byla zkoordinována předem, a nejsou také nikterak škodlivá, pokud se nepoužívají předpojatě k eliminaci protivníkových argumentů. Jejich cílem je pouze potlačit či utlumit vliv lidí, kteří vznášejí námitky jen proto, aby nevyšli ze cviku; o nich viz více v části **Čím jednodušší téma, tím delší debata** v kapitole **6. Komunikace**.

Otevřeně hovořte o své motivaci

O cílech své organizace hovořte co nejotevřeněji, aniž byste však porušili obchodní tajemství. Pokud chcete, aby byl váš projekt obohacen o nějakou novou funkční vlastnost, protože ji zákazníci vyžadují, řekněte to přímo v mailing listech. Pokud chtějí zákazníci zůstat v anonymitě, což se někdy stává, požádejte je alespoň, zda je můžete použít jako nejmenované příklady. Čím více ví vývojářská komunita o důvodech, *proč* chcete to, co chcete, tím snadněji budou přijímat vaše návrhy.

Tento efekt zcela odporuje snadno osvojenému—instinktu, kterého se lze jen stěží zbavit,—že vědění je moc, že čím více vědí ostatní o vašich záměrech, tím snadněji vás mohou ovládat. V tomto případě vás však tento instinkt klame. Veřejným obhajováním určité funkční vlastnosti (nebo záplaty chyby či čehokoli jiného), jste již vyložili své karty na stůl. Jedinou otázkou teď je, zda se vám podaří usměrňovat komunitu tak, aby sdílela váš záměr. Pokud pouze prohlásíte, že to chcete, ale nebudete schopni uvést žádný konkrétní příklad proč, vaše argumentace bude chabá a lidi začnou mít podezření na skrytou agendu. Naopak, pokud uvedete jen několik scénářů z reálného světa, na nichž doložíte, proč je navrhovaná funkční vlastnost důležitá, můžete tak dramaticky ovlivnit celou debatu uvnitř komunity.

Posuďte následující alternativu, která vám ukáže, proč tomu tak je. Debaty o nových funkčních vlastnostech nebo nových směrech vývoje jsou v mnoha případech dlouhé a únavné. Předkládané argumenty se zpravidla zúží na tvrzení typu „Já osobně chci X," nebo stále populárnější výrok „Za ty roky, co navrhuji software, můžu s plným vědomím prohlásit, že X je pro uživatele nesmírně důležité / zbytečný luxus, který nikomu nijak nepomůže." Lze předpokládat, že absence údajů o skutečném využití některé funkční vlastnosti takovéto debaty ani nezkrátí, ani neuklidní, naopak bude je odvádět dál a dál od jakéhokoli zakotvení ve zkušenostech opravdových uživatelů. Bez určitých vyvažujících sil asi nebude konečný výsledek určovat to, kdo je nevýřečnější, nejvytrvalejší nebo nejvýše postavený.

Jako organizace, která má k dispozici velké množství údajů o zákaznících, máte možnost takovéto vyvažující síly zajistit. Můžete se stát kanálem informací, které by se za jiných okolností k vývojářské komunitě nemohly vůbec dostat. Za to, že informace podněcují vaše touhy, se nemusíte stydět. Většina vývojářů nemá nijak velké zkušenosti s tím, jak je software, který vytvořili, využíván. Každý vývojář využívá software osobitým způsobem; spoléhá na svou intuici a odhad a v hloubi duše o tom dobře ví. Poskytnutím věrohodných dat o velkém množství uživatelů dáváte komunitě vývojářů něco jako kyslík k dýchání. Pokud budete tyto údaje správně prezentovat, nadšeně je uvítají, a věci se tak pohnou vámi požadovaným směrem.

Klíčová je právě ona správná prezentace těchto údajů. Vůbec vám nepomůže, pokud budete trvat na tom, aby vaše řešení bylo implementováno jen a jen kvůli tomu, že přicházíte do styku s velkým počtem uživatelů a že tito uživatelé potřebují (nebo se domníváte, že potřebují) určitou funkční vlastnost. Své první komentáře byste naopak měli zaměřit na vlastní problém, nikoli na jedno konkrétní řešení. Velmi podrobně popište, s čím se vaši zákazníci potýkají, poskytněte co nejvíce analýz, které máte k dispozici, a co nejvíce rozumných řešení, která vás napadla. Jakmile lidé začnou spekulovat o efektivnosti různých řešení, můžete dále využívat své údaje a na jejich základě podporovat nebo vyvracet jednotlivá tvrzení. Celou dobu můžete mít na mysli jedno konkrétní řešení, ale zprvu jej nepředkládejte ke zvláštnímu posuzování. Nedopouštíte se tím žádného podvodu, chováte se jako „moudrý hospodář“. Vždyť vaším skutečným cílem je vyřešit problém; řešení je čistě prostředkem, který k tomuto cíli vede. Pokud je řešení, které upřednostňujete, skutečně kvalitní, ostatní vývojáři to nakonec sami uznají—a poté se za něj sami dobrovolně postaví, což je mnohem lepší, než kdybyste je do implementace tohoto řešení nutili zastrašováním. (Je dokonce možné, že přijdou na lepší řešení).

Tím nechci říci, že byste neměli svou podporu určitému řešení projevovat nikdy. Musíte však s maximální trpělivostí sledovat, jak se analýza, kterou jste si již dříve interně provedli, sama opakovaně vyvíjí na veřejných vývojářských mailing listech. Nevyvěšujte komentáře typu: „Ano, na to už jsme přišli, nefunguje to, protože A, B a C. Když se do toho pořádně vnoříte, zjistíte, že jediný způsob řešení je ...“ Problém není ani tak v tom, že budete působit arogantně, spíše vzbudíte dojem, že jste *již* do daného problému – za zavřenými dveřmi – investoval neznámé (ale pravděpodobně významné) analytické zdroje. Bude to celé vypadat, že ačkoliv je na celý problém vynakládáno velké úsilí a pravděpodobně již byla přijata určitá řešení, vývojářská veřejnost vlastně není do celého problému řádně zasvěcena, což je ten nejlepší recept, jak odradit lidi.

Samozřejmě, že dobře *víte*, kolik úsilí jste danému problému v rámci firmy věnovali; vědomí o tomto úsilí je jistým způsobem vaší nevýhodou. Staví totiž vaše vývojáře do poněkud odlišné pozice oproti ostatním členům mailing listu, oslabuje jejich schopnost nahlížet věci z úhlu pohledu, který mohou zaujmout vývojáři, již dosud o daném problému tolik neuvažovali. Čím dříve se vám podaří, aby všichni ostatní uvažovali o věcech ve stejných intencích jako vy, tím slabší bude tento efekt odcizení. Tento princip platí nejen pro jednotlivá technická řešení, ale i pro širší kontext, v němž se snažíte co nejvíce osvětlit své cíle a záměry ostatním. Neznámé je vždy více destabilizující nežli známé. Jakmile lidé pochopí, proč chcete to, co chcete, budou s vámi rádi hovořit a jednat, byť s vámi nemusí přímo souhlasit. Pokud nepochopí důvody vašeho jednání, budou - přinejmenším v některých případech – očekávat to nejhorší.

Přirozeně nebudete moci zveřejnit úplně všechno a lidé to od vás ani nebudou očekávat. Každá organizace má svá tajemství; výdělečné organizace jich možná mají více, ale i v neziskové organizaci se nějaké to tajemství střeží. Pokud musíte obhajovat určitý směr vývoje, ale nemůžete odhalit všechny důvody, proč tak činíte, tak jednoduše předložte ty nejlepší argumenty, které v rámci své kompetence předložit můžete, a přijměte jako nutný fakt, že v diskusi prostě nebudete mít takový vliv, jaký byste si možná přáli. Je to jeden z kompromisů, který musíte učinit, abyste vývojářské komunitě nemuseli vyplácet mzdu.

Lásku si za peníze nekoupíte

Pokud pracujete na projektu jako placený vývojář, určete si hned na začátku, co lze a co nelze za peníze koupit. To neznamená, že byste museli dvakrát denně přispívat na mailing list a dokola opakovat vaše vznešené a neúplatné zásady. To pouze znamená, že byste se měli mít na pozoru před pokušeními a tlaky, které *mohou* peníze vytvářet. Nemusíte vycházet z předpokladu, že takovéto tlaky vždy byly a budou; musíte však prokázat vědomí o tom, že takové tlaky mohou kdykoli vyvstat.

Dokonalý příklad k tomuto tématu se objevil i v rámci projektu Subversion. Projekt Subversion spustila v roce 2000 společnost CollabNet, která od samotného začátku fungovala jako jeho primární financovatel, zaměstnávala několik vývojářů (disclaimer: Jsem jedním z nich). Krátce po zahájení projektu jsme zaměstnali dalšího vývojáře, Mike Pilata, který se připojil k našemu úsilí. V té době již bylo zahájeno kódování. Ačkoliv byl celý projekt Subversion dosud jen v prvních fázích vývoje, byla k němu již ustavena vývojářská komunita s kodexem základních pravidel.

Příchod nového vývojáře Mika vyvolal zajímavou otázku. Subversion již měl stanovená pravidla, jak bude případný nový vývojář dostávat commit access. Nejprve musí předložit několik záplat do vývojářského mailing listu. Po předložení dostatečného počtu záplat, které ostatní committery přesvědčí, že nový přispěvatel ví, co dělá, někdo navrhne, aby tento nový přispěvatel začal přímo potvrzovat provedené změny v kódu (tento návrh je soukromý, vizte **Committeři**). Pokud s tím budou ostatní committeři souhlasit, jeden z nich zašle dotyčnému novému vývojáři e-mail a nabídne mu přímé právo commitu do úložiště projektu.

Společnost CollabNet si najala Mika přímo pro práci na projektu Subversion. Ti, kdo jej znali, nepochybovali o jeho schopnostech v oblasti kódování a jeho důkladné průpravě k práci na projektu. Kromě toho měli dobrovolní vývojáři velice dobré vztahy se zaměstnanci CollabNet a pravděpodobně by nic nenamítali, kdybychom Mikovi udělili commit access hned při jeho přijetí do projektu. My jsme však věděli, že bychom tím vytvořili určitý precedens. Kdybychom Mikovi udělili commit access „z moci úřední“, vyslali bychom tím signál, že CollabNet může ignorovat projektová pravidla a směrnice, a to jen proto, že je primárním financovatelem projektu. Ačkoliv by následky takového rozhodnutí nemusely být bezprostředně zřejmé, mohlo by postupně dojít k tomu, že by se neplacení vývojáři začali cítit kráceni na svých právech. Zatímco ostatní si musí commit access vysloužit—, CollabNet si ho prostě koupí.

Mike tedy souhlasil s tím, že u CollabNet začne jako každý jiný dobrovolný vývojář, tj. bez commit access. Zasílal záplaty na veřejný mailing list, kde je mohl kdokoli kontrolovat a posuzovat. Na mailing listu jsme rovněž uvedli, že tak činíme úmyslně, aby nebylo nic vynecháno. Po několika týdnech řádně odvedené práce někdo (už si nepamatuji, jestli vývojář z CollabNet nebo někdo jiný) navrhl udělit Mikovi commit access; návrh byl přijat, o čemž jsme od prvopočátku nepochybovali.

Díky zásadovosti tak získáte důvěru, kterou si za peníze nikdy nekoupíte. A důvěryhodnost je na poli technických diskusí cenná deviza: působí jako jistá imunizace proti pozdějším pochybám o motivech vašeho jednání. V zápalu boje se mohou lidé občas uchýlit i k netechnickým argumentům, jen aby spor vyhráli. Primární financovatel projektu, díky své angažovanosti a očividnému zájmu o to, jakým směrem se bude projekt ubírat, je v tomto kontextu tím nejsnadnějším terčem. Přísným dodržováním všech směrnic a pravidel projektu od jeho začátku se financovatel staví na stejnou úroveň jako všichni ostatní účastníci projektu.

(Podobný případ s commit access viz také v blogu Danese Coopera zde **http://blogs.sun.com/roller/page/DaneseCooper/20040916.** Cooper byl tehdy „Open Source Divou“ —u Sun Microsystem, zdá se mi dokonce, že to byl její oficiální titul—, ve svém blogu popisuje, jak komunita vývojářů Tomcat přinutila společnost Sun, aby její vlastní vývojáři dodržovali stejné commit access standardy jako všichni ostatní mimofiremní vývojáři).

Nutnost, aby financovatelé hráli podle stejných pravidel jako všichni ostatní, znamená, že se v přítomnosti finančních zdrojů poněkud hůře prosazuje řídící model benevolentní diktatury (viz **Benevolentní diktátoři** kapitola **4. Společenská a politická infrastruktura**), zvláště v případě, že diktátor pracuje pro primárního financovatele. Jelikož má diktatura jen málo pravidel, pro financovatele je dosti obtížné dokázat, že se řídí standardy komunity, byť by tak skutečně činil. Rozhodně to není nemožné; chce to však mít takového vedoucího projektu, který je schopen pohlížet na různé věci z pohledu externích vývojářů i z pohledu financovatele, a podle toho jednat. I tak je pravděpodobně vhodné mít v záloze návrh na nediktátorské řízení projektu, které lze vynést na světlo ve chvíli, kdy se uvnitř komunity objeví první známky rozsáhlé nespokojenosti.

Dodavatelské smlouvy

S prací na smlouvu by se v rámci projektů svobodného softwaru mělo zacházet velice opatrně. V ideálním případě by dílo dodavatele měla schválit komunita vývojářů a poté by se předalo do veřejné distribuce. Teoreticky by mělo být jedno, kdo je dodavatelem, pokud je jeho práce kvalitní a splňuje projektové směrnice a pravidla. I teorie se občas snoubí s praxí: naprosto neznámý vývojář, který přijde s dobrou záplatou, *bude* obecně schopen tuto záplatu zakomponovat do softwaru. Problém je v tom, že vytvořit skutečně dobrou záplatu pro složitý softwarový doplněk nebo novou funkční vlastnost je velice těžké, zvláště pro někoho, kdo je v daném oboru naprosto neznámý; celý problém se nejprve musí prodiskutovat se zbylými účastníky projektu. Trvání této diskuse nelze přesně odhadnout. Pokud je smluvní

dodavatel placen od hodiny, může dojít k tomu, že budete platit více, než jste si původně mysleli; pokud je placen paušálně, může to skončit tak, že bude dělat více práce, než si může ve skutečnosti dovolit.

Těmto nebezpečím se lze vyhnout dvěma způsoby. Upřednostňovaným způsobem řešení je kvalifikovaně odhadnout délku diskusního procesu na základě předchozích zkušeností, včetně časové rezervy pro případné chyby, a podle tohoto odhadu poté koncipovat celou smlouvu o dodávce. Tento způsob vám také pomůže rozdělit celý problém na co možná nejvíce menších, nezávislých kousků, u nichž pak lze jednotlivě zvýšit pravděpodobnost správného odhadu. Druhý způsob spočívá v uzavření smlouvy o dodávce záplaty; přijetí této smlouvy v rámci veřejného projektu se pak řeší jako další oddělený problém. Poté je již mnohem snazší sepsat smlouvu, ale brzdí vás nutnost udržovat soukromou záplatu, dokud jste na daném softwaru závislí nebo přinejmenším do chvíle, kdy se vám podaří zakomponovat tuto záplatu nebo ekvivalentní funkčnost do hlavního programu. Samozřejmě ani v případě výše uvedené preferované varianty nemůže smlouva sama o sobě vyžadovat, aby byla záplata přijata do kódu, neboť by se jednalo o prodej něčeho, co není určeno k prodeji. (Co když se zbývající členové projektu nečekaně rozhodnou nepodpořit danou funkční vlastnost?) Smlouva však může vyžadovat *upřímnou snahu* o přijetí změny v komunitě a o její zápis do úložiště, pokud s tím bude komunita souhlasit. Například, pokud má projekt psané standardy týkající se změn kódu, může smlouva na tyto standardy odkazovat a stanovit, aby příslušná práce (dílo) tyto standardy splňovala. V praxi tato opatření obvykle fungují podle představ všech zúčastněných.

Nejlepší taktika pro úspěšné uzavírání smluv s dodavateli je najmout jednoho z projektových vývojářů—, nejlépe committera—jako dodavatele. Může to vypadat, jako byste si kupovali vliv, a je to skutečně tak. Ale rozhodně to není tak korupční jednání, jak to vypadá. Vliv vývojáře v rámci projektu je patrný především kvůli kvalitě jím vypracovaného kódu a jeho interakcím s ostatními vývojáři. Skutečnost, že tento vývojář podepsal smlouvu na provedení určitých prací, nijak nepovyšuje jeho statut, ani jej nesnižuje, ačkoliv jej možná někteří lidé začnou pečlivěji kontrolovat, hlídat. Většina vývojářů by neriskovala svou dlouhodobou pozici v projektu kvůli podpoře nevhodné nebo obecně neoblíbené funkční vlastnosti. Pokud zaměstnáte takovéhoto dodavatele, lze očekávat, že vám poradí nebo by měl poradit, jaký druh změn bude pravděpodobně komunitou přijat. Rovněž tak dosáhnete určitého posunu v prioritách celého projektu. Jelikož se priority stanovují podle toho, kdo má čas na čem pracovat, pokud si zaplatíte něčí čas, jeho práce se na žebříčku priorit trochu posune směrem vzhůru. Tento jev je mezi zkušenými open source vývojáři dobře znám a alespoň někteří z nich budou věnovat práci smluvního dodavatele pozornost, čistě z toho důvodu, že se pravděpodobně blíží ke konci, a tak by rádi přispěli k tomu, aby to byl konec zdárný. Možná nebudou zapisovat žádný kód, ale budou alespoň diskutovat o designu a kontrolovat kód, což jsou obě velice užitečné činnosti. Ze všech těchto důvodů je nejlepší vybírat smluvního dodavatele ze skupiny osob, které se na daném projektu již podílejí.

Tím okamžitě vyvstávají dvě otázky: Měly by být tyto smlouvy vůbec uzavírány neveřejně? A pokud ne, měli byste se obávat toho, že se uvnitř komunity vytvoří napětí kvůli tomu, že jste s některými vývojáři uzavřeli smlouvy a s jinými ne?

O smlouvách byste měli mluvit zcela otevřeně, pokud je to možné. Jinak se může chování smluvního dodavatele zdát ostatním v komunitě podezřelé—, když začne z ničeho nic nevysvětlitelně klást vysokou prioritu funkčním vlastnostem, o které se dříve ani v nejmenším nezajímal. Když se jej zeptají, proč chce najednou mít tyto funkční vlastnosti v projektu zakomponovány, jak může tento vývojář podat svým kolegům přesvědčivou odpověď, když nebude moci hovořit o tom, že je k vytvoření těchto funkčních vlastností smluvně vázán?

Zároveň byste se ani vy, ani smluvní dodavatel neměli chovat tak, aby ostatní pokládali vaše vzájemné smluvní ujednání za něco světoborného. Často jsem byl svědkem toho, jak tito smluvní dodavatelé povýšeně vplouvali na mailing list a tvářili se přitom, jakoby jejich příspěvky měly být brány vážněji z toho prostého důvodu, že jsou za svou práci na projektu placeni. Takový způsob chování ostatním účastníkům projektu signalizuje, že daný smluvní dodavatel považuje svou smlouvu za důležitější—než vlastní kód, *který z této* smlouvy—vzejde. Z pohledu ostatních vývojářů je však tím nejdůležitějším právě kód. Po celou dobu by se měla pozornost soustředit na technické záležitosti, nikoli na takové detaily, jako kdo koho platí. Například jeden z vývojářů v komunitě Subversion řeší celou problematiku smluv zvlášť elegantním způsobem. Při diskusích o svých změnách kódu v IRC jen tak mimochodem zmíní (často formou soukromé poznámky, IRC *privmsg* před jedním z ostatních committerů), že je za svou práci na dané programové chybě nebo funkční vlastnosti placen. Ale zároveň budí neustále dojem, že by na dané změně chtěl tak jako tak pracovat a je vlastně šťastný, že mu peníze umožňují plnit si toto své přání. Může nebo nemusí odhalit identitu svého klienta, v každém případě se vlastní smlouvou nijak zvlášť nezaobírá. Jeho poznámky o ní jsou jen drobným doplňkem jinak čistě technických diskusí o tom, jak to či ono správně provést.

Tento příklad ukazuje další důvod, proč je vhodné hovořit o smlouvách zcela otevřeně. V určitém open source projektu může být několik organizací sponzorujících takovéto smlouvy o dílo a když budou navzájem vědět, o co se ostatní organizace snaží, mohou své zdroje sloučit. Ve výše uvedeném případě se největší financovatel (CollabNet) nijak nepodílí na těchto smlouvách o úkolové práci, ale pokud se dozví, že někdo jiný sponzoruje opravu určité chyby, může CollabNet přesměrovat své zdroje na chyby jiné, což vede k vyšší efektivitě projektu jako celku.

Budou se ostatní vývojáři pohoršovat nad tím, že jsou někteří jejich kolegové za práci na projektu placeni? Obecně nikoli, zvláště pokud jsou placení vývojáři etablovaní a uznávaní členové komunity. Nikdo nepředpokládá, že bude práce na smlouvu rovnoměrně rozdělena mezi všechny committery. Lidé chápou důležitost dlouhodobých vztahů: nejistoty, které s sebou najímání smluvních dodavatelů přináší, jsou takového rázu, že jakmile narazíte na někoho, s kým lze spolehlivě pracovat, nejste už příliš nakloněni myšlence ohlížet se po někom dalším jen pro zachování nestrannosti. Podívejte se na celou problematiku takto: při vašem prvním najímání pracovníka si nikdo nemůže na nic stěžovat, musíte si přeci *někoho zvolit*—; a nemůžete za to, že si nelze najmout všechny. Později, když si najmete stejného pracovníka podruhé, máte pro to ty nejrozumnější praktické důvody: už jej znáte, posledně se osvědčil, tak proč riskovat? Je tedy zcela přirozené mít spíše jednoho či dva „dvorní“ vývojáře v rámci komunity, než pracovní úkoly rovnoměrně rozdělovat mezi všechny zúčastněné.

Kontrola a přijetí změn

Komunita je však pro úspěšnost smluvního díla nadále velice důležitá. Podíl členů komunity na tvorbě designu a proces kontroly u rozsáhlejších změn nelze vynechat. Musí být nadále vnímány jako součást celé práce a smluvní dodavatel je musí do své činnosti řádně zahrnout. Nevnímejte kontrolu ze strany komunity jako překážku, kterou je nutno překonat, —chápejte ji jako svobodný projektový tým a oddělení pro zajišťování kvality. Je Vaší výhodou, pokud jste aktivně sledován, a nikoli jen trpěn.

Případová studie: protokol pro ověřování hesla CVS

V roce 1995 jsem byl jedním ze dvou členů společenství, které poskytovalo podporu a rozšiřování CVS (Concurrent Versions System, viz **http://www.cvshome.org/**). Společně s Jimem, mým partnerem, jsme tehdy neformálně zajišťovali údržbu CVS. Nikdy jsme však nějak hlouběji nepřemýšleli o tom, jak bychom se měli správně chovat ke stávající, většinou dobrovolné, vývojářské komunitě CVS. Předpokládali jsme, že budou prostě zasílat záplaty a my je budeme aplikovat; takhle nějak celý systém práce v podstatě fungoval.

Tenkrát bylo možno síťově propojit CVS pouze přes vzdálený přihlašovací program, např. `rsh`. Použití stejného hesla pro přístup na CVS i pro přihlašování bylo jednoznačným bezpečnostním rizikem a mnoho organizací tato skutečnost odrazovala. Jedna z velkých investičních bank si nás najala, abychom do programu přidali nový ověřovací mechanismus, který by zajistil bezpečné používání síťového CVS i v oddělených pracovištích a pobočkách této banky.

Společně s Jimem jsme podepsali smlouvu a začali vytvářet nový ověřovací systém. Vytvořili jsme velice jednoduchý systém (Spojené státy tehdy kontrolovaly export zdrojových kódů umožňujících šifrování, klient byl tedy srozuměn s tím, že jsme nemohli do programu implementovat silnou autentizaci), jelikož jsme však ve vytváření těchto protokolů nebyli tak zběhlí, dopustili jsme se několika přehmatů, které by skutečný expert ihned odhalil. Tyto chyby by byly snadno zachyceny, kdybychom měli dost času sepsat návrh a nechat jej projít kontrolou ostatních vývojářů. To jsme však neučinili, protože nás nenapadlo použít mailing list jako vhodný zdroj. Věděli jsme, že lidé asi přijmou jakoukoli změnu, kterou v kódu potvrdíme, a —jelikož jsme nevěděli, co nevíme,—neobtěžovali jsme s tím, abychom celou práci prováděli viditelným způsobem, např. častým vyvěšováním záplat, krátkými, snadno stravitelnými změnami v samostatných větvích apod. Výsledný ověřovací protokol nebyl příliš dobrý a po jeho etablování už bylo samozřejmě složité jej jakkoli vylepšovat z důvodů ohrožení jeho kompatibility.

Celý problém nepramenil z nedostatku zkušeností; snadno jsme si mohli nastudovat, co jsme potřebovali vědět. Problém byl v našem přístupu ke komunitě dobrovolných vývojářů. Přijímání změn jsme považovali spíše za překážku než za proces, kterým lze zvýšit kvalitu změn. Jelikož jsme si byli jisti, že téměř vše, co uděláme, bude přijato (a skutečně bylo), ani jsme se moc nesnažili zapojit do procesu ostatní vývojáře.

Pokud si vybíráte dodavatele, pochopitelně chcete někoho se správnými technickými dovednostmi a zkušenostmi vhodnými pro danou práci. Je však rovněž důležité vybrat někoho, kdo se může prokázat dlouhodobou konstruktivní interakcí s ostatními vývojáři v komunitě. Získáte tak více než jednu jedinou osobu; zajistíte si zprostředkovatele, který bude schopen čerpat z celé sítě odborných dovedností a znalostí, a zajistit tak stabilní a udržitelné provedení celého díla.

Financování neprogramovacích činností

Programování je pouze částí práce, která v open source projektech probíhá. Z pohledu projektových dobrovolníků se jedná o nejviditelnější a nejpřitažlivější část celého projektu. To však bohužel znamená, že ostatní činnosti, jako např. dokumentace, formální testování apod., mohou být někdy zanedbávány, alespoň v porovnání s tím, jaké pozornosti se tyto činnosti těší u proprietárních softwarů. Firemní organizace jsou někdy schopny tuto nevýhodu kompenzovat tím, že pro open source projekty vyčlení některou ze svých interních infrastruktur na vývoj softwaru.

Klíčovým momentem pro úspěšné zvládnutí tohoto problému je správný převod mezi interními procesy ve firmě a procesy veřejné vývojářské komunity. Takový převod nebývá snadný: často se obě sféry špatně slučují a rozdíly mezi nimi může přemostit pouze lidský zásah. Firma například může používat jiný bug tracker (systém pro sledování chyb) než veřejný projekt. A byť by třeba používali stejný monitorovací software, data v něm ukládaná budou velice odlišná, protože potřeby firmy v oblasti sledování chyb se velice liší od potřeb komunity zapojené ve svobodném softwaru. Informaci, která začala v jednom monitorovacím softwaru, může být nutno reflektovat i v druhém softwaru, s odstraněním důvěrných částí nebo naopak s doplněním důvěrných částí.

Následující kapitoly pojednávají o tom, jak vytvářet a udržovat takové převodové mosty mezi oběma sférami. V konečném důsledku by měl open source projekt běžet hladčeji, komunita by měla uznat zdroje, které firma do projektu investovala, aniž by měla pocit, že firma nepřiměřeným způsobem usměrňuje vývoj ve prospěch svých vlastních cílů.

Zajišťování kvality (tj. Profesionální testování)

Při vývoji proprietárního softwaru se běžně využívají týmy, které se věnují výhradně zajišťování kvality: vyhledávání chyb, testování výkonnosti a škálovatelnosti, kontrola rozhraní a dokumentace atd. Tyto aktivity zpravidla dobrovolná komunita v rámci projektu svobodného softwaru neprovádí tak nadšeně a energicky. Je to zčásti kvůli tomu, že je obtížné získat dobrovolníky pro tak neatraktivní práci, jako je testování, a zčásti kvůli tomu, že si lidé často myslí, že velká komunita uživatelů dává celému projektu dostatečné pokrytí testy; v případě testování výkonnosti a přizpůsobitelnosti je to rovněž způsobeno tím, že dobrovolníci mají málokdy přístup k nezbytným hardwarovým zdrojům. Domněnka, že velké množství uživatelů znamená také velké množství testerů, není zcela neopodstatněná. Opravdu nemá příliš velký smysl vyčleňovat tým testerů pro základní funkcionalitu v běžných

prostředích: zde budou chyby rychle a zcela přirozeně odhaleny uživateli. Protože se však uživatelé pouze snaží dokončit rozdělanou práci, vědomě se nepokoušejí prozkoumávat nezmapované hraniční případy ve funkčnosti programu a pravděpodobně přejdou některé typy chyb zcela bez povšimnutí. Dále, pokud objeví nějakou chybu s jednoduchým řešením, často toto řešení mlčky implementují, aniž by chybu vůbec nahlásili. Největší zrada tkví v tom, že se mohou modely používání vašimi klienty (tj. lidmi, kteří řídí *váš* zájem o software) lišit, a to statisticky velice významně, od modelů používání softwaru průměrnými uživateli (lidmi z ulice).

Profesionální testovací tým může tyto typy chyb odhalit a může tak učinit stejně snadno u svobodného softwaru jako u softwaru proprietárního. Hlavním úkolem je předat výsledky testovacího týmu v použitelné podobě zpět veřejnosti. Firemní testovací oddělení mají obvykle svůj vlastní způsob hlášení a prezentace zkušebních výsledků, který tvoří specifický žargon nebo specializované znalosti o určitých klientech a jejich skupinách dat. Tyto zprávy o výsledcích nemusí být právě vhodné pro veřejný bug tracker, jednak díky jejich formě, jednak z důvodu zachování důvěrnosti informací. I kdyby byl váš interní firemní bug tracker stejný jako software užívaný veřejným projektem, management vaší společnosti by mohl například vyžadovat možnost vkládání vlastních komentářů a změn v metadatech, jež jsou specifické pro vaši společnost (např. zvýšení interní priority u určité záležitosti nebo časový plán jejího řešení u konkrétního klienta). Tyto poznámky a komentáře jsou obvykle důvěrné—, někdy zůstávají utajeny i před klientem. Ale i v případě, že důvěrné nejsou, nemají pro veřejný projekt žádný velký význam, a neměly by tedy veřejnost zbytečně rozptylovat či mást.

Základní bug report však *je* pro veřejnost velice důležitý. Bug report, který obdržíte od vašeho testovacího oddělení, je v určitém ohledu cennější než obecný report od uživatelů, neboť testovací oddělení sleduje problémy, kterým se ostatní uživatelé nevěnují. Vzhledem k tomu, že z žádného jiného zdroje asi tento konkrétní bug report neobdržíte, určitě si jej budete chtít ponechat a zpřístupnit jej i veřejnému projektu.

Za tímto účelem může oddělení QA (zajišťování kvality) vložit tyto problémy (issues) do veřejného trackeru, pokud jsou s tímto postupem spokojeni, nebo může zprostředkovatel (zpravidla jeden z vývojářů) „přeložit/převést“ interní zprávy testovacího oddělení do nových problémů „issues“ ve veřejném trackeru. Překlad/převod jednoduše znamená popsat chyby způsobem, který nikterak neodkazuje na informace specifické pro daného klienta (reprodukční předpis může využívat data klienta; pochopitelně za předpokladu, že je klient schválí).

Je do jisté míry vhodnější, aby QA oddělení zakládalo problémy do veřejného trackeru přímo. Veřejnost tak bude moci přímo ocenit zapojení vaší společnosti v celém projektu: užitečné bug reporty dodávají vaší organizaci na důvěryhodnosti podobně jako všechny ostatní technické příspěvky. Kromě toho poskytují vývojářům přímé komunikační propojení s testovacím týmem. Například: interní QA tým monitoruje veřejný issue tracker, vývojář může potvrdit opravu chyby v přizpůsobitelnosti softwaru (pro jejíž testování však nemá k dispozici žádné zdroje) a k danému problému (issue) přidat poznámku, v níž požádá QA tým, aby prověřil, zda má tato oprava očekávaný efekt. U některých vývojářů očekávejte jistý odpor; programátoři mají tendenci pohlížet na QA jako na (přinejlepším) nutné zlo. QA tým může tento despekt jednoduše překonat nalezením významných chyb a založením srozumitelných

reportů; na druhou stranu, pokud nebudou jejich reporty přinejmenším tak dobré jako ty, které přicházejí z komunity běžných uživatelů, nemá smysl, aby byl tento tým v přímé interakci s týmem vývojářů.

Tak či tak, jakmile existuje nějaký problém ve veřejném trackeru, původní interní problém by měl jednoduše odkazovat na veřejný problém, včetně technického obsahu. Management a placení vývojáři mohou nadále dle potřeby komentovat a připojovat poznámky k internímu problému pomocí specifických firemních poznámek/komentářů, veřejný problém však budou využívat kvůli informacím, které by měly být k dispozici každému.

Tento proces byste měli absolvovat a očekávat určité dodatečné režijní náklady. Udržování dvou záznamů u jediné chyby znamená přirozeně více práce než udržování jediného. Výhodou bude skutečnost, že si report pročte mnohem více programátorů, kteří dokážou přijít s nějakým řešením.

Právní poradenství a ochrana

Společnosti, výdělečné či neziskové, jsou snad jediné subjekty, které věnují pozornost komplexní právní problematice svobodného softwaru. Jednotliví vývojáři sice často chápou nuance různých licencí na open source software, ale obecně nemají čas či prostředky na to, aby se podrobně zaobírali autorským, známkovým a patentovým právem. Pokud má vaše společnost právní oddělení, může projektu napomoci prověřením autorských práv váznoucích na kódu; vývojářům může osvětlit případnou patentovou a známkovou problematiku. Přesná podoba této poradenské činnosti je pojednána v kapitole **9. Licence, autorská práva a patenty**. Především je nutno zajistit, aby komunikace mezi právním oddělením a komunitou vývojářů, pokud vůbec k nějaké dojde, probíhala ve vzájemné úctě vůči tak rozdílným světům, z nichž jednotliví účastníci této komunikace pocházejí. Tyto dvě skupiny spolu příležitostně promluví, aniž by se vzájemně poslouchaly; předpokládají přitom, že specifickými znalostmi jedné skupiny nedisponuje skupina druhá. Vhodnou strategií v tomto případě je ustanovit prostředníka (zpravidla vývojáře nebo naopak právníka s technickými znalostmi), který bude stát mezi oběma skupinami a podle potřeby jim „tlumočit“.

Dokumentace a použitelnost

Dokumentace a použitelnost jsou notoricky známé slabiny open source projektů, ačkoliv si myslím, že přinejmenším v případě dokumentace se rozdíl mezi svobodným a proprietárním softwarem často zbytečně zveličuje. Nicméně je i tak empiricky dokázáno, že mnoho open source softwarů postrádá prvotřídní výzkum v oblasti dokumentace a použitelnosti.

Pokud chce vaše organizace tyto mezery v rámci projektu zacelit, bude nejlepší najmout si lidi, kteří sice *nejsou* regulérními vývojáři projektu, ale jsou schopni s vývojáři produktivně spolupracovat. Najímání nevývojářů má dva dobré důvody: zaprvé, projekt tím neochuzujete o dobu vývoje; zadruhé,

osoby, které mají k softwaru nejblíže, jsou zpravidla nevhodnými kandidáty na pořizování dokumentace nebo zkoumání použitelnosti softwaru, neboť se jen obtížně dokážou dívat na software z nezasvěceného pohledu.

Ať bude na těchto problémech pracovat kdokoli, vždy bude muset komunikovat s vývojáři. Najděte si lidi, kteří jsou dostatečně technicky zdatní na to, aby mohli hovořit s vývojářským týmem, ale na druhou stranu nebyli v oblasti softwaru takovými experty, aby se nedokázali vcítit do role běžného uživatele. Středně pokročilý uživatel je pravděpodobně tou správnou osobou k pořizování dokumentace. Po prvním vydání této knihy jsem od pana Dirka Reinerse, vývojáře open source softwaru, obdržel následující e-mail:

```
Jedna poznámka ke kapitole Peníze – Dokumentace a použitelnost: pokud jsme měli nějaké
peníze nazbyt a shodli jsme se, že nejkritičtějším problémem, který potřebujeme vyřešit,
je výukový tutoriál pro začátečníky, najali jsme k napsání tohoto tutoriálu středně
pokročilého uživatele. Do systému byl zasvěcen dosti nedávno, takže si jeho problémy
pamatoval, ale dostal se přes ně, takže věděl, jak je výstižně popsat. To mu umožnilo
napsat program, který sice potřeboval pár drobných oprav provedených hlavními vývojáři
(jednalo se o věci, které původně špatně pochopil), ale jinak pokrýval všechny
„viditelné“ problémy, které by vývojáři opomněli.

V případě tohoto uživatele se navíc jednalo o šťastnou shodu okolností, neboť jeho běžnou
pracovní náplní bylo uvádění různých lidí (studentů) do systému, takže v sobě kombinoval
zkušenosti mnoha lidí, což je jistě vzácná náhoda.
```

Poskytování hostingu/síťové propustnosti

U projektu, který nevyužívá některou z bezplatných kompletních hostingů (viz **Kompletní hosting** v kapitole **3. Technická infrastruktura**), může být poskytnutí serveru a síťového připojení—a především asistence s administrací systému—významnou pomocí. I kdyby vaše organizace neudělala pro projekt už nic dalšího, jsou i tato opatření dosti efektivní cestou ke kladné odezvě na poli public relations, ačkoliv vám nezajistí žádný vliv na směřování projektu.

Pravděpodobně se vaše společnost dočká reklamního banneru nebo poděkování za poskytnutí hostingových služeb na domovské stránce projektu. Pokud hosting nastavíte tak, aby byla internetová adresa projektu pod názvem domény vaší společnosti, získáte další dodatečné propojení s projektem prostřednictvím URL. Díky tomu si bude většina uživatelů myslet, že software má *něco* do činění s vaší společností, přestože do vlastního vývoje softwaru nijak nepřispějete. Problém je, že i vývojáři o tomto asociativním efektu vědí a nemusí se jim dvakrát zamlouvat, že by byl projekt uveden na vaší doméně, pokud do něj nepřispějete nějakým dalším zdrojem kromě síťové propustnosti. Koneckonců, míst k hostování je dnes více než dost. Komunita může nakonec dospět k závěru, že hrozba nesprávně asociovaných zásluhy o projekt nestojí za komfort, který váš hosting nabízí, a přesune projekt jinam.

Takže, pokud chcete poskytnout hostingové služby, klidně tak učiňte,—ale buď si naplánujte, že se budete v projektu co nejdříve více angažovat, nebo velice obezřetně inzerujte svou skutečnou zaangažovanost v projektu.

Marketing

Marketing skutečně funguje, přes pravděpodobně velkou nelibost většiny open source vývojářů. Správná marketingová kampaň *dokáže* kolem open source produktu vytvořit takovou atmosféru, že si i tvrdošíjní programátoři najednou uvědomí, že je daný software oslovuje něčím, co lze jen stěží popsat. Nejsem tu od toho, abych kriticky rozebíral dynamiku závodů ve zbrojení, která je obecně marketingu vlastní. Každá firma podílející se na svobodném softwaru začne nakonec uvažovat o vlastním tržním využití, o tržním využití softwaru nebo svého vztahu k softwaru. Následující rada by vám měla pomoci vyhnout se všem běžným nástrahám, jež jsou s tímto úsilím spojeny; viz také část **Publicita** v kapitole **6. Komunikace.**

Pamatujte na to, že jste sledováni

Chcete-li si udržet komunitu dobrovolných vývojářů na své straně, je velice důležité neříkat nic, co by bylo prokazatelně nepravdivé. Všechna svá prohlášení si důkladně prověřte, dříve než je učiníte, a veřejnosti dejte možnost si tato prohlášení samostatně ověřit. Nezávislá kontrola faktů je hlavní částí open source projektu a týká se nejen kódu.

Samozřejmě by nikdo žádné firmě neradil, aby vydávala neověřitelná prohlášení. Ovšem kolem open source činností se pohybuje neobvykle velké množství lidí dostatečně kvalifikovaných na to, aby dokázali ověřit vydávaná prohlášení—, lidí, kteří pravděpodobně mají k dispozici vysokorychlostní internetové připojení a správné společenské kontakty, aby svá zjištění mohli škodlivým způsobem publikovat. Když společnost Global Megacorp Chemical Industries znečistí vodní tok, mohou tento ověřitelný fakt prokázat pouze zkušení vědečtí pracovníci, jejich tvrzení však mohou následně vyvrátit vědečtí pracovníci z Global Megacorp, ve výsledku si veřejnost bude jen drbat hlavu, aniž by tušila, co si o celém případu myslet. Naopak vaše chování v prostředí open source není jen viditelné a zaznamenávané; mnoho lidí je schopno vaše chování nezávisle kontrolovat, utvořit si o něm vlastní závěry a ty pak dále zcela neformálně šířit. Takovéto komunikační sítě již existují; jsou vlastní esencí fungování open source projektů a lze je použít k přenosu informací jakéhokoli druhu. Vyvrátit takové informace je zpravidla velice obtížné, ne-li nemožné, zvláště, pokud jsou pravdivé.

Například: je zcela v pořádku prohlásit, že vaše organizace „založila projekt X“, pokud tak skutečně učinila. Ale neoznačujte se za „tvůrce projektu X“, pokud větší část kódu vytvořili externí pracovníci. Naopak, netvrďte, že využíváte komunitu dobrovolných vývojářů, která se na projektu intenzivně podílí, pokud se kdokoli může podívat do úložiště a tam zjistit, že změny kódu jen sporadicky nebo vůbec nepodávají lidé mimo vaši organizaci.

Docela nedávno jsem narazil na oznámení jedné velice známé počítačové firmy, která uváděla, že svůj důležitý softwarový balíček vydává pod open source licencí. Po vydání tohoto prvotního oznámení jsem se podíval na jejich, dnes již veřejné, úložiště správy verzí a zjistil, že obsahuje pouhé tři revize. Jinými slovy, provedli úvodní import zdrojového kódu a od té doby se již nic vážného nestalo. To by samo o sobě ještě nebylo znepokojující—, prostě vydali oznámení, nic víc. Od té chvíle nebyl důvod očekávat jakoukoli intenzivnější vývojovou činnost.

Po nějaké době však tatáž společnost vydala další oznámení. Oznámení cituji s pozměněným názvy a číslem verze:

> *S radostí oznamujeme, že po provedení náročných testů komunitou Singer Community je Singer 5 pro Linux a Windows připraveno k produktivnímu využívání.*

Velmi mě zajímalo, co komunita svými „náročnými testy“ odhalila, opět jsem se podíval do úložiště na poslední historii změn. A ejhle, projekt byl stále na úrovni revize 3. Podle všeho tedy nenašli jedinou chybu, kterou by bylo nutno před oficiálním vydáním produktu opravit! Myslel jsem si, že výsledky testování komunitou budou zaznamenány někde jinde, tak jsem nahlédl do bug trackeru. A v něm jsem nalezl přesně šest otevřených problémů, z nichž čtyři byly otevřeny již několik měsíců.

Nevěřil jsem vlastním očím. Když testovací tým zkoumá po určitou dobu rozsáhlý a složitý software, vždycky najde nějaké chyby. I kdyby opravy těchto chyb nestihly být zakomponovány do připravované verze, pořád by se měla očekávat nějaká aktivita kolem správy verzí jako důsledek testování nebo přinejmenším několik nově otevřených problémů. Přesto se podle všeho mezi prvním oznámením o open source licenci a druhým oznámením o vydání prvního open source nic nestalo.

Nejedná se ani tak o to, že společnost lhala o testování prováděném komunitou. Nevím, jestli lhali či nikoli. Ale vůbec nehleděli na to, jak snadno *ze sebe* lháře udělali. Jelikož ani úložiště správy verzí ani issue tracker nenaznačovaly, že by ono inzerované náročné testování někdy proběhlo, společnost buď vůbec neměla žádná taková prohlášení vydávat, nebo měla uvést jasný odkaz na hmatatelné výsledky tohoto testování („Odhalili jsme 278 chyb, pro bližší informace klikněte zde“). Druhá alternativa by každému rychle umožnila udělat si obrázek o skutečné úrovni komunitní činnosti. Takto jsem však během několika minut zjistil, že ať už komunita testovala cokoli, nenechala o tomto testování žádné zprávy na žádném z běžně používaných míst. To není zrovna intenzivní činnost; jsem navíc přesvědčen, že podobný průzkum jsem si neudělal jen já sám.

Transparentnost a ověřitelnost jsou rovněž důležitou součástí uvádění všech spolupracovníků a osob podílejících se na projektu. Další informace viz **Uvádění tvůrců a spoluautorů (Credit)** v kapitole **8. Řízení dobrovolníků.**

Nekritizujte konkurenční open source produkty

Zdržte se negativních komentářů na adresu konkurenčního open source softwaru. Je zcela v pořádku zmínit negativní *fakta*—, tj. snadno potvrditelná tvrzení, která lze často vidět v dobrých srovnávacích tabulkách a grafech. Ale vyhněte se negativním charakteristikám méně exaktní povahy, a to ze dvou důvodů. Zaprvé vedou k rozpoutání plamenných bojů, které vás odvádějí od konstruktivní diskuze. Zadruhé – což je důležitější – může vyjít najevo, že někteří dobrovolní vývojáři ve *vašem* projektu pracují i na konkurenčním projektu. Je to pravděpodobnější, než se na první pohled zdá: projekty jsou ze stejného oboru či tržního segmentu (proto si vzájemně konkurují) a vývojáři s odbornými dovednostmi v tomto oboru mohou přispívat kamkoli, kde jsou jejich odborné dovednosti užitečné a použitelné. I když třeba nedochází k přímému překrývání pracovníků, vývojáři z vašeho projektu se pravděpodobně přinejmenším znají s vývojáři z jiných souvisejících projektů. Jejich schopnost uchovávat konstruktivní osobní vazby může být narušena nadměrně negativními marketingovými zprávami.

Kritizování konkurenčních closed-source produktů je ve světě open source projektů častěji akceptováno, zvláště pokud se jedná o produkty Microsoft. Osobně nejsem z tohoto přístupu příliš nadšen (ačkoliv bych chtěl znovu zdůraznit, že přímočaré faktické porovnávání je zcela v pořádku), a to nikoli jen proto, že je příkladem primitivní hrubosti, ale také proto, že projektu hrozí nebezpečí, že začne věřit vlastním reklamním trikům, a ignorovat tak aspekty, v nichž může být konkurenční výrobek skutečně lepší. Obecně platí, že si musíte dávat pozor na účinky, které mohou mít marketingová prohlášení na vaši vlastní komunitu vývojářů. Lidé mohou být marketingovou a finanční podporou, které se jim dostává, natolik nabuzení, že ztratí objektivní náhled na skutečné silné a slabé stránky vlastního softwaru. Je zcela běžné, dokonce se předpokládá, že vývojáři určité společnosti vystupují s jistým despektem vůči marketingovým prohlášením, a to i na veřejných fórech. Samozřejmě, že nemohou jen tak vystoupit a přímo popírat marketingové zprávy (pokud nejsou skutečně chybné, ačkoliv takovéto záležitosti by snad měly být vyřešeny podstatně dřív). Mohou si z nich ale někdy utahovat, a držet tak ostatní členy vývojářské komunity pěkně při zemi.

6. Komunikace

6. Komunikace — 151

Jste to, co píšete — 152
Struktura a formátování — **152**
Obsah — **154**
Tón — **155**
Jak rozeznat hrubost — **156**
Tvář — **158**

Jak předejít častým potížím — 160
Nepřispívejte zbytečně — **160**
Produktivní a neproduktivní vlákna — **161**
Čím jednodušší téma, tím delší debata — **163**
Vyvarujte se svatých válek — **164**
Efekt „hlučné minority“ — **166**

Obtížní lidé — 166
Jak se s obtížnými lidmi vypořádat — **167**
Případová studie — **168**

Jak se vyrovnat s růstem — 170
Nápadné využívání archivů — **171**
Se všemi zdroji zacházejte jako s archivy — **173**
Kodifikace tradic — **174**

Nediskutujte v bug trackeru — 177
Publicita — **179**
Ohlášení bezpečnostních chyb — **181**
Přijměte report — **181**
Opravu vyvíjejte potichu — **182**
Čísla CAN/CVE — **183**
Předběžné oznámení — **184**
Distribuujte opravu veřejně — **186**

6. Komunikace

Schopnost psát srozumitelně je možná tou vůbec nejdůležitější, jakou může člověk v open sourcovém prostředí mít. Z dlouhodobé perspektivy je důležitější než programovací talent. Skvělý programátor, který neumí komunikovat s okolím, dokáže vždy dělat věci jen postupně, jednu po druhé, a i tak může mít potíže přesvědčit ostatní, aby mu věnovali pozornost. Ale mizerný programátor s výbornými komunikačními schopnostmi dokáže koordinovat a přesvědčit mnoho lidí, aby dělalo mnoho různých věcí, což může mít výrazný dopad na směřování a setrvačnost projektu.

Nezdá se, že by spolu schopnosti psát dobrý kód a komunikovat s ostatními lidmi nijak zvlášť souvisely. Existuje vztah mezi tím dobře programovat a dobře popisovat technické problémy, ale popisování technických problémů tvoří jen drobnou část komunikace v celém projektu. Mnohem důležitější je schopnost soucítit s publikem, umět vnímat své příspěvky a komentáře tak, jak je vidí ostatní, a přimět okolí, aby své vlastní příspěvky dokázalo brát podobně objektivně. Zrovna tak důležité je povšimnout si, že nějaké médium nebo komunikační kanál přestal plnit svou funkci, třeba proto, že nedokázal zvládnout rostoucí počet uživatelů, a najít si čas na to s tím něco udělat.

Tohle vše je teoreticky dost jednoduché; důvod, proč je to v praxi výrazně složitější, je ten, že prostředí, v nichž je svobodný software vyvíjen, jsou ohromně různorodá, jak co se týče jejich cílového publika, tak jejich komunikačních mechanismů. Měla by nějaká myšlenka být vyjádřena příspěvkem v mailing listu, poznámkou v systému sledování chyb nebo komentářem v kódu? Když odpovídáte na nějakou otázku na veřejném fóru, jakou úroveň znalostí můžete předpokládat u čtenáře, vzhledem k tomu, že čtenářem zde není jen ten, kdo otázku položil, ale každý, kdo může na vaši odpověď narazit? Jak mohou vývojáři udržet konstruktivní kontakt s uživateli, aniž by byli zavaleni žádostmi o nové funkce, falešnými bug reporty a záplavou irelevantní komunikace? Jak poznat, že určité médium dosáhlo limitu své kapacity, a co s tím udělat?

Řešení těchto problémů jsou obvykle jen částečná, neboť každé konkrétní řešení časem zastará kvůli růstu projektu nebo změnám jeho struktury. Jsou to také často řešení *ad hoc*, pouze improvizované reakce na dynamické situace. Všichni účastníci si musí být vědomi toho, kdy a jak nastává situace, v níž je komunikační kanál přetížen, a zapojit se do řešení. Pomoc v těchto situacích je pak významnou součástí spravování open source projektu. V následujících oddílech se podíváme na to, jak řídit vlastní komunikaci, i na to, jak zajistit, aby bylo udržování komunikačních mechanismů ve funkčním stavu pro všechny účastníky projektu prioritou.[22]

---

[22] Na tohle téma existuje i celkem pozoruhodný vědecký výzkum, například v článku *Group Awareness in Distributed Software Development* (Povědomí skupiny v distribuovaném vývoji softwaru) od Gutwina, Pennera a Schneidera. Tento článek byl po nějakou dobu dostupný online, pak zase ne a pak zase ano na adrese **http://www.st.cs.uni-sb.de/edu/empirical-se/2006/PDFs/gutwin04.pdf**. Zkuste jej tedy nejprve hledat tam, ale buďte připraveni na to, že se mohl mezitím přesunout někam jinam a budete muset použít vyhledávač.

Jste to, co píšete

Uvědomte si, že jediné, co o vás lidé na internetu vědí, pochází z toho, co píšete a co ostatní píšou o vás. V osobním vztahu můžete klidně být pohotoví, vnímaví a charismatičtí, ale pokud jsou vaše e-maily roztěkané a postrádají strukturu, budou všichni předpokládat, že takoví jste i vy. Zrovna tak můžete být ve skutečnosti velmi roztěkaný člověk, jehož mluvený projev postrádá strukturu, ale nikdo se to nemusí dozvědět, protože vaše příspěvky na internetu jsou srozumitelné a přínosné.

Pokud budete svému psaní věnovat trochu péče, bohatě se vám to vrátí. Ostřílený hacker svobodného softwaru Jim Blandy jednou vyprávěl následující historku:

> V roce 1993 jsem pracoval pro Free Software Foundation a betatestovali jsme verzi 19 GNU Emacs. Zhruba jednou týdně jsme vydávali novou betaverzi, kterou lidé zkoušeli a posílali nám hlášení chyb. Ve skupině byl jeden člověk, kterého jsme osobně nikdo neznali, ale který odváděl skvělou práci – jeho bug reporty byly vždy výborně srozumitelné, naváděly nás přímo na problém a když sám provedl opravu, téměř vždycky byla bezchybná. Opravdu prvotřídní spolupracovník.
>
> V FSF to je tak, že než můžeme začít používat kód, který napsal někdo jiný, musíme mít na papíře potvrzené, že daná osoba svá autorská práva k tomu kódu převádí na FSF. Když jenom popadnete kus kódu od úplně neznámých lidí a vložíte ho do projektu, říkáte si o právní průšvih.
>
> Takže jsem mu poslal příslušné formuláře e-mailem, v němž jsem napsal: „Tady je pár papírů, které potřebujeme vyplnit, a znamená to asi tohle. Tenhle podepíšete vy, tenhle váš zaměstnavatel a pak už můžeme začít používat vaše opravy. Díky moc.“
> Poslal mi zpátky zprávu, v níž stálo: „Já ale nemám zaměstnavatele“.
> Tak jsem napsal, „To se nic neděje, tak ať to podepíše univerzita.“
> Po nějaké době mi odpověděl: „Víte, ono totiž… je mi třináct a bydlím u rodičů.“

Protože ten kluk nepsal, jako by mu bylo třináct, nikoho to ani nenapadlo. Podíváme se teď na několik způsobů, jak i váš psaný projev může dobře zapůsobit.

Struktura a formátování

Nenechte se vlákat do pasti psát všechno, jako by to byly SMSky. Pište celé věty s velkými písmeny na začátku a používejte odstavce tam, kde je to nutné. To je v e-mailech a dalších komponovaných textech nejdůležitější. Na IRC nebo podobných fórech krátkého trvání není celkem problém, když se psaní velkých písmen vynechává, používají se různé zkratky atd. Nedovolte ale, aby se tyto vaše návyky projevily i na formálnějších, trvalejších fórech. E-maily, dokumentace, hlášení chyb a další texty, které mají být trvalé, by měly být psány podle standardní gramatiky a pravopisu a mít souvislou

strukturu. To ne proto, že by se slušelo dodržovat nějaká arbitrárně nastavená pravidla, ale proto, že tato pravidla ve skutečnosti arbitrární vůbec nejsou – do své stávající podoby se vyvinula proto, že díky nim je výsledný text lépe čitelný, což je také přesně ten důvod, proč byste se jich měli držet i vy. Psaní dobře čitelného textu je důležité nejen proto, že pak lidé pochopí, co se jim snažíte říct, ale také proto, že pak působíte jako osoba, která si dává záležet na tom, aby komunikovala srozumitelně, tedy jinými slovy osoba, které stojí za to věnovat pozornost.

Zejména při psaní e-mailů dospěli časem zkušení open source vývojáři k určitým nepsaným pravidlům:

E-maily posílejte ve formátu prostého textu a ne HTML, RichText nebo jiných, s nimiž mohou mít některé e-mailové čtečky problém. Své řádky zalamujte po zhruba 72 sloupcích. Nepřekračujte limit 80 sloupců, jenž se stal jakýmsi *de facto* standardem pro šířku terminálu (tzn. někteří lidé sice používají terminály širší, ale nikdo nemá užší). Tím, že budete držet své řádky o něco kratší než 80 sloupců, necháte dost místa pro několik úrovní citačních značek, takže na váš příspěvek bude moct někdo reagovat, aniž by se musel celý text přelámat.

Použijte skutečné konce řádků. Některé e-mailové klienty provádějí falešné zalamování řádků, kdy vám při psaní e-mailu ukazují zalomení, která tam ve skutečnosti nejsou. Když je pak e-mail odeslán, nemusí mít konce řádků tam, kde jste si mysleli, a na některých obrazovkách se pak bude zalamovat zcela podivně. Pokud váš e-mailový klient falešné zalamování řádků používá, podívejte se po nastavení, kterým by se tato funkce dala vypnout, aby vám ukazoval opravdová zalomení rovnou při psaní.

Pokud do textu vkládáte výstup nějakého programu, kusy kódu nebo nějaký jiný předformátovaný text, viditelně jej oddělte, aby bylo i letmým pohledem vidět, kde končí váš text a začíná citace. (Když jsem začal psát tuhle knihu, nečekal jsem, že bych tuhle radu kdy psal, ale v poslední době jsem na mnoha open source mailing listech viděl přispěvatele směšovat texty z různých zdrojů, aniž by se dalo poznat, který patří kam. Výsledný efekt je hrozně frustrující. Celý příspěvek je pak mnohem hůř srozumitelný a nezbavíte se pocitu, že jeho autor bude asi poněkud neorganizovaný člověk.)

Když citujete e-mail někoho jiného, vkládejte své odpovědi tam, kam se nejvíc hodí, klidně do různých míst, a vymažte ty části e-mailu, které jste nepoužili. Pokud píšete jen rychlou reakci, která se vztahuje k celému příspěvku, můžete klidně *psát nahoru*, tedy nad text e-mailu, který citujete; v ostatních případech byste ale měli nejdřív citovat relevantní pasáž původního textu a až pak přidat svou odpověď.

Věnujte pozornost tomu, co píšete do předmětu nových e-mailů. Předmět je tím vůbec nejdůležitějším řádkem v celém e-mailu, neboť umožňuje ostatním účastníkům projektu rozhodnout, zda budou číst dál nebo ne. Moderní software pro čtení e-mailů sdružuje skupiny souvisejících zpráv do vláken, která lze definovat na základě společného předmětu nebo různých jiných hlaviček (které mohou být i skryté). Z toho plyne, že pokud se nějaké vlákno začne vzdalovat od původního tématu, můžete – a měli byste – podle toho při psaní své odpovědi předmět upravit. Zachováte tím soudržnost celého vlákna, díky těm ostatním hlavičkám, ale zároveň pomůže nový předmět těm, kdo se dívají na přehled vlákna, zjistit, že se téma debaty změnilo. Podobně pokud opravdu chcete založit nové téma, začněte novým

e-mailem a ne tím, že odpovíte na nějaký starší a změníte předmět. V takovém případě by váš e-mail zůstal zařazen do stejného vlákna jako ten, na nějž odpovídáte, a vyvolával by dojem, že je o něčem, o čem není. Ve výsledku taková chyba znamená, že čtenáři ztrácejí čas a vy zase část své reputace jako někdo, kdo umí používat komunikační nástroje.

Obsah

Dobře naformátovaný e-mail je to, co čtenáře přiláká, ale to, co je udrží, je obsah. Samozřejmě neexistuje soubor pravidel, který by zajistil dobrý obsah, ale existuje několik principů, které mohou výrazně pomoct.

Ulehčete svým čtenářům pochopení. Každý aktivní open source projekt obsahuje celou hromadu informací a nemůžete očekávat, že vaši čtenáři budou většinu z nich znát – a dokonce ani to, že budou vědět, jak nějakou věc zjistit. Kdykoliv je to možné, měl by váš příspěvek podávat informace formou, která je pro vaše čtenáře nejpohodlnější. Pokud budete muset strávit dvě minuty navíc tím, že dohledáte URL konkrétního vlákna v archivech mailing listu, aby jej nemuseli hledat vaši čtenáři, udělejte to. Pokud budete muset strávit 5 nebo 10 minut navíc tím, že shrnete dosavadní závěry dosažené ve složitém vláknu, aby lidé pochopili kontext vašeho příspěvku, udělejte to. Přistupujte k věci takto: čím úspěšnější projekt je, tím větší je v každém jeho fóru poměr čtenářů k přispěvatelům. Pokud každý váš příspěvek vidí *n* lidí, tak čím vyšší je *n*, tím větší význam má snaha ušetřit těmto lidem čas. Navíc když ostatní uvidí, že tento standard dodržujete, pokusí se ve vlastní komunikaci postupovat stejně. Ideálním výsledkem je pak nárůst celkové efektivity projektu – pokud existuje možnost, že něco udělá buď n osob nebo jen jeden člověk, pak je pro projekt lepší to druhé.

Nepřehánějte. Na internetových fórech to mívá za následek hotové závody ve zbrojení. Například pokud se někdo, kdo chce nahlásit chybu, obává, že mu nebudou vývojáři věnovat dostatečnou pozornost, popíše ji jako závažný, kritický problém, který zabraňuje jemu (a také všem jeho přátelům / kolegům / bratrancům a sestřenicím) v produktivním používání softwaru, třebaže je to ve skutečnosti jen otravná maličkost. Jenže přehánění není jen zbraní uživatelů – programátoři dělají během technických debat často to samé, zejména když se prodiskutovává něco, co je víc záležitostí vkusu než správného postupu:

> „Když to uděláme takhle, bude celý kód naprosto nečitelný. Hledat v tom chyby by bylo k zešílení. Ale to, co navrhuje J. Novák…“

Pokud tu samou věc zkusíme vyjádřit méně ostře, bude to působit kupodivu silněji:

> „To by sice šlo, ale myslím, že co se týče čitelnosti a údržby, je to velmi nevhodné řešení. Návrh J. Nováka tyhle problémy nemá, protože…“

Nadsázky se nikdy úplně nezbavíte a většinou to ani není nutné. Na rozdíl od jiných forem nedorozumění nezpůsobuje nadsázka projektu jako takovému žádnou škodu – ubližuje převážně tomu, kdo se jí dopustil. Příjemci se s tím vyrovnají, ale odesílatel zprávy s každým použitím ztrácí trochu důvěryhodnosti. V zájmu vlastního vlivu na projekt tedy buďte spíše střídmější. Protože když pak budete opravdu potřebovat něco zdůraznit, budou vás lidé brát vážně.

Všechno upravujte dvakrát. Každou zprávu, která je delší než průměrný odstavec, si před odesláním (ale poté, co máte pocit, že je hotová) znovu přečtěte odshora až dolů. Tohle je rada, kterou slyšeli všichni, kdo někdy psali nějaké školní cvičení, ale v online diskuzích je zvlášť důležitá. Protože proces psaní online zprávy je obvykle velmi přerušovaný (při psaní si často potřebujete něco ověřit ve starších mailech, podívat se na pár webových stránek, spustit příkaz, abyste získali jeho výstup pro debugging atd.), je velmi snadné ztratit nit. To, že byla nějaká zpráva psána nesouvisle a před odesláním nijak neupravována, je obvykle hodně znát a její autoři se za ni pak akorát stydí (tedy, doufejme). Vyhraďte si čas na to přečíst, co jste napsali. Čím více bude struktura vašich příspěvků držet pohromadě, tím více lidí si je přečte.

Tón

Až budete mít za sebou pár tisíc e-mailů, pravděpodobně zjistíte, že máte tendenci psát poněkud stručně. Na většině technických fór tomu tak bývá a samo o sobě na tom nic špatného není. Určitá strohost, která by byla v normálních sociálních situacích nepřijatelná, je mezi hackery svobodného softwaru zkrátka normální. Tady je pro ilustraci úplná citace jedné odpovědi, kterou jsem kdysi dostal na jednom mailing listu o svobodném softwaru pro správu obsahu:

```
Můžete popsat trochu přesněji, na jaký problém jste narazil atd.?

A jakou verzi Slashe používáte? Z vaší zprávy jsem to nepoznal.

Jak přesně jste zkompiloval zdroj apache/mod_perl?

Zkoušel jste záplatu Apache 2.0, která je na slashcode.com?

  Shane
```

Tomu říkám stručnost! Žádný pozdrav, místo rozloučení pouze jméno a celá zpráva je jen řadou otázek, které jsou tak strohé, jak to jen jde. Jediná oznamovací věta, která tam je, implicitně kritizuje mou původní zprávu. A přesto jsem měl radost, že mi tenhle e-mail od Shanea přišel, a jeho stručnost jsem nebral jako známku ničeho jiného, než že je to velmi vytížený člověk. Už jenom to, že se zeptal, místo aby mě úplně ignoroval, znamená, že byl ochoten mému problému věnovat svůj čas.

Budou na takovýhle styl reagovat všichni čtenáři pozitivně? Ne nutně. To záleží na osobě a na kontextu. Například pokud někdo právě poslal příspěvek, v němž uznal svůj omyl (třeba napsal kód s chybou), a vy z předchozí zkušenosti víte, že je to člověk, jenž má trochu nižší sebevědomí, pak je docela dobře možné napsat odpověď, která je sice stručná, ale bere trochu ohled na jeho city. Většinu zprávy může zabírat věcná analýza situace, tak strohá, jak se vám zlíbí. Měla by ale být ukončena něčím, co ukáže, že vaše stručnost neznamená chlad. Například pokud jste právě poskytli hromadu rad o tom, jak by se daná chyba přesně měla opravit, dodejte na závěr „Hodně štěstí, <vaše jméno>," abyste ukázali, že této osobě přejete úspěch a že se na ni nezlobíte. Vhodně umístěný smajlík nebo nějaká jiný emotikon mohou také adresátovi pomoct zbavit se pocitu viny.

Může vám připadat divné brát stejný ohled na něčí city jako na to, co říkají, ale pokud to mám říct natvrdo, city mají vliv na produktivitu. Jsou samozřejmě důležité i z jiných důvodů, ale i pokud se budeme dívat na věc z čistě praktického hlediska, zjistíme, že nešťastní lidé píšou horší software a píšou ho méně. Vzhledem k tomu, jaká omezení většina elektronických médií má, vám ale často bude chybět nějaký indikátor toho, jak se daná osoba cítí. Budete to muset odhadnout na základě toho a) jak by se většina osob v dané situaci cítila a b) co víte o tomto konkrétním jednotlivci na základě předchozích zkušeností. Někteří lidé si raději celou věcí nelámou hlavu a se všemi jednají zcela bez obalu; jejich úvaha je asi taková, že pokud někdo otevřeně neřekne, jak se cítí, pak není důvod se k němu chovat nějak jinak. Tenhle přístup se mi nezdá z několika důvodů. Za prvé, tohle lidé nedělají ani v reálném životě, tak proč by to měli dělat online? Za druhé, protože většina komunikace probíhá na veřejných fórech, mají lidé většinou tendence vyjadřovat své emoce méně, než by to dělali v soukromí. Přesněji řečeno jsou často ochotni vyjádřit své pocity vůči někomu jinému, jako například vděk nebo rozčilení, ale už ne své pocity vnitřní, jako je nejistota nebo pýcha. Většina lidí ale funguje lépe, když ví, že ti ostatní jsou si jejich rozpoložení vědomi. Když budete pozorně vnímat drobné signály, bude většina vašich odhadů správná, což bude účastníky projektu více motivovat k tomu, aby v něm zůstali.

Tím samozřejmě nechci říct, že je vaší úlohou dělat skupině terapeuta a pomáhat každému jednotlivci pochopit city těch ostatních. Pokud ale budete pečlivě sledovat dlouhodobé vzorce chování lidí v projektu, začnete je víc vnímat jako jednotlivce, i když se s nimi nikdy osobně nesetkáte. A pokud budete vnímat tón, jaký používáte v písemné komunikaci, můžete až překvapivým způsobem ovlivnit to, jak se jiní cítí, což v konečném důsledku projektu jen prospěje.

Jak rozeznat hrubost

Jedním z charakteristických rysů open source kultury je její specifický pohled na to, co je a co není hrubost. Zvyklosti, které popisuji níže, se sice nevztahují jen k vývoji svobodného softwaru ani softwaru obecně – budou povědomé každému, jehož oborem je matematika, přírodní vědy nebo strojírenství –, ale vzhledem k tomu, že hranice společenství svobodného softwaru jsou velmi propustné a příval nováčků je velký, existuje dost velká pravděpodobnost, že se objeví někdo, kdo je nezná.

Začněme tím, co se za hrubost nepovažuje:

Technická kritika, i pokud je přímá a podávaná bez servítků, není hrubá. Může být svým způsobem dokonce lichotivá, neboť implikuje, že autor považuje kritizovaného člověka za někoho, koho stojí za to brát vážně a jemuž stojí za to věnovat čas. Jinak řečeno, čím jednodušší by bylo něčí příspěvek jednoduše ignorovat, tím větší kompliment skládá ten, jenž si vyhradil čas na to, aby jej zkritizoval (samozřejmě jen tehdy, pokud se kritika nepromění v útok ad hominem nebo nějakou jinou vyloženou nezdvořilost).

Věcné, nijak nepřikrášlované otázky, jako jsou ty, jež mi položil Shane v e-mailu, který jsem citoval o něco výš, také nejsou hrubé. Otázky, které by v jiných situacích mohly působit chladně, jako řečnické obraty nebo přímo výsměch, jsou často myšleny zcela vážně a neskrývá se za nimi nic jiného než snaha získat informaci tak rychle, jak je to možné. Slavná otázka technické podpory „Je váš počítač zapojen do sítě?“ je klasickou ukázkou stejného postupu. Technik podpory opravdu potřebuje vědět, zda je počítač zapojený, a za tu dobu, co tuto práci dělá, už jej unavilo před tuto otázku klást zdvořilostní fráze („Promiňte, ale nejdřív vám chci položit pár jednoduchých dotazů, abychom vyloučili některé možnosti. Některé z nich vám asi budou připadat celkem primitivní, ale zkuste to, prosím, vydržet…“). Teď už je ve stavu, kdy se těmito oklikami nezdržuje a zeptá se přímo: je zapojený, nebo není? Podobné otázky se v mailing listech svobodného softwaru objevují prakticky pořád. Jejich cílem není nikoho urazit, ale rychle vyloučit některé možnosti, které se nejvíc nabízejí (a jež jsou možná i nejběžnější). Ti, kteří to pochopí a podle toho budou i reagovat, získávají body navíc za to, že bez vyzvání přistupují k celé věci chápavě. Na druhou stranu by ale nemělo být nikomu bráno za zlé, když na to bude reagovat podrážděně. Není to ničí chyba, ale pouhý střet kultur. Přátelským tónem vysvětlete, že vaše otázka (nebo kritika) neměla žádný skrytý podtext, pouze se snažila získat (nebo předat) informaci co nejefektivnějším způsobem, nic víc.

Co tedy je hrubost?

Na základě stejného principu, podle nějž se dá podrobná technická kritika vnímat jako jistý druh komplimentu, může být neposkytnutí kvalitní kritiky urážející. Tím nemyslím to, že je něčí práce jednoduše ignorována, ať už je to návrh, změna kódu, oznámení chyby nebo cokoliv jiného. Pokud jste předem explicitně neslíbili podrobnou reakci, pak je neposkytnutí žádné odpovědi zcela v pořádku. Lidé si řeknou, že jste zkrátka neměli čas. Pokud se ale rozhodnete reagovat, nesmíte to odbýt. Vyhraďte si čas na to věc skutečně zanalyzovat, poskytnout konkrétní příklady tam, kde je to vhodné, prohledat archivy, abyste našli podobné příspěvky z minula atd. Nebo, pokud na něco takového nemáte čas, ale stejně musíte napsat nějakou reakci, to ve své zprávě otevřeně zmiňte („Myslím, že tahle chyba už byla nahlášena, ale teď nemám čas se po tom podívat, nezlobte se“). Hlavní věcí je uznat, že existuje nějaká kulturní norma, ať už tím, že ji dodržíte, nebo tím, že otevřeně přiznáte, že tentokrát jste jí nevyhověli. Oba způsoby tuto normu posilují. Pokud ale normu nedodržíte a neposkytnete žádné vysvětlení proč, bude to úplně stejné, jako kdybyste řekli, že celé téma (a všichni, kdo se jej účastní) vám za moc času nestojí. To, že je váš čas cenný, lépe ukážete tím, že budete struční, než tím, že budete líní.

Hrubost má samozřejmě mnoho podob, ale většina jich není specifická pro vývoj svobodného softwaru a na to, abyste se jich vyvarovali, bohatě stačí zdravý rozum. Více informací, pokud jste je ještě nečetli, najdete také v části **Nezdvořilé chování potlačte hned v zárodku** v kapitole **2. Zahájení projektu**.

Tvář

V lidském mozku existuje oblast vyhrazená pro rozeznávání tváří. Její přesné označení je „tvářová oblast gyrus fusiformis“ a její schopnosti jsou z velké části vrozené, nikoliv naučené. Zdá se, že rozeznávání jednotlivých osob je natolik důležité pro přežití, že jsme si pro to vyvinuli specializovaný hardware.

Společná činnost na internetu je proto psychologicky poněkud zvláštní, protože při ní úzce spolupracuje skupina lidí, kteří se téměř nikdy neidentifikují těmi nejpřirozenějšími, intuitivními metodami – rozeznávání tváře je nejdůležitější z nich, ale patří sem i hlas, postoj atd. Jako kompenzaci tohoto problému byste měli všude používat stejnou *virtuální identitu*. Ta by měla tvořit první část vaší e-mailové adresy (tedy část před zavináčem), vaše uživatelské jméno na IRC, vaši commiterskou identifikaci v úložišti, uživatelské jméno v issue trackeru a tak dál. Tohle jméno je vaší online „tváří“, krátkým identifikátorem, který slouží podobnému účelu jako vaše opravdová tvář, třebaže, bohužel, neaktivuje stejný hardware v mozku.

Tato identita by měla být nějak intuitivně odvoditelná z vašeho reálného jména (například moje je „kfogel“). V některých situacích ji stejně bude doprovázet vaše plné jméno, například v hlavičkách e-mailů:

```
Od: "Karl Fogel" <kfogel@nejakadomena.com>
```

Tento příklad ale ukazuje ještě něco dalšího. Jak už jsem zmínil, je to identita, která se intuitivně vztahuje k reálnému jménu. A když říkám reálné jméno, myslím skutečně reálné. Není to tedy něco vymyšleného jako například:

```
Od: „Velký hacker" <velkyhacker@nejakadomena.com>
```

Existuje jeden slavný kreslený vtip od Paula Steinera, který otiskl časopis *The New Yorker* ve svém čísle z 5. července 1993, na němž je pes, který je přihlášen k počítačovému terminálu, dívá se dolů na jiného psa a spiklenecky mu říká: „Na internetu nikdo neví, že jsi pes.“ Stejné uvažování pravděpodobně vede k vytváření mnoha z těch bombastických, rádoby drsných online identit, které si lidé vymýšlejí, jako kdyby to, že se někdo pojmenuje „Velký hacker“, znamenalo, že mu někdo uvěří, že opravdu je velký hacker. Ale jedna věc je jistá: i když nikdo neví, že jsi pes, pořád jsi pes. Vybájená online identita na nikoho dojem neudělá. Akorát si pomyslí, že se jen předvádíte, místo toho, abyste něco dělali, nebo že máte problémy se sebevědomím. Ve všech interakcích používejte své pravé jméno; pokud ale z nějakého důvodu chcete zůstat anonymní, tak si vymyslete nějaké, které zní zcela věrohodně, a důsledně jej používejte.

Nejde jen o to, udržet svou online tvář konzistentní, je také možné ji trochu zatraktivnit. Pokud máte nějaký oficiální titul (např. „doktor“, „profesor“, „ředitel“), tak se jím neohánějte; nejlépe jej vůbec nezmiňujte, pokud to není v konverzaci přímo relevantní. Mezi hackery obecně a v kultuře svobodného softwaru zvlášť se předvádění titulů obvykle považuje za elitářství a projev nejistoty. Je v pořádku, pokud se váš titul objevuje ve standardním podpisu na konci všech e-mailů, které posíláte, ale nikdy jej nepoužívejte jako nástroj, jak posílit svou pozici v nějaké diskuzi, protože to zaručeně k ničemu dobrému nepovede. Chcete, aby ostatní respektovali vás a ne váš titul.

A když už mluvíme o automatických podpisech na konci e-mailů, čím menší a vkusnější jsou, tím lépe, a vůbec nejlépe je, když chybí úplně. Vyvarujte se toho na konec každého e-mailu přilepit dlouhé právní pojednání, zejména pokud je jeho obsah v rozporu s účastí na projektu svobodného softwaru. Jako příklad uvedu klasickou ukázku tohoto žánru, která se objevuje na konci každého příspěvku od jistého uživatele na jednom veřejném mailing listu, jehož jsem členem:

```
DŮLEŽITÉ UPOZORNĚNÍ

Pokud jste dostali tento e-mail omylem nebo se chcete seznámit s našim
prohlášením zásad používání a monitorování e-mailů, čtěte prosím dále nebo
se spojte s odesilatelem.

Tato komunikace pochází od firmy Deloitte & Touche LLP. Deloitte &
Touche LLP je komanditní společnost registrovaná v Anglii a Walesu pod
číslem OC303675. Seznam jejích členů je k nahlédnutí v Stonecutter Court, 1
Stonecutter Street, Londýn EC4A 4TR, Velká Británie, hlavní kanceláři a
registrovaném sídle společnosti. Společnost Deloitte & Touche LLP spadá
pod řízení a regulaci finančního úřadu Velké Británie.

Tato komunikace a všechny její případné přílohy obsahují informace, které
jsou důvěrné a mohou být i tajné. Je určena výhradně k užití svého
zamýšleného adresáta nebo adresátů. Pokud nejste zamýšleným adresátem,
upozorňujeme, že všechny formy zveřejnění, šíření, kopírování nebo užití
této komunikace nebo informací obsažených v ní či v jejích přílohách jsou
přísně zakázány a mohou být protizákonné. Pokud jste tuto komunikaci
obdrželi omylem, vraťte ji, prosím, s nadpisem „obdrženo omylem“ na adresu
IT.SECURITY.UK@deloitte.co.uk, tento e-mail vymažte a zničte i všechny jeho
kopie.

U e-mailové komunikace nelze zaručit bezpečnost ani bezchybnost, neboť
informace mohou být zachyceny, poškozeny, upraveny, ztraceny, zničeny, mohou
dorazit se zpožděním nebo neúplné nebo obsahovat viry. Za tyto a podobné
události a jejich následky nepřijímáme zodpovědnost. Každý, kdo s námi
komunikuje prostřednictvím e-mailu, touto činností vyjadřuje, že přijímá
všechna rizika s ní spojená.
```

```
Názory a rady obsažené v tomto e-mailu a jeho přílohách, pokud jsou adresáty
naši klienti, se řídí podmínkami užití, které jsou popsány v příslušném
dopise, jímž Deloitte & Touche LLP zahájila s klientem spolupráci.

Názory, závěry a další informace obsažené v tomto e-mailu a jeho přílohách,
které se nevztahují k oficiální obchodní činnosti společnosti, nejsou touto
společností vydané ani schválené.
```

Pokud se tento špalek textu objevuje u někoho, kdo se objevuje jen čas od času, aby se na něco zeptal, vypadá to trochu hloupě, ale žádné velké škody to nezpůsobí. Kdyby se ale tato osoba chtěla projektu aktivně účastnit, začalo by tohle právní brnění postupně vytvářet potíže. Vysílá hned dva potenciálně škodlivé signály najednou: za prvé, že se jedná o člověka, který nemá kontrolu nad svými nástroji – je uvězněn v nějakém korporátním e-mailovém serveru, který na konec každé zprávy přilepí tuhle otravnou doložku, a nemá způsob, jak to obejít –, a za druhé, že pro své činnosti vztahující se ke svobodnému softwaru nemá žádnou organizační podporu. Je sice pravda, že jeho organizace mu zasílání příspěvků na veřejné mailing listy vyloženě nezakázala, ale rozhodně se nezdá, že by je vítala, jako by riziko toho, že mohou uniknout důvěrné informace, přebíjelo všechno ostatní.

Pokud pracujete pro organizaci, která trvá na tom, aby k veškeré odchozí poště připojovala podobné dodatky, zvažte, zda byste si neměli zdarma zařídit e-mailovou adresu například u **gmail.google.com**, **www.hotmail.com** nebo **www.yahoo.com** a používat ji pro potřeby projektu.

Jak předejít častým potížím

Nepřispívejte zbytečně

Častou chybou, kterou lidé u projektu vedeného online dělají, je to, že se domnívají, že musí odpovídat na všechno. To ale není pravda. Jednak je běžné, že diskuze probíhá ve více vláknech, než stihnete sledovat, zejména u projektů, které jsou starší než několik měsíců; kromě toho i ve vláknech, v nichž se rozhodnete reagovat, platí, že značná část toho, co lidé píšou, odpověď nevyžaduje. Zejména na vývojářských fórech obvykle dominují tři typy zpráv:

1. Příspěvky, které navrhují něco netriviálního
2. Příspěvky, které vyjadřují souhlas nebo nesouhlas s tím, co napsal někdo jiný
3. Shrnující příspěvky

Ani jeden z těchto tří typů sám o sobě odpověď nevyžaduje, zvlášť pokud si můžete být na základě toho, co ve vláknu dosud zaznělo, celkem jisti, že někdo jiný pravděpodobně řekne přesně to, co byste řekli i vy. (Obava, že se takhle chytíte do nekonečné smyčky, protože stejnou taktiku budou používat i všichni ostatní, je celkem zbytečná. Téměř vždy se najde někdo, kdo se do příslušné debaty rád pustí.) Vaše reakce by měla být motivována nějakým konkrétním cílem. Zeptejte se nejprve sami sebe: vím, čeho chci dosáhnout? A za druhé: určitě se toho nedosáhne, bez mého vyjádření?

Dva dobré důvody proč se přidat do diskuze jsou a) když vidíte chybu v něčím návrhu a máte podezření, že jste jediný, kdo si jí všiml, a b) když vidíte, že mezi ostatními dochází k nedorozumění, které můžete objasnit. Obecně je také v pořádku zasílat příspěvky, jimiž někomu děkujete, nebo které vyjadřují jen rychlou souhlasnou reakci, protože u nich čtenář hned pozná, že nevyžadují odpověď ani další činnost, takže jasně ví, že po dočtení už jim nemusí věnovat další pozornost. I přesto si ale rozmyslete, než něco řeknete. Vždy je lepší, když si budou ostatní přát, abyste přispívali častěji, než aby doufali, že začnete přispívat méně. (V druhé polovině přílohy **C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?** najdete pár dalších rad o tom, jak se chovat ve velmi živém mailing listu.)

Produktivní a neproduktivní vlákna

V mailing listech s velkým množstvím příspěvků existují dvě priority. Jednou z nich je samozřejmě rozpoznat, co vyžaduje vaši pozornost a co můžete ignorovat. Tou druhou je chovat se tak, abyste sami k vzniku šumu nepřispívali. Nechcete jen, aby vaše vlastní e-maily měly velký poměr signál–šum, ale také aby fungovaly jako výzva ostatním: buď pište zprávy, které mají podíl signálu podobně velký, nebo raději nepište vůbec.

Abychom zjistili, jak to udělat, zvažme nejprve kontext, v němž se to odehrává. Jak poznat neproduktivní vlákno?

- Argumenty, které už byly uvedeny, se začnou opakovat, jako by se přispěvatel domníval, že si jich předtím nikdo nevšiml.
- Rostoucí míra nadsázky a osobního zapojení do tématu, třebaže o nic velkého nejde.
- Většina příspěvků pochází od lidí, kteří sami dělají velmi málo nebo nic, zatímco ti, kdo obvykle problémy spíš řeší, zůstávají zticha.
- Prodiskutovává se mnoho nápadů, ale bez jasných konkrétních návrhů. (Jistě, každý zajímavý nápad začíná jako jenom jakási mlhavá vize; to důležité je, jakým směrem se vydá potom. Je ve vláknu vidět proces přetváření této vize do něčeho hmatatelnějšího, nebo se místo toho rozbíhá do menších vizí, vedlejších vizí a ontologických debat?)

To, že nějaké vlákno není od samého začátku produktivní, ještě neznamená, že je to ztráta času. Jeho téma může být důležité; v takovém případě je to, že nijak nepostupuje, o to znepokojivější.

Navést vlákno k tomu, aby začalo být užitečné, aniž by to bylo provedeno násilně, je umění. Nebude fungovat, když všechny okřiknete, ať přestanou ztrácet čas, nebo pokud je požádáte, ať nepřispívají, pokud nemají co konstruktivního říct. To si, samozřejmě, můžete myslet, ale pokud to řeknete nahlas, všechny tím urazíte. Místo toho musíte navrhnout podmínky pro další postup – nabídnout ostatním cestu, jež je navede k výsledkům, které chcete, aniž by to znělo jako diktátorské nařízení. Tím hlavním rozdílem je, jakým tónem to podáte. Tohle je například špatně:

> *Tahle diskuze nikam nevede. Můžete, prosím, téma opustit, dokud někdo nenapíše záplatu, kterou by jeden z těchto návrhů implementoval? Je zbytečné opakovat pořád to samé dokola. Zdrojový kód je důležitější než slova, vážení.*

Tohle je ovšem dobře:

> *V tomhle vláknu se už objevilo několik návrhů, ale žádný z nich nebyl rozpracován podrobněji, nebo přinejmenším ne natolik, aby se o něm dalo konkrétně diskutovat. Taky už jsme ale ve fázi, kdy nic nového neříkáme, jen opakujeme to, co už zaznělo. V tenhle moment by asi bylo nejlepší, kdyby další příspěvky buď obsahovaly úplnou specifikaci navržené funkce, nebo záplatu. Pak bychom aspoň před sebou měli konkrétní problém (tedy buď získat konsenzus ohledně této specifikace nebo aplikovat a otestovat záplatu).*

Srovnejte, jak jinak zní druhý příklad oproti tomu prvnímu. Použitím toho druhého způsobu se nedistancujete od ostatních, ani je neobviňujete z toho, že z diskuze udělali nekonečnou spirálu. Hovoří se v něm o „nás“, což je důležité, ať už jste do vlákna předtím přispěli nebo ne, protože to všem připomene, že i ti, kteří dosud mlčeli, mají zájem na tom, jakým způsobem se vlákno vyvine. Popisuje, proč tohle vlákno k ničemu nevede, aniž by někoho urážel nebo posuzoval – pouze nezaujatě konstatuje fakta. A co je nejdůležitější, navrhuje pozitivní řešení, takže místo toho, abyste v ostatních vyvolali pocit, že diskuzi uzavíráte (a proti tomuto zákazu okamžitě vyvstane pokušení se vzepřít), budou váš příspěvek chápat jako návrh, jak přesunout debatu k něčemu konstruktivnějšímu. A to je standard, s nímž budou všichni souhlasit.

Nebude pokaždé žádoucí, aby se příslušné vlákno stalo konstruktivnějším; někdy budete chtít, aby prostě zmizelo. Cílem vašeho příspěvku je tedy dosáhnout buď jednoho, nebo druhého. Pokud na základě toho, jak se vlákno dosud vyvíjelo, můžete usoudit, že to, co navrhujete, stejně nikdo neudělá, pak stejný příspěvek celé vlákno v podstatě uzavírá, aniž by to vypadalo, že to bylo schválně. Samozřejmě neexistuje žádný neprůstřelný způsob, jak nějaké vlákno uzavřít, a i kdyby existoval, nebylo by dobré jej používat. Ovšem to, že ostatní požádáte, aby buď dospěli k nějakému závěru, nebo přestali diskutovat, je zcela obhajitelné řešení, pokud to uděláte diplomaticky. Dejte si ale pozor na to, abyste nezavírali vlákna předčasně. Určitá míra spekulativního klábosení může být u některých témat produktivní; pokud byste žádali o řešení příliš brzy, mohli byste celý tvořivý proces zadusit a navíc budete vypadat netrpělivě.

Neočekávejte také, že se nějaké vlákno zastaví okamžitě. Po vašem příspěvku se pravděpodobně objeví ještě pár dalších, buď proto, že se e-maily navzájem minuly, nebo proto, že někdo chce mít poslední slovo. Tím se není potřeba rozrušovat a není ani nutné psát do vlákna znovu. Jen jej nechte, ať samo vyhasne, popřípadě nevyhasne. Úplnou kontrolu nad tím mít nebudete nikdy, ale na druhou stranu můžete očekávat, že vaše působení bude mít v statisticky výrazném počtu vláken významné výsledky.

Čím jednodušší téma, tím delší debata

U každé diskuze existuje riziko, že se začne divoce vzdalovat od tématu, nicméně pravděpodobnost takových odboček roste s tím, jak klesá technická náročnost celé debaty. Protože pochopitelně čím je téma technicky náročnější, tím méně lidí dokáže sledovat, o čem se vlastně mluví. Ti, kteří to dokážou, budou obvykle právě ti nejzkušenější vývojáři, kteří už takových debat zažili tisíce a vědí, jak nejlépe dosáhnout konsenzu, s nímž se smíří všichni.

Z toho vyplývá, že konsenzu je nejtěžší dosáhnout u technických problémů, které jsou snadno pochopitelné a na něž je lehké si udělat názor, a u takzvaně „jednodušších" otázek, jako je organizace, publicita, financování atd. Takové debaty mohou probíhat v podstatě donekonečna, protože na to, aby se jich někdo účastnil, nepotřebuje žádnou kvalifikaci, nelze jednoznačně určit (a to ani zpětně), zda je nějaké rozhodnutí dobré nebo špatné, a taky proto, že někdy je úspěšnou taktikou jednoduše vydržet diskutovat déle než všichni ostatní.

Princip, jenž říká, že objem diskuze je nepřímo úměrný složitosti tématu, je známý už dlouho a označuje se neformálním termínem *Efekt přístřešku na kola.* Takto ho jednou vysvětlil Poul-Henning Kamp ve svém dnes slavném příspěvku vývojářům BSD:

> Je to dlouhá historka; tedy spíše stará historka, protože dlouhá moc není. V raných 60. letech napsal C. Northcote Parkinson knihu, která se jmenovala „Parkinsonův zákon", a v ní napsal hodně o dynamice řízení lidí.
>
> [...]
>
> V tom konkrétním příkladu, v němž zmiňuje přístřešek na kola, je tím druhým hlavním komponentem jaderná elektrárna, což docela dobře odráží dobu, kdy ta knížka vznikla.
>
> Parkinson tam ukazuje, jak můžete jít za představenstvem nějaké firmy a získat jejich schválení pro stavbu jaderné elektrárny v hodnotě miliónů, třeba i miliard dolarů, ale pokud navrhnete postavit přístřešek na kola, zamotáte se do nekonečných debat.
>
> Parkinsonovo vysvětlení je, že to je proto, že jaderná elektrárna je tak obrovská, tak drahá a tak složitá, že ji lidé nedokážou úplně pochopit; místo toho, aby se o to pokusili, budou předpokládat, že předtím, než se celá věc dostala až takhle daleko, už všechny důležité detaily zkontroloval někdo jiný. Richard P. Feynman ve svých knihách uvádí několik zajímavých a velmi případných příkladů týkajících se Los Alamos.
>
> Jenže s přístřeškem na kola je to jiné. Něco takového stlučete dohromady za víkend a ještě budete mít čas se večer podívat na fotbal. Takže ať už jste připravení sebelépe, ať už je váš návrh seberozumnější, vždy se najde někdo, kdo se chopí této příležitosti předvést, že dělá svou práci, že věci věnuje pozornost, že u toho je.

> V Dánsku tomu říkáme „nechat na něčem svůj otisk prstu“. Je to věc osobní cti a prestiže; jde o to, moct později na něco ukázat a říct: „Vidíte? To jsem dělal já.“ Tohle je chování velmi typické pro politiky, ale pokud se naskytne vhodná příležitost, pustí se do toho kdekdo. Je to trochu jako otištění vašich stop v ještě mokrém betonu.

(Celý ten příspěvek stojí za přečtení. Viz příloha **C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?** a také **http://bikeshed.com**.)

S tím, co zde Kamp popisuje, se setkal každý, kdo se někdy pravidelně účastnil skupinového rozhodování. Úplně zabránit debatám o tom, jakou barvou ten přístřešek na kolo natřeme, se obvykle ukazuje jako nemožné. To nejlepší, co v takových případech můžete udělat, je upozornit na to, že to je známý fenomén, a přesvědčit ty nejrespektovanější vývojáře, jejichž slova mají největší váhu, aby své kbelíky s barvami někam odložili a alespoň nepřispívali k vzniku dalšího šumu. Skupiny lidí, kteří budou barvu přístřešku na kola řešit dál, nikdy úplně nevymizí, ale pokud budete v kultuře projektu šířit povědomí o tomto efektu, dosáhnete toho, že jich bude méně a budou se ozývat méně často.

Vyvarujte se svatých válek

Svatá válka je debata týkající se často (ale ne vždy) nějaké maličkosti, již není možné vyřešit argumenty, ale kterou lidé vnímají natolik intenzivně, že se o ni budou přít pořád dál a doufat, že jejich strana časem zvítězí. Svaté války nejsou totéž, co natírání přístřešků na kola. Lidé, kteří se rozhodnou natírat přístřešek, obvykle rychle dojdou k nějakému závěru (protože to není těžké), ale jejich názor nemusí být nutně nijak zvlášť silný; často v průběhu debaty přejdou na stranu nějakého zcela odlišnému návrhu, aby ukázali, že rozumí všem aspektům problému. Pokud ale ve svaté válce vyjádříte pochopení té druhé strany, vyjadřujete tím vlastní slabost. Ve svaté válce všichni vědí, že existuje jen jediná správná odpověď. Jenom se nějak nemohou shodnout na tom, která to je.

Jak jednou svatá válka propukne, tak se obvykle nedá vyřešit tak, aby byli všichni spokojení. V jejím průběhu nemá žádný význam poukázat na to, že se jedná o svatou válku. To už všichni vědí. Bohužel společným rysem svatých válek je to, že panuje neshoda ohledně samotné otázky, zda je možné debatu vyřešit prostřednictvím diskuze. Při pohledu zvenčí je jasné, že ani jedna strana tu druhou nepřesvědčí. Při pohledu zevnitř to vypadá tak, že ta druhá strana je strašně tvrdohlavá a odmítá uvažovat o věci správně, ale když vytrvám, možná změní názor. Tím ovšem neříkám, že ve svatých válkách nikdy neexistuje strana, která má pravdu. Někdy se to opravdu stává – ve svatých válkách, jichž jsem se účastnil já, to samozřejmě byla vždycky ta moje. Ale na tom nezáleží, protože stejně neexistuje algoritmus, jakým lze přesvědčivým způsobem prokázat jedné straně, že pravdu má ta druhá.

Běžným, ale neuspokojivým způsobem, jímž se mnozí pokoušejí svaté války dohnat ke smíru, je říct: „Tomuhle už jsme věnovali mnohem víc času a energie, než si to zaslouží. Nemohli bychom toho prostě nechat?“ To má dva problémy. Za prvé, čas a energie už byly vyplýtvány a zpátky se nevrátí – jediná otázka, která zbývá, je, kolik jich ještě bude třeba? Pokud má někdo pocit, že už stačí jen málo a diskuze bude uzavřena, pak pořád má (z jeho úhlu pohledu) smysl v ní pokračovat.

Druhý problém žádosti, aby toho všichni nechali, je v tom, že to často znamená, že té straně, která prosazuje status quo, přiznává vítězství tím, že se nebude dělat nic. V některých případech se navíc obecně uznává, že status quo je neakceptovatelné; všichni souhlasí s tím, že je potřeba rozhodnout a něco udělat. Pokud diskuze přestane, bude to pro všechny horší řešení, než když se někdo vzdá. Ale protože tenhle pocit sdílejí všichni, nikdo to nevzdá a bude se pokračovat do nekonečna.

Jak se tedy se svatými válkami vypořádat?

První odpovědí je, zkuste nastavit věci tak, aby vůbec nevznikaly. To není zas tak marná snaha, jak to na první pohled vypadá:

Některé běžné svaté války lze dobře předvídat, protože se často objevují u témat, jako jsou programovací jazyky, licence (viz **GPL a kompatibilita licence** v kapitole **9. Licence, autorská práva a patenty**), použití hlavičky reply-to (viz **Velká debata o Reply-to** v kapitole **3. Technická infrastruktura**) a pár dalších věcí. Každý projekt pak obvykle má ještě jednu nebo dvě vlastní svaté války, které vývojáři, jež jsou u něj dlouho, už dobře znají. Techniky, jak zastavit svaté války nebo přinejmenším omezit jejich dopad, jsou všude prakticky stejné. I pokud jste přesvědčeni o tom, že vaše strana má pravdu, zkuste najít alespoň nějaký způsob, jak vyjádřit pochopení a sympatie vůči argumentům vaší protistrany. Problémem svatých válek je často to, že každá strana vystaví své obranné zdivo tak vysoko, jak to jen dokáže, a prohlásí, že jakýkoliv jiný názor je čiré bláznovství. Už jen představa toho, že by se někdo vzdal nebo změnil názor, se tak pro něj stává psychologicky neúnosnou: přiznal by tím nejen to, že se mýlil, ale také to, že si byl věcí naprosto jistý a stejně se mýlil. Tohle přiznání se dá trochu ulehčit tím, že sami vyjádříte určitou nejistotu, a to právě tak, že ukážete, že jejich argumentům rozumíte a považujete je za rozumné, i když ne přesvědčivé. Udělejte gesto, jejž vám můžou oplatit svým gestem, a situace se obvykle zlepší. Na pravděpodobnost, že dosáhnete technického výsledku, který chcete, to nebude mít vůbec žádný vliv, ale alespoň omezíte dopad na morálku projektu.

Pokud nelze svaté válce zabránit, rozhodněte se už na začátku, jak moc vám na tom záleží, a buďte připraveni se veřejně vzdát. Když to uděláte, můžete říct, že to je proto, že svatá válka za tu věc nestojí, ale nedělejte to s hořkostí a v žádném případě nevyužívejte této příležitosti pro to, abyste si proti argumentům té druhé strany ještě jednou naposledy šlehli. Odcházení z bojiště má význam jenom tehdy, pokud je provedeno důstojně.

Svaté války týkající se programovacích jazyků jsou trochu zvláštní případ, protože jsou často vysoce technického rázu, ale přesto má mnoho lidí pocit, že jsou kvalifikovaní se vyjádřit, a také proto, že v nich jde o hodně: jejich výsledek může rozhodnout, v jakém jazyce bude psaná velká část zdrojového kódu projektu. Nejlepším řešením je rozhodnout o tom, jaký jazyk bude zvolen, již velmi brzy a po konzultacích s vlivnými vývojáři projektu a toto rozhodnutí potom hájit na základě toho, že v tomto jazyce se vám nejlépe pracuje, a ne proto, že je lepší než nějaký jiný jazyk, který jste mohli použít místo něj. Nikdy nedovolte, aby se konverzace zvrhla v akademické srovnávání programovacích jazyků (jako se z nějakého mně neznámého důvodu velmi často stává, když někdo navrhne Perl) – to je vyloženě smrtící téma, k němuž se musíte odmítnout vyjadřovat.

Více informací o historickém pozadí svatých válek najdete na **http://catb.org/~esr/jargon/html/H/holy-wars.html** a v článku Dannyho Cohena, který tento termín zpopularizoval, **http://www.ietf.org/rfc/ien/ien137.txt**.

Efekt „hlučné minority“

V jakékoliv diskuzi na mailing listech může malá skupina lidí snadno vyvolat dojem, že v projektu bobtná velké množství nespokojenosti, jednoduše tím, že jej zaplaví mnoha dlouhými e-maily. Tato taktika má jisté společné rysy s hrou o čas, ale je ještě účinnější, neboť tento pocit nespokojenosti se vybudovává napříč mnoha jednotlivými příspěvky a málokdo se bude obtěžovat tím, aby sledoval, kdo kdy a co řekl. Všichni ale získají instinktivní pocit, že se jedná o velmi kontroverzní téma, a budou čekat, až nálady trochu opadnou.

Nejlepším způsobem, jak tomuto efektu zamezit, je velmi jednoznačně a s použitím důkazů předvést, jak málo těch nespokojených ve skutečnosti je oproti těm, kdo souhlasí. Abyste ten nepoměr ještě zdůraznili, můžete se soukromě dotázat lidí, kteří spíše mlčeli, ale o nichž si myslíte, že budou na straně většiny. Neříkejte nic, co by naznačovalo, že se odbojná skupinka pokoušela celou věc nafouknout úmyslně. Pravděpodobně to tak nebylo, a i pokud by bylo, pak vám to, že na to upozorníte, žádnou strategickou výhodu nepřinese. Stačí, když ukážete příslušná čísla vedle sebe a každý pochopí, že jejich intuitivní vnímání situace neodpovídá realitě.

Tato rada neplatí jen pro témata, u nichž existují jasné pozice pro a proti. Platí obecně pro všechny diskuze, které jsou velmi divoké, ale v nichž se nezdá, že by jejich téma většina lidí považovala za nějak zásadní. Pokud jste toho názoru, že diskutovanou otázku není potřeba řešit, a po nějaké době jasně uvidíte, že se k debatě nikdo další nepřipojuje (třebaže už vyprodukovala mnoho e-mailů), můžete tohle své pozorování oznámit veřejně. Pokud zde fungoval efekt „hlučné minority“, bude váš příspěvek působit jako závan čerstvého vzduchu. Většina lidí měla totiž dosud z celé věci jen mlhavý pocit: „Zdá se, že tu jde o něco důležitého, protože těch příspěvků rozhodně není málo, ale nevidím, že by se tu k něčemu došlo.“ Objasněním toho, že díky své formě vypadala diskuze mnohem bouřlivěji, než jaká ve skutečnosti byla, jí zpětně přeměníte v něco jiného, díky čemuž bude pro ostatní snadnější pochopit, co se vlastně stalo.

Obtížní lidé

Vypořádat se s obtížnými lidmi není na elektronických fórech o nic lehčí než osobně. Když říkám „obtížní“, nemyslím tím „nezdvořilí“. Nezdvořilí lidé jsou otravní, ale ne nutně obtížní. V této knize už jsme probrali, co s nimi dělat: napoprvé na jejich nezdvořilost upozorněte, ale od té doby je buď ignorujte, nebo s nimi zacházejte jako se všemi ostatními. Pokud budou i nadále hrubí, obvykle se stanou natolik nepopulárními, že nebudou mít v projektu žádný vliv, takže se jako problém odstraní sami.

Skutečně obtížnými případy jsou ti, kteří nejsou otevřeně nezdvořilí, ale kteří manipulují procesy projektu nebo je zneužívají způsobem, jenž ostatní stojí čas a energii, aniž by projektu sami přinášeli jakýkoliv užitek. Takoví lidé obvykle hledají slabá místa v zavedených postupech projektu, aby s jejich využitím získali více vlivu, než jaký by jinak měli. To je mnohem zákeřnější než obyčejná hrubost, protože ani jejich chování, ani škody, které působí, nejsou na první pohled nápadné. Klasickým příkladem je hra o čas, tedy situace, v níž někdo neustále tvrdí (a přitom zní tak rozumně, jak jen to je možné), že o probíraném tématu je ještě příliš brzo rozhodovat, a nabízí další a další možná řešení nebo nový pohled na stará řešení; ve skutečnosti je to ovšem tak, že cítí, že se blíží dosažení konsenzu nebo že se bude už brzy hlasovat, a nelíbí se mu, jak to pravděpodobně dopadne. Dalším příkladem může být debata, ve které se nedaří dosáhnout konsenzu, takže se v ní skupina alespoň pokouší vyjasnit, v čem spočívá problém, a sestavit pro ostatní členy projektu shrnutí. Člověk potížista, jenž ví, že toto shrnutí může vést k výsledku, který se mu nebude líbit, se často pokusí zdržet už to shrnutí samotné tím, že bude neúnavně komplikovat otázku, co by v něm mělo být, tím, že bude buď napadat rozumné návrhy, nebo do diskuze neustále vnášet nečekané nové prvky.

Jak se s obtížnými lidmi vypořádat

Pokud se chcete takovému chování postavit, je dobré nejprve pochopit mentalitu těchto lidí. Obvykle to nedělají vědomě. Nikdo se ráno neprobudí s tím, že by si řekl: „Tak a dnes budu cynicky manipulovat rozhodovacími procesy projektu, abych všem házel klacky pod nohy a úplně je tím otrávil.“ Je to jinak: podobným činnostem často předchází poloparanoidní pocit, že danou osobu skupina vyloučila ze svých interakcí a ze svého rozhodování. Tento člověk má pak pocit, že jej nikdo nebere vážně nebo (ve vážnějších případech) že se proti němu všichni spikli – že se ostatní členové projektu rozhodli ustanovit jakýsi exkluzivní klub, do nějž on nepatří. V jeho hlavě je pak toto podezření dostatečným oprávněním pro to brát všechna nařízení doslova a zmanipulovat standardní postupy projektu tak, aby ostatní donutil brát jej vážně. V extrémních případech může dokonce získat dojem, že je osamělým válečníkem ve šlechetné bitvě, jež zachrání projekt před projektem samotným.

Typickým rysem takového útoku zevnitř je to, že si jej lidé začnou všímat až postupně, nikoliv najednou; někteří si jej nevšimnou dokonce vůbec, dokud jim nepředložíte důkazy. Právě proto je tak obtížné se s ním vypořádat. Nestačí přesvědčit sám sebe, že k něčemu takovému dochází – musíte sehnat dostatek důkazů, abyste přesvědčili i ostatní, a pak tyto důkazy promyšleně zveřejnit.

Vzhledem k tomu, že to opravdu stojí spoustu práce, je často lepší celou věc nějakou dobu jednoduše tolerovat. Berte to jako jakousi parazitní, ale celkem neškodnou chorobu. Pokud to vyloženě nenarušuje jeho provoz, může si projekt takové onemocnění dovolit a léčba by mohla mít nepříjemné vedlejší účinky. Nicméně v momentě, kdy už začne tento postup projektu škodit natolik, že už jej nelze přehlížet, je čas jednat. Začněte si psát poznámky o tom, jaké vzorce chování se u útočníka objevují. Nezapomeňte k nim přidávat odkazy na veřejný archiv – tohle je jeden z důvodů, proč si projekt archivy vůbec vede, takže je jen rozumné je využít. Jakmile máte před sebou dost důkazního materiálu, začněte se o věci bavit soukromě s ostatními účastníky projektu. Neříkejte jim hned, k čemu jste dospěli;

nejprve se jich zeptejte, čeho si všimli oni. Může to být vaše poslední šance získat nefiltrovanou zpětnou vazbu o tom, jak chování problémového člena vnímají ostatní. Jakmile o věci začnete mluvit veřejně, začnou se názory radikalizovat a nikdo už si nevzpomene, co si myslel dřív.

Pokud soukromé rozhovory ukážou, že alespoň někteří z oslovených vnímají stejný problém, je čas něco udělat. V tento moment byste měli začít být opravdu velmi opatrní, protože pro daného člověka je velmi snadné překroutit věc tak, aby to vypadalo, že jste si na něj nespravedlivě zasedli. Ať už se rozhodnete udělat cokoliv, nikdy jej nenapadněte, že úmyslně zneužívá postupy projektu, že je paranoidní, ani z ničeho jiného, co předpokládáte, že je v pozadí celé věci. Vaše strategie by měla vypadat rozumně a vyjadřovat starost o celkové zdraví projektu; jejím cílem je buď změnit chování tohoto jednotlivce, nebo jej trvale odstavit. V závislosti na ostatních vývojářích a na tom, jaké s nimi máte vztahy, může být užitečné nejprve získat spojence v soukromých rozhovorech, ale také nemusí. Může to pouze vést k tomu, že si o vás ostatní pomyslí, že hrajete nefér hru a spolčujete se proti někomu v zákulisí.

Nezapomeňte, že třebaže je to ten druhý, kdo se chová destruktivně, budete to vy, kdo bude vypadat jako záškodník, pokud vznesete obvinění, které nebudete schopni doložit. Připravte si hromadu příkladů, kterými můžete dokázat to, co říkáte, a formulujte své obvinění sice přímo, ale tak opatrně, jak jen to je možné. Osobu, o kterou se jedná, sice asi nepřesvědčíte, ale to nevadí. Důležité je přesvědčit všechny ostatní.

Případová studie

Za těch více než 10 let, co pracuji ve svobodném softwaru, si pamatuji jen jedinou situaci, kdy došly věci tak daleko, že jsme museli někoho požádat, aby úplně přestal přispívat. A jak se často stává, nebylo to proto, že by byl hrubý, naopak: zcela upřímně chtěl jen pomoct. Jenom nedokázal vůbec odhadnout, kdy přispívat a kdy ne. Naše mailing listy byly veřejně přístupné a on přispíval tak často a kladl otázky týkající se tolika různých témat, že tím celou komunitu zavalil. Zkoušeli jsme ho zdvořile požádat, aby si dal trochu víc práce s hledáním odpovědi předtím, než se zeptá, ale bezvýsledně.

Strategie, která nám nakonec pomohla, je ukázkovým příkladem toho, jak vystavět případ na základě neutrálních, kvantitativních dat. Jeden z našich vývojářů provedl průzkum v archivech a pak soukromě poslal následující zprávu několika ostatním. Člověk, který způsoboval problémy (třetí jméno v seznamu, zde uveden jako „J. Novák“), nebyl u projektu dlouho a nepřispěl žádným kódem ani dokumentací. Přesto ale byl třetím nejaktivnějším přispěvatelem v mailing listu:

```
Od: "Brian W. Fitzpatrick" <fitz@collab.net>
Komu: [… seznam adresátů byl kvůli udržení anonymity vynechán …]
Předmět: Vysávání energie z projektu Subversion
Datum: Wed, 12 Nov 2003 23:37:47 -0600
```

```
Za posledních 25 dní bylo nejaktivnějších těchto 6 přispěvatelů do svn
[dev|users]:

      294 kfogel@collab.net
      236 "C. Michael Pilato" <cmpilato@collab.net>
      220 "J. Novák" <jnovak@problematicka-osoba.com>
      176 Branko Čibej <brane@xbc.nu>
      130 Philip Martin <philip@codematters.co.uk>
      126 Ben Collins-Sussman <sussman@collab.net>

Řekl bych, že pět z těchto lidí přispívá tomu, aby v blízké budoucnosti
Subversion dosáhl verze 1.0.

Také bych řekl, že jeden z těchto lidí ustavičně vysává čas a energii těm
ostatním 5, nemluvě o mailing listu jako takovém, čímž (třebaže neúmyslně)
zpomaluje vývoj Subversion. Analýzu po vláknech jsem nedělal, ale po projetí
svého mailového úložiště jsem zjistil, že na každý e-mail od této osoby
poslali nejméně 2 z těch ostatních 5 lidí alespoň jednu reakci.
Myslím, že je tady potřeba provést nějaký radikální zásah, i pokud by to
znamenalo zmíněnou osobu úplně odstavit. Už jsme zjistili, že zdvořile a po
dobrém to nejde.

dev@subversion je mailing list, jehož smyslem je pomoct při vývoji systému
správy verzí, a ne skupinová terapie.

- Fitz (po vyčerpávajícím prohrabání se hromadou svn e-mailů, která se
  nashromáždila za tři dny)
```

Třebaže to tak na první pohled nemusí vypadat, chování J. Nováka byla klasická ukázka zneužívání postupů projektu. Nedělal nic nápadného, jako je hra o čas před hlasováním, ale využíval toho, že mailing list spoléhal na sebemoderaci svých členů. Nechali jsme na uvážení každého jednotlivce, kdy a co přispěje. Neměli jsme tedy vůbec žádný mechanismus pro to, jak se vypořádat s někým, kdo takové uvážení nebyl schopen nebo ochoten provést. Nebylo žádné pravidlo, na které bychom mohli ukázat a říct, že jej dotyčný porušil, ale všichni věděli, že frekvence jeho příspěvků způsobovala závažné problémy.

Ze zpětného pohledu se mi Fitzova strategie jeví jako geniální. Shromáždil dostatek průkazného, kvantitativního materiálu, ale podělil se o něj diskrétním způsobem, nejprve s několika lidmi, jejichž podpora by byla při provádění radikálního zásahu klíčová. Všichni souhlasili s tím, že je něco potřeba udělat; nakonec jsme J. Novákovi zavolali po telefonu, problém mu přímo popsali a požádali jej, aby přestal přispívat. Myslím, že nikdy úplně nepochopil, proč vlastně. Kdyby to byl schopen pochopit, pravděpodobně by k celé věci vůbec nedošlo. Ale souhlasil s tím, že přestane přispívat, a mailing list se stal opět použitelným. Jeden z důvodů, proč tato strategie fungovala, byla možná implicitní

výhrůžka, že bychom jeho příspěvky začali omezovat použitím moderátorského softwaru, obvykle nasazovaného proti spamu (viz **Ochrana před nežádoucími zprávami** v kapitole **3. Technická infrastruktura**). Ale důvod, proč jsme mohli mít tuto možnost v záloze, bylo to, že Fitz předtím získal podporu klíčových osob.

Jak se vyrovnat s růstem

Cena za úspěch je v open source světě vysoká. Jak se váš software stává populárnějším, počet lidí, kteří hledají informace, drasticky roste, zatímco počet těch, kteří je mohou poskytnout, stoupá mnohem pomaleji. Navíc i kdyby tento poměr zůstal vyvážen, pořád zde existuje zcela zásadní problém škálovatelnosti prostředků, kterými většina open source projektů zajišťuje komunikaci. Vezměte si například mailing listy. Většina projektů má mailing list pro obecné dotazy uživatelů – obvykle se jmenuje „uživatelé“, „diskuze“, „nápověda“ nebo nějak podobně. Ať už se jmenují jakkoliv, jejich význam je vždy stejný: poskytnout místo, kam lidé mohou psát své dotazy a přečíst si odpovědi a kde jiní lidé mohou z těchto interakcí načerpat informace.

Takový mailing list funguje velmi dobře až do limitu několika tisíc uživatelů a pár stovek příspěvků denně. Někde za těmito čísly se ale systém začíná rozpadat, protože každému účastníku se zobrazují všechny příspěvky; jakmile tedy jejich počet začne překračovat objem, který jeden člověk dokáže za den zpracovat, stává se mailing list pro své účastníky přítěží. Představte si například, že by Microsoft měl takový mailing list pro Windows XP. Systém Windows XP má stovky miliónů uživatelů. I kdyby vznášela za každých 24 hodin nějaký dotaz pouhá jedna desetina jednoho procenta z nich, znamenalo by to stovky tisíc příspěvků za den! Takový mailing list by tedy samozřejmě neexistoval, protože by k němu nikdo nezůstal přihlášen. A tento problém není omezen jen na mailing listy – stejná logika funguje u IRC kanálů, online diskuzních fór i jakéhokoliv jiného systému, v němž skupina odpovídá na dotazy jednotlivců. Důsledky těchto skutečností jsou poněkud depresivní: klasický open source model masivně paralelizované podpory jednoduše není škálovatelný na úrovně, které by dokázaly pokrýt celosvětové nasazení softwaru.

Když fóra překročí tento bod zlomu, nedojde k žádné okázalé explozi. Místo toho se vplíží tichá, negativní zpětná vazba: lidé se začnou odhlašovat z mailing listu, opustí IRC kanál nebo přinejmenším přestanou klást otázky, protože je jasné, že přes všechen ten šum je nikdo neuslyší. Jak se více a více lidí rozhodne pro tento vysoce rozumný přístup, aktivita na fóru se zdánlivě bude udržovat na stejné, zvladatelné úrovni. Jenže na této zvladatelné úrovni se bude držet právě proto, že všichni racionální (nebo přinejmenším zkušení) lidé začali hledat informace jinde – zůstávají tedy pouze ti nezkušení, kteří dál zasílají své příspěvky. Jinými slovy, jeden vedlejší účinek toho, že projekt při svém růstu používá stále stejné, neškálovatelné komunikační modely, je to, že klesá průměrná kvalita otázek i odpovědí, takže se zdá, že noví uživatelé jsou hloupější než bývali, což ale pravděpodobně nejsou. Stalo se pouze to, že poměr vložené energie vůči zisku na takto vysoce frekventovaných fórech klesl, takže ti, kteří jsou dostatečně zkušení na to, aby věděli kam jít informace shánět jinam, je zcela pochopitelně opustili. Úprava komunikačních mechanismů zaměřená na to, vyrovnat se s růstem projektu, tedy obsahuje dvě vzájemně propojené strategie:

1. Rozeznání konkrétních částí fóra, které nadměrným růstem netrpí, třebaže fórum jako takové ano, a jejich oddělení do nových, více specializovaných fór (tedy nenechat to špatné, ať s sebou stáhne dolů i to dobré).
2. Zajištění, aby existovalo velké množství automatizovaných zdrojů informací, které jsou organizované, aktualizované a dají se snadno najít.

Strategie (1) obvykle není obtížná. Většina projektů začne s jedním hlavním fórem: mailing listem pro obecnou diskuzi, kde se probírají nápady na nové funkce, otázky návrhu a programátorské problémy. V tomto mailing listu jsou všichni, kdo se projektu účastní. Po nějaké době bude znát, že se mailing list vyvinul do jiné podoby, v níž má několik různých dílčích listů, rozdělených podle témat. Některá vlákna se například týkají pouze vývoje a návrhu, další jsou uživatelské otázky typu „Jak udělám X?"; může existovat i třetí skupina témat týkající se zpracovávání bug reportů a žádostí o zlepšení a tak dál. Každý jednotlivec samozřejmě může přispívat do mnoha různých typů vláken, ale důležité je, že typy samotné se moc nepřekrývají. Lze je rozdělit do samostatných mailing listů, aniž by to vedlo k zbytečnému rozštěpení pozornosti, protože vlákna samotná hranice typů jen málokdy překročí.

Toto dělení probíhá ve dvou fázích. Založíte nový mailing list (nebo IRC kanál nebo cokoliv jiného) a pak strávíte tolik času, kolik bude potřeba, tím, že budete všechny zdvořile otravovat a připomínat jim, že by jej měli používat. To může trvat týdny, ale časem se to uchytí. Stačí být důsledný a napomínat každého, kdo pošle svůj příspěvek tam, kam nemá, a to veřejně, aby všichni ostatní byli motivováni dělat totéž. Je také užitečné založit webovou stránku, která bude obsahovat informace o všech vašich listech; ve své odpovědi pak můžete dotyčnou osobu na tuto stránku odkázat, což má ještě tu výhodu, že se pak možná naučí, že před přispíváním by si měl přečíst pravidla.

Strategie (2) je dlouhodobý proces, který probíhá po celý život projektu a do nějž je zapojeno mnoho lidí. Do určité míry tato strategie spočívá v tom mít aktualizovanou dokumentaci (viz **Dokumentace** v kapitole **2. Zahájení projektu**) a uživatele na ni odkazovat. Patří sem ale ještě mnohem víc věcí, které podrobně probereme v následujících oddílech.

Nápadné využívání archivů

Typicky je veškerá komunikace v open source projektu (občas s výjimkou IRC konverzací) archivována. Tyto archivy jsou veřejné, lze v nich hledat a jsou referenčně stabilní, což znamená, že jakmile je nějaká informace zaznamenána na určité adrese, zůstává na této adrese dostupná napořád.

Tyto archivy používejte, co nejvíc to jde a co nejvíc nápadně to jde. I pokud znáte odpověď na nějakou otázku z hlavy, tak pokud si myslíte, že je zodpovězena někde v archivu, věnujte čas tomu tento záznam v archivu najít a poslat místo odpovědi odkaz na něj. Když tohle uděláte na veřejném fóru, nenápadně tím všechny, kdo to nevěděli, informujete o tom, že existuje něco jako archiv a že se v něm dají najít odpovědi. Odkázáním na archiv místo zopakováním rady také posilujete sociální normu, která radí neduplikovat informace. Proč mít stejnou odpověď na dvou různých místech? Když bude počet míst, kde ji lze najít, držen na minimu, je větší pravděpodobnost, že ti, kteří ji našli, si zapamatují,

jak se k ní dostat znovu. Dobře použité reference také obecně zlepšují kvalitu výsledků hledání, protože zvyšují význam cílového odkazu v internetových vyhledávačích.

Někdy ale má duplikování informací smysl. Například si představme situaci, kdy už v archivu existuje odpověď, která není od vás a v níž stojí:

```
Zdá se, že se vám rozházely Scanley indexy. Pokud je chcete dát dohromady,
udělejte tohle:

1. Vypněte Scanley server.
2. Spusťte program „defrobnicate“, který je dodáván se Scanley.
3. Spusťte server.
```

Pak o několik měsíců později uvidíte příspěvek, z nějž usoudíte, že má někdo stejný problém s indexy. Prohledáte archiv a najdete odpověď zmíněnou výše, ale všimnete si, že v ní několik kroků chybí (možná kvůli přehlédnutí původního autora, možná proto, že software se od té doby změnil). Nejelegantnější způsob, jak se s tím vyrovnat, je napsat nové, doplněné instrukce, a explicitně prohlásit starý návod za neplatný:

```
Zdá se, že se vám rozházely Scanley indexy. Stejný problém už se vyskytl v červenci
a J. Novák uveřejnil řešení na http://blablabla/bla. Tady je popis, jak indexy spravit,
který vychází z postupu J. Nováka, ale je o něco podrobnější:

1. Vypněte Scanley server.
2. Přihlaste se jako uživatel, pod nímž Scanley server standardně běží.
3. Jako tento uživatel spusťte na indexech program „defrobnicate“.
4. Spusťte Scanley ručně a přesvědčte se, že indexy už fungují.
5. Restartujte server.
```

(V ideálním světě by bylo možné ke starému příspěvku přidat poznámku o tom, že jsou dostupné novější informace, a přidat odkaz. Nicméně nevím o žádném software pro správu archivu, který by měl funkci „příspěvek zastaralý kvůli jinému příspěvku“; možná proto, že by ji bylo složité implementovat způsobem, který by nenarušil integritu archivů jako spolehlivého historického záznamu. To je další důvod, proč je vytvoření specializovaných webových stránek s odpověďmi na často kladené dotazy dobrý nápad.)

Archivy jsou nejčastěji používány jako zdroj odpovědí na technické dotazy, ale jejich význam pro projekt je mnohem větší. Pokud jsou formální pravidla projektu jeho zákonným právem, pak archivy jsou jeho právem zvykovým: záznamem všech rozhodnutí a toho, jak se k nim došlo. V každé opakující se diskuzi je dnes v podstatě povinností začít prohledáváním archivu. Díky tomu můžete zahájit debatu shrnutím stávajícího stavu věcí, předvídat námitky, připravit si na ně odpovědi a možná objevit i úhly pohledu, které vás nenapadly. Nezapomeňte také, že ostatní účastníci budou jaksi automaticky předpokládat, že archivy jste už prohledali. I pokud předchozí diskuze nikam nevedly, měli byste

na ně při vznesení stejné otázky odkázat, aby všichni věděli, že a) nikam nevedly a že b) jste splnili svou povinnost a pravděpodobně tedy říkáte něco nového, co ještě řečeno nebylo.

Se všemi zdroji zacházejte jako s archivy

Všechny rady uvedené výše se nevztahují jen na archivy mailing listů. To, že některé informace lze nalézt na stabilních, snadno dohledatelných adresách, by měl být organizující princip celého projektu. Jako případovou studii vezmeme seznam často kladených otázek projektu, FAQ.

Jak lidé FAQ používají?

1. Chtějí v něm hledat specifická slova a fráze.
2. Chtějí si jej pročítat a vstřebávat informace, aniž by nutně hledali odpověď na nějaké konkrétní dotazy.
3. Očekávají, že vyhledávače, jako je Google, obsah FAQ znají, takže se ve výsledcích hledání objeví.
4. Chtějí mít možnost poskytnout ostatním odkaz na konkrétní položky FAQ.
5. Chtějí mít možnost přidat k FAQ nový materiál; mějte ale na paměti, že to se stává nesrovnatelně méně často než vyhledávání informací. Mnohem více lidí FAQ čte, než do něj píše.

Bod 1 implikuje, že FAQ by mělo být dostupné v nějakém textovém formátu. Body 2 a 3 naznačují, že FAQ by mělo být k dispozici jako HTML stránka; bod 2 navíc znamená, že by toto HTML mělo být navrženo tak, aby bylo dobře čitelné (jinými slovy, chcete mít nějakou kontrolu nad tím, jak bude vypadat a působit) a mělo by mít na začátku obsah. Bod 4 znamená, že každý jednotlivý záznam v FAQ by měl mít v HTML přidělenou tzv. *kotvu* (named anchor), tedy tag, který umožňuje ostatním dostat se na konkrétní místo na stránce. Bod 5 znamená, že zdrojové soubory pro FAQ by měly být dobře dostupné (viz **Sledujte verze u všeho** v kapitole **3. Technická infrastruktura**) a ve formátu, jenž lze snadno upravit.

Kotvy a atributy id

Existují dva způsoby, jak prohlížeč odkázat na konkrétní místo na webové stránce: kotvy a atributy id. Takzvaná *kotva* je standardní HTML odkazovací tag (`<a>…</a>`), který má navíc atribut name, tedy jméno:

```
<a name="popisek">...</a>
```

Novější verze HTML podporují také generický *atribut id*, který lze připojit k jakémukoliv HTML prvku, nejen k `<a>`. Například:

```
<p id="popisek">…</p>
```

Kotvy a atributy id se používají stejným způsobem. Za URL se přidá mřížka a jméno popisku, díky čemuž prohlížeč na stránce přeskočí na dané místo:

```
http://mujprojekt.priklad.com/faq.html#popiska
```

Kotvy jsou podporovány v podstatě ve všech existujících prohlížečích; většina moderních prohlížečů podporuje i atribut id. Pro jistotu bych doporučil používat buď kotvy samotné, nebo kotvy a id dohromady (se stejným popiskem, samozřejmě). Kotvy nemohou být samouzavírací – i pokud uvnitř prvku nic není, stejně musíte napsat obě poloviny:

```
<a name="popisek"></a>
```

…třebaže standardně mezi nimi nějaký text bude, například jméno oddílu.

Ať už použijete kotvu, nebo atribut id, nebo obojí, nezapomeňte, že popisek nebude viditelný tomu, kdo na stránku dorazí běz něj. Taková osoba ovšem může chtít zjistit, jaký popisek má konkrétní položka FAQ, aby například mohli někomu poslat přesný odkaz. V tom jim může pomoci, když ke stejnému prvku, k němuž jste přidali atributy „name“ a/nebo „id“, dodáte ještě *atribut title*, tedy titul, například takto:

```
<a name="popisek" title="#popisek">…</a>
```

Když nad textem elementu s atributem title podržíte kurzor myši, většina prohlížečů zobrazí malé okénko s jeho obsahem. Do „title“ obvykle přidávám i znak mřížky, abych uživatelům připomněl, co musí přidat na konec URL, pokud se budou příště chtít dostat přímo na stejné místo.

Formátování FAQ tímto způsobem je jen jedním příkladem, jak připravit nějaký zdrojový materiál do dobře přístupné podoby. Stejné vlastnosti – možnost přímého prohledávání, dostupnost pro velké internetové vyhledávače, možnost prohlížení, referenční stabilita a (tam, kde je to relevantní) možnost editace – by měly mít i ostatní webové stránky, strom zdrojového kódu, bug tracker atd. Většina sofwaru pro archivování mailing listů už si důležitost těchto vlastností dávno uvědomila, takže mailing listy obvykle mají tyhle funkce už nativně; ostatní formáty mohou vyžadovat trochu více údržby (kapitola **8. Řízení dobrovolníků** přináší rady, jak tuto povinnost rozdělit mezi více dobrovolníků).

Kodifikace tradic

S přibývajícím časem roste složitost projektu a s ní i objem dat, která musí zpracovat každý nově příchozí člen. Ti, kteří jsou u projektu už dlouho, se se zvyklostmi projektu seznamovali a pomáhali je vytvářet postupně v průběhu celého procesu. Nebudou si uvědomovat, jak obrovské množství tradic v projektu vlastně je, a možná budou překvapeni, kolik chyb v nich nováčci dělají. Samozřejmě problém není v tom, že by nováčci byli méně kvalitní než dříve, ale v tom, že zapojení se do nové kultury je pro ně mnohem těžší než pro ty, kdo přišli před nimi.

Tradice projektu zahrnují jak zvyklosti komunikace a uchovávání informací, tak standardy pro psaní kódu a další technické detaily. V této knize už jsme se setkali s oběma typy standardů, v části **Dokumentace pro vývojáře** v kapitole **2. Zahájení projektu** a v části **Jak to všechno zapsat**, kapitole

4. Společenská a politická infrastruktura, kde jsou uvedeny i příklady. V tomto oddílu se podíváme na to, jak tyto standardy a návody udržovat aktuální během vývoje projektu, a to zejména ty, jež se týkají řízení komunikace, neboť ty se s tím, jak projekt a jeho složitost roste, proměňují nejvíc.

Za prvé sledujte, co nejčastěji vede ke zmatení. Pokud uvidíte, že se některé situace pořád opakují, zejména u nově příchozích účastníků, pak je dost pravděpodobné, že existuje nějaké pravidlo, které by mělo být zdokumentováno, ale není. Za druhé, nesmí vás unavit to, že budete něco opakovat pořád dokola, a nesmí to ani vypadat, že vás to už unavuje. Vy i další veteráni projektu se budete muset opakovat celkem často – to je nevyhnutelný vedlejší efekt přibírání nových lidí.

Každou stránku, každou zprávu na mailing listu a každý IRC kanál byste měli považovat za prostor pro reklamu – a tím nemyslím komerční reklamu, ale propagaci zdrojů vašeho projektu. To, co tam umístíte, záleží na demografickém složení těch, kdo to pravděpodobně budou číst. Na IRC kanálu pro uživatelské dotazy se například pravděpodobně budou objevovat lidé, kteří s projektem dosud nijak nekomunikovali – často je to někdo, kdo si software právě nainstaloval a má dotaz, který by chtěl mít zodpovězen hned (koneckonců kdyby mohl čekat, poslal by svůj dotaz raději na mailing list, což by ho pravděpodobně v celkovém součtu stálo méně času, ale trvalo by déle, než by odpověď přišla). Lidé obvykle do IRC kanálů nijak trvale neinvestují – přijdou, položí svůj dotaz a zmizí.

Téma kanálu by tedy mělo mířit na ty, kdo shánějí technické odpovědi na otázky týkající se softwaru, a to co nejdřív, spíše než na ty, kteří by se možná chtěli do projektu zapojit dlouhodobě a komu by tedy více prospělo přečíst si pravidla pro interakci s komunitou. Tady je příklad, jak to řeší jeden velmi vytížený kanál (porovnejte to s dřívějším příkladem v části **IRC a jiné systémy pro diskuzi v reálném čase** v kapitole **3. Technická infrastruktura**):

```
Nacházíte se v #linuxhelp

Téma #linuxhelp je Prosím PŘEČTĚTE SI
http://www.catb.org/~esr/faqs/smart-questions.html &&
http://www.tldp.org/docs.html#howto PŘEDTÍM, než se na něco zeptáte |
Pravidla kanálu: http://www.nerdfest.org/lh_rules.html | Předtím, než se
zeptáte na upgrade na kernel 2.6.x, si přečtěte
http://kerneltrap.org/node/view/799 | čtení z paměti je možné:
http://tinyurl.com/4s6mc -> aktualizujte na 2.6.8.1 nebo 2.4.27 |
havárie hash algoritmu: http://tinyurl.com/6w8rf | reiser4
```

U mailing listů je toto „místo pro reklamu“ krátkou patičkou přidanou ke každé zprávě. Sem dává většina projektů instrukce, jak se k mailing listu přihlásit nebo se odhlásit, a často i odkaz na domovskou stránku nebo FAQ projektu. Možná si pomyslíte, že ti, kdo jsou k mailing listu přihlášeni, už dávno vědí, kde tyto věci najít, a pravděpodobně máte pravdu, ale tyto zprávy čte mnohem víc lidí než jen ti, kdo mailing list odebírají. Odkaz na nějaký archivovaný příspěvek se může objevovat na mnoha jiných místech; některé příspěvky se dokonce stanou natolik populárními, že je nakonec bude číst mnohem více lidí mimo mailing list než na něm.

Formátování zde hraje velkou roli. Například v projektu Subversion jsme neměli moc velký úspěch s použitím techniky pro filtrování chyb popsané v části **Předběžné filtrování v bug trackeru** v kapitole **3. Technická infrastruktura**. Nezkušení uživatelé zadávali velké množství falešných bug reportů a pokaždé, když se to stalo, je o tom musel někdo informovat úplně stejným způsobem, jako těch 500 lidí před ním. Jednoho dne už jednomu z našich vývojářů konečně došla trpělivost a nějakému nebohému uživateli, který si pravidla issue trackeru nepřečetl dost podrobně, strašně vynadal; jiný vývojář to viděl a usoudil, že už to stačilo. Navrhl změnit formát hlavní stránky issue trackeru tak, aby její nejdůležitější část, tedy výzva prodiskutovat chybu před zanesením do systému nejprve v mailing listech a na IRC kanálech, byla uvedena obrovskými tučnými červenými písmeny na jasně žlutém pozadí a aby zastínila všechno ostatní na stránce. To jsme také udělali (výsledek najdete na **http://subversion.tigris.org/project_issues.html**) a vedlo to k výraznému snížení počtu chybně hlášených problémů. Samozřejmě jsme se jich nezbavili úplně – to ostatně ani není možné –, ale začaly se objevovat mnohem méně často, třebaže počet uživatelů dále rostl. Výsledkem je nejen to, že databáze chyb obsahuje méně zbytečných dat, ale i to, že ti, kdo na nahlášené problémy odpovídají, teď na celkem vzácné chybné záznamy reagují podstatně přátelštěji. Tím se zlepšuje jak image projektu, tak i psychická pohoda jeho dobrovolníků.

Poučení, které jsme tak získali, bylo, že jen napsat pravidla nestačí. Také je potřeba je umístit tam, kde si jich všimnou ti, kteří je nejvíc potřebují, a naformátovat je tak, aby je ti, kdo s projektem nejsou obeznámeni, okamžitě identifikovali jako úvodní, informativní materiál.

Statické webové stránky nejsou jediným místem, kde je možné informovat o zvyklostech projektu. Je potřeba také určité množství interaktivního dozoru (jehož nástroje mají být spíš typu „přátelské upozornění“, než „želízka a zavřít“). Všechny revize, dokonce i revize commitů popsané v části **Kontrolujte kód veřejně** v kapitole **2. Zahájení projektu**, by měly zmínit i to, nakolik daná osoba vyhověla nebo nevyhověla normám projektu, zejména co se týče zvyklostí komunikace.

Další příklad z projektu Subversion: shodli jsme se na konvenci, podle níž „r12908“ znamenalo „revize 12908 v úložišti správy verzí“. Malé „r“ na začátku se snadno píše, a protože má poloviční výšku oproti číslům, které za ním následují, je celý blok textu snadno identifikovatelný. Samozřejmě to, že se na této konvenci shodnete, ještě neznamená, že ji hned začnou všichni používat. Takže pokud pak přijde commit mail, v němž je následující zpráva z logu:

```
------------------------------------------------------------------------
r12908 | qsimon | 2005-02-02 14:15:06 -0600 (Wed, 02 Feb 2005) | 4 řádky

záplaty od přispěvatele J. Novák <jncontrib@gmail.com>

* trunk/contrib/client-side/psvn/psvn.el:
  Upraveno pár překlepů revize 12828.
------------------------------------------------------------------------
```

…součástí revize tohoto commitu je i oznámení „Mimochodem, při odkazování na starší změny prosím pište ‚r12828‘ a ne ‚revize 12828‘.“ To není jen puntičkářství – je to důležité jak pro strojové zpracovávání, tak pro lidské čtenáře.

Dodržováním obecného principu, že běžné entity by měly mít stanovené způsoby odkazování a že tyto způsoby by měly být používány konzistentně a všude, projekt vlastně vytváří jisté standardy. Tyto standardy umožňují lidem psát nástroje, které prezentují komunikaci projektu použitelnějšími způsoby – například revize formátovaná „r12828“ může být v systému prohlížení úložiště automaticky převedena na živý odkaz. To by bylo obtížnější, pokud by byla psána ve formátu „revize 12828“, a to jak proto, že by se mezi slovem a číslem mohl vyskytnout konec řádku, tak proto, že je to méně výrazné (slovo „revize“ se často objevuje samostatně a skupiny číslic se také často objevují samostatně; kombinace „r12828“ ale nemůže znamenat nic jiného, než číslo revize). Podobně je potřeba přistupovat k věcem, jako jsou čísla verzí, položky FAQ (rada: používejte URL s kotvami, jak je popsáno v části **Kotvy a atributy id**), atd.

I u entit, pro které neexistuje zavedená krátká, kanonická forma, byste stejně měli podporovat to, aby klíčové informace byly dodávány konzistentním způsobem. Například pokud odkazujete na zprávu z archivu mailing listu, neuvádějte jen odesilatele a předmět, ale také její URL v archivu a hlavičku Message-ID. Poslední položkou umožňujete těm, kdo mají vlastní kopii mailing listu (lidé si občas pořizují offline kopie, například pro čtení na notebooku na cestách), jednoznačně identifikovat konkrétní zprávu i bez přístupu do archivu. Odesilatel a předmět nestačí, protože jedna osoba může ve stejném vlákně přispívat několikrát, často i v ten samý den.

Čím více projekt roste, tím důležitější tato konzistence je. Konzistence znamená, že všude, kam se lidé podívají, uvidí stejné vzorce chování, což je bude motivovat je následovat také. Tím se pak snižuje počet otázek, které musí klást. Zátěž toho, když máte milión uživatelů, není o nic větší, než když máte jednoho; problémy škálovatelnosti se začnou projevovat jen tehdy, když z nich určité procento začne klást otázky. Jak tedy projekt roste, musí toto procento snížit tím, že navýší hustotu a přístupnost informací, takže je větší pravděpodobnost, že někdo nalezne svou odpověď sám, aniž by se musel ptát.

Nediskutujte v bug trackeru

V každém projektu, který aktivně využívá svůj bug tracker, hrozí riziko, že se tento tracker promění v samostatné diskuzní fórum, třebaže by mailing listy posloužily lépe. Obvykle to začne poměrně nevinně: někdo připojí k nějakému problému například návrh řešení nebo částečnou záplatu. To uvidí někdo jiný a všimne si nějakých nedostatků. Připojí tedy další poznámku, v níž je popíše. Na to opět zareaguje ta první osoba připojením další poznámky… a tak dál.

To vede ke dvěma problémům: za prvé, bug tracker je na vedení diskuze velmi nešikovný nástroj, a za druhé, ostatní lidé nemusí celé věci věnovat pozornost – předpokládají totiž, že diskuze o vývoji probíhá na mailing listu, takže tam ji i hledají. Mailing list s oznámením změn stavu nahlášených problémů sledovat vůbec nemusí; pokud ano, tak třeba jen velmi zběžně.

Kde se ale přesně v procesu stala chyba? Bylo to tehdy, když první osoba připojila k problému své řešení – měla jej zaslat do mailing listu? Nebo to bylo, když druhá osoba reagovala v trackeru a ne v mailing listu?

Na tuto otázku neexistuje správná odpověď, ale platí tu obecný princip: pokud přidáváte data k nějakému problému, udělejte to v trackeru, ale pokud zahajujete konverzaci, udělejte to v mailing listu. Tohle rozhodnutí nemusí být vždy úplně jednoznačné; pokud není, spolehněte se na svůj úsudek. Například pokud přikládáte záplatu, která obsahuje potenciálně kontroverzní řešení, můžete předvídat, že k ní budou ostatní vznášet dotazy. Takže třebaže standardně byste tuto záplatu připojili k problému (pokud tedy nechcete nebo nemůžete přímo provést commit), v tomto případě byste se mohli rozhodnout ji raději publikovat v mailing listu. Tak jako tak dříve nebo později dorazíte do bodu, kde by měla být jedna nebo druhá strana schopna poznat, že z pouhého připojení dat se stává plnohodnotná konverzace – v příkladu na začátku tohoto oddílu by touto osobou byl ten druhý, který poté, co si všiml problémů se záplatou, měl odhadnout, že to vyvolá diskuzi, a zahájit ji tedy s použitím vhodného média.

Matematickou analogií řečeno, pokud se zdá, že informace bude rychle konvergovat, dejte ji přímo na bug tracker; pokud se zdá, že vývoj bude divergentní, bude pro ni lepší mailing list nebo IRC kanál.

To neznamená, že by v bug trackeru nemělo nikdy docházet k výměnám názorů. Například požádání prvního respondenta o více podrobností o tom, jak chybu reprodukovat, je obvykle velmi konvergentní proces. Taková odpověď jen těžko vyvolá další debatu, pouze doplní informace, které už v trackeru jsou. Není důvod tímto procesem tříštit pozornost v mailing listu, takže nikdo nebude mít nic proti tomu, když se vyřeší pomocí komentářů v trackeru. Podobně pokud jste si celkem jisti, že chyba byla nahlášená nesprávně (tedy že to chyba není), můžete to ohlásit rovnou na místě. I to, že poukážete na drobný problém s něčím řešením, je v pořádku, pokud to tedy není něco, na čem celé to řešení spočívá.

Na druhou stranu pokud vznášíte filozofické dotazy týkající se rozsahu chyby nebo správného chování softwaru, můžete si být jisti, že se k tomu budou chtít vyjádřit i ostatní vývojáři. Tato diskuze bude pravděpodobně před konvergencí nějakou dobu probíhat divergentně, takže ji zahajte na mailing listu.

Pokud se rozhodnete přesunout debatu na mailing list, přidejte k chybě odkaz na příslušné vlákno. Je důležité, aby někdo, kdo chybu sleduje, dokázal diskuzi najít, i pokud neprobíhá přímo v trackeru. Tomu, kdo celé vlákno začal, může tento proces připadat zbytečně pracný, ale celá kultura open source v zásadě klade zodpovědnost do rukou těch, kteří o chybách píšou – je mnohem důležitější usnadnit věci těm desítkám nebo stovkám lidí, kteří budou chybu číst, než těm třem nebo pěti, kteří o ní budou psát.

Je v pořádku, pokud vezmete důležité závěry diskuze z mailing listu a vložíte je do trackeru, pokud to čtenářům ulehčí práci. Obvykle se to dělá tak, že se začne diskuze v mailing listu, odkaz se vloží k chybě a až diskuze skončí, přidá se k chybě shrnutí (spolu s odkazem na zprávu, z níž to shrnutí pochází), takže někdo, kdo chybu čte, snadno zjistí, jakého závěru bylo dosaženo, aniž by musel překlikávat někam jinam. Všimněte si, že klasický problém duplikace dat s „dvěma hlavními zdroji“ se zde neobjevuje, protože jak archivy, tak komentáře problémů jsou obvykle statická, neměnná data.

Publicita

U svobodného softwaru je hranice mezi čistě interní diskuzí a veřejnými prohlášeními celkem neostrá. Jedním z důvodů je to, že cílová skupina není moc dobře vymezena – vzhledem k tomu, že většina nebo i všechny příspěvky jsou veřejně přístupné, nemá projekt plnou kontrolu nad tím, jaký dojem zanechává na okolní svět. Někdo – řekněme třeba redaktor **slashdot.org** – může přitáhnout pozornost miliónů čtenářů k příspěvku, u nějž nikdo nepředpokládal, že by byl kdy čten mimo projekt. S touhle skutečností musí žít všechny open source projekty, ale v praxi je toto riziko obvykle nízké. Obecně se dá říct, že oznámení, které projekt nejvíc chce publikovat, také těmi nejvíc publikovanými budou, pokud tedy použijete ty správné nástroje, abyste okolnímu světu naznačili, že za zveřejnění stojí.

U velkých oznámení obvykle existuje čtyři nebo pět hlavních distribučních kanálů, na nichž by se měla oznámení objevit pokud možno současně:

1. Titulní stránku projektu pravděpodobně uvidí víc lidí než jakoukoliv jinou jeho část. Pokud tedy máte nějaké opravdu velké oznámení, dejte ho tam. Třeba v podobě malého rámečku, který shrne, o čem oznámení je, a poskytne odkaz na tiskovou zprávu (viz níže), kde se nachází další informace.
2. Vaše stránky by kromě toho měly mít i sekci „Novinky“ nebo „Tiskové zprávy“, kde lze tato oznámení rozepsat podrobně. Jeden z důvodů, proč se vydávají tiskové zprávy, je to, aby existoval jediný, kanonický „předmět oznámení“, na který mohou ostatní servery odkazovat, čemuž by měla odpovídat i jejich struktura – pro každou zprávu samostatná stránka, záznam na blogu nebo nějaká jiná entita, na niž je možno jednoznačně odkázat, aby nebyla zaměněna s jinou zprávou ve stejné sekci.
3. Pokud váš projekt má RSS feed (viz **RSS**), ujistěte se, že se oznámení objevilo i tam. To se může stát automaticky poté, co jste tiskovou zprávu vytvořili – podle toho, jak jsou vaše stránky nastaveny.
4. Pokud se vaše oznámení týká nové verze softwaru, aktualizujte záznam o svém projektu na **http://freshmeat.net/** (pro informace, jak se takový záznam vytvoří, viz **Oznamování**). Pokaždé, když aktualizujete svůj záznam na Freshmeatu, se objeví na jejich seznamu změn pro příslušný den. Tento seznam změn se neaktualizuje jen na Freshmeatu samotném, ale na mnoha dalších portálech (včetně **http://slashdot.org**), které bedlivě sleduje řada lidí. Freshmeat nabízí stejná data i přes RSS, takže ti, kteří neodebírají RSS vašeho projektu, si jej stále ještě můžou všimnout na feedu Freshmeatu.
5. Pošlete e-mail do projektového mailing listu s oznámeními. Jméno takového listu by mělo být „announce“, tedy `announce@domenavasehoprojektu.org`, protože to už je dnes standardní zvyklost, a v jeho pravidlech by mělo být jasně řečeno, že je to mailing list s velmi malým objemem zpráv, určený výhradně pro velká oznámení projektu. Většina z těchto oznámení bude o nových releasech softwaru, ale mohou se zde také objevit zprávy o jiných událostech, jako je výzva k financování, odhalení bezpečnostní chyby (viz **Ohlášení bezpečnostních chyb**) dále v této kapitole, nebo velká změna ve směřování projektu. Protože je využíván poměrně málo a jenom pro důležitá oznámení, mívá mailing list announce obvykle ze všech mailing listů projektu

největší čtenářskou základnu (což samozřejmě znamená, že byste jej neměli zneužívat – zvažte pečlivě, než něco napíšete). Abyste zabránili tomu, že by zasílal oznámení někdo jiný, nebo ještě hůř, že by se tudy dostával spam, musí být mailing list announce vždy moderovaný.

Pokuste se zajistit, aby se stejné oznámení objevilo na všech těchto místech najednou, nebo alespoň s minimálním časovým rozdílem. Lidé mohou být zmateni, pokud v mailing listu uvidí oznámení, ale na hlavní stránce projektu nebo v oblasti pro tiskové zprávy nic takového nenajdou. Pokud si všechny dílčí změny (e-maily, úpravy webových stránek atd.) nejprve nashromáždíte a pak je odešlete všechny najednou, bude to časové období, v němž jsou informace nekonzistentní, jen velmi malé.

U méně důležitých oznámení můžete některé ze zmíněných způsobů vynechat. Okolní svět si této události stejně všimne, a to do takové míry, jakou si zasluhuje její význam. Například zatímco nový release softwaru je významnou událostí, oznámení data příští release, třebaže není úplně nezajímavé, zdaleka tak důležité není. To, že jste se shodli na nějakém datu, je informace, kterou stojí za to poslat e-mailem do standardních mailing listů (ne do listu announce) a přidat do časového harmonogramu projektu nebo k jeho současnému stavu na vašich webových stránkách, ale nic víc.

Stejně se pak může stát, že se s tímto datem setkáte v diskuzích i jinde na internetu, kde má někdo o váš projekt zájem. Lidé, kteří jsou ve vašich mailing listech takzvanými lurkery, tedy ti, kteří pečlivě poslouchají, ale sami nic neříkají, se tak nemusí chovat i jinde. Neformální šíření informací mezi uživateli samotnými je velice schopný nástroj, na nějž byste měli spoléhat; malá oznámení byste dokonce mohli formulovat tak, aby takové šíření podporovala. Konkrétně tím myslím, že příspěvky, u nichž předpokládáte, že budou citovány i jinde, by měly obsahovat i část, která si o citování vyloženě říká, protože připomíná formální tiskovou zprávu. Například:

Jenom malá zpráva o současném stavu: předpokládáme, že Scanley 2.0 vyjde v polovině srpna 2005. Aktuální informace najdete vždy na **http://www.scanley.org/status.html**. *Nejvýznamnější novou funkcí v této verzi bude hledání pomocí regulárních výrazů.*

Mezi další nové funkce patří: … Nová verze také opraví několik známých chyb, včetně: …

První odstavec je krátký, podává dvě nejdůležitější informace (datum vydání a hlavní novinku) a URL s podrobnostmi. Pokud tenhle odstavec bude to jediné, co si někdo přečte, pak jste svou práci odvedli dobře. Zbytek e-mailu se může klidně ztratit, aniž by to mělo velký dopad na pochopení jeho obsahu. Samozřejmě ho někdy lidé vezmou a někam ho zkopírují celý, ale zrovna tak často z něj budou citovat jen kousek. A právě proto, že to není nijak neobvyklé, jim můžete práci trochu ulehčit; pro vás to pak znamená, že budete mít částečný vliv na to, jakou pasáž se rozhodnou ocitovat.

Ohlášení bezpečnostních chyb

S bezpečnostními chybami je třeba zacházet jinak než s jakýmkoliv jiným druhem bug reportu. V kultuře svobodného softwaru se obvykle krédo dělat věci otevřeně a transparentně dodržuje s někdy až fanatickým zápalem. Každý krok standardního procesu pro řešení chyb je otevřený všem, kdo mají zájem jej sledovat: zaslání zprávy o chybě, následující diskuze a nakonec oprava.

Bezpečnostní trhliny jsou ale něco jiného. Mohou ohrozit data vašich uživatelů, možná i celé jejich počítače. Kdybyste takovýto problém probírali na veřejnosti, oznámili byste tak jeho existenci celému světu – tedy i těm, kdo by mohli chybu zneužít. I samotný commit opravy může fungovat jako oznámení, že chyba existuje (najdou se potenciální útočníci, kteří sledují commit logy veřejných projektů a systematicky hledají změny, které by mohly naznačovat, že se v kódu aktuální verze nacházejí bezpečnostní problémy). Většina open source projektů se shodla na v zásadě stejných krocích, jimiž řeší tento spor mezi otevřeností a tajnůstkářstvím, na základě následujících pravidel:

1. Nemluvte o chybě veřejně, dokud není dostupná oprava; tu dodejte přesně v tu samou chvíli, kdy oznamujete chybu.
2. Vytvořte opravu, co nejrychleji to dokážete – zejména tehdy, když chybu oznámil někdo zvenčí, protože pak víte, že existuje alespoň jedna osoba mimo projekt, která ji může zneužít.

V praxi tato pravidla vedou k poměrně standardizované sérii kroků, které popíšeme v dalších oddílech.

Přijměte report

V projektu z pochopitelných důvodů musí být možnost přijímat oznámení o bezpečnostní chybě od kohokoliv. Nehodí se k tomu ale běžná adresa pro bug reporty, protože i tu může sledovat kdokoliv. Pro přijímání reportů bezpečnostních chyb byste tedy měli mít zvláštní mailing list. Tento mailing list nesmí mít veřejně přístupné archivy a skupina jeho odběratelů musí být přísně kontrolována – smí do něj jen vývojáři, kteří jsou u projektu dlouho a kterým věříte. Pokud potřebujete přesnější definici, kdo jsou „ti, kterým věříte“, můžete například vzít podmínku „každý, kdo má commit access dva roky nebo déle“, nebo něco podobného, abyste byli spravedliví. Tohle je pak ta skupina, která bude řešit bezpečnostní problémy.

Ideálně by tento mailing list neměl používat antispamovou ochranu ani by neměl být moderován, protože nechcete, aby nějaké důležité oznámení buď zůstalo zachycené ve filtru, nebo se zpozdilo jen proto, že ten víkend zrovna žádný moderátor nebyl online. Pokud používáte software na automatickou ochranu proti spamu, nastavte jej na nejvyšší toleranci – je lepší, když trochu spamu pustí, než když zadrží nějaký report. Aby tento list mohl fungovat, musí být samozřejmě jeho adresa veřejně dostupná, ale právě proto, že je nemoderovaný a s jen minimální ochranou proti spamu, dejte si pozor, abyste tuto adresu nikde nepublikovali bez použití nějaké transformace pro její skrytí,

jak popisuje část **Skrývání adres v archivech** v kapitole **3. Technická infrastruktura**. Naštěstí to, že je adresa skrytá, nemusí znamenat, že je nečitelná – podívejte se například na stránku na adrese **http://subversion.tigris.org/security.html** a prohlédněte si její HTML zdrojový kód.

Opravu vyvíjejte potichu

Co tedy tento specializovaný mailing list udělá, když obdrží report? Prvním úkolem je vyhodnotit důležitost a naléhavost problému:

1. Jak závažná chyba to je? Umožňuje osobě s nekalými úmysly převzít kontrolu nad počítačem někoho, kdo používá váš software? Nebo nedělá nic jiného, než že například oznamuje informace o velikosti některých jejich souborů?
2. Jak snadné je chybu zneužít? Stačí útočníkovi napsat jednoduchý skript, nebo je k tomu potřeba velké množství znalostí, kvalifikovaný odhad a určitá dávka štěstí?
3. Kdo vám problém nahlásil? Odpověď na tuto otázku samozřejmě nemění podstatu problému, ale můžete si podle ní udělat představu, kolik jiných lidí o chybě může vědět. Pokud report pochází od jednoho z vývojářů projektu, pak si můžete trochu oddechnout (ale ne zase moc), protože jim můžete věřit, že o tom nikomu jinému neřekli. Pokud vám ale přišla e-mailem z adresy `anonymous14@globalhackerz.net`, pak byste měli reagovat tak rychle, jak jen dokážete. Tato osoba sice ukázala dobrou vůli tím, že vás o problému informovala, ale nemáte jak zjistit, kolika jiným lidem už to oznámila nebo jak dlouho bude čekat, než začne tuto chybu zneužívat na počítačích v ostrém provozu.

Je potřeba si uvědomit, že rozdíl, o němž tady mluvíme, je často jen drobný – tenká hranice mezi tím, co je naléhavé a co je extrémně naléhavé. I pokud report pochází ze známého, přátelského zdroje, mohou být někde na internetu jiní lidé, kteří chybu objevili už dávno, ale nenahlásili ji. Věc není naléhavá pouze tehdy, když chyba ohrožuje bezpečnost uživatelů jen opravdu velmi málo.

Tím příkladem s „`anonymous14@globalhackerz.net`“ jsem si mimochodem nedělal legraci. Opravdu se může stát, že dostanete bug reporty od osob se skrytou identitou, z jejichž slov a chování nikdy tak úplně nepoznáte, jestli jsou na vaší straně nebo ne. Na tom ale nezáleží: pokud vám nahlásili bezpečnostní chybu, budou mít pocit, že vám vykonali službu, a podle toho byste na ně měli reagovat. Poděkujte jim za tuto zprávu, napište, do jakého data plánujete zveřejnit opravu, a pak je trvale informujte o průběhu. Někdy se může stát, že to budou oni, kdo vám dá nějaké datum – s implicitní výhrůžkou, že tohoto dne zveřejní informace o chybě, ať už budete připraveni nebo ne. Může vám to připadat jako vydírání, ale pravděpodobně je to spíš následek toho, že danou osobu už v minulosti zklamali jiní vývojáři software, kteří její oznámení bezpečnostní chyby nebrali moc vážně. Ať už je to jakkoliv, nemůžete si dovolit tohoto člověka naštvat. Koneckonců, pokud je tato chyba skutečně závažná, pak tato osoba třímá v ruce informace, které mohou vašim uživatelům způsobit velké problémy. Chovejte se k takovýmto lidem slušně a doufejte, že to samé udělají i oni.

Dalším typem člověka, který často oznamuje bezpečnostní chyby, je odborník, který se kontrolou zdrojového kódu živí a sleduje novinky o softwarových bezpečnostních dírách. Takoví lidé mají obvykle velké zkušenosti s oběma stranami procesu – pravděpodobně odeslali i přijali víc zpráv o chybách než většina vývojářů ve vašem projektu. I ti vám obvykle dají datum, kdy chybu oznámí veřejně. O tomhle datu se s nimi někdy dá vyjednávat, ale to záleží na nich. V jejich profesi se to na základě dlouholetých zkušeností považuje za v podstatě jediný spolehlivý způsob, jak různé organizace donutit, aby se na chybu zaměřili přednostně. Neberte to tedy od nich jako nezdvořilost – je to vážená tradice, pro kterou existují dobré důvody.

Jakmile víte, jaká je závažnost a naléhavost chyby, můžete začít pracovat na opravě. Někdy si budete muset vybrat mezi opravou, která je elegantní, a opravou, která je rychlá – právě proto je důležité určit naléhavost problému ještě předtím, než začnete. Diskuze o chybě by se měli samozřejmě účastnit jen členové bezpečnostního mailing listu, ten, kdo chybu oznámil (pokud o to stojí) a další vývojáři, kteří se na opravě z technických důvodů musí podílet.

Opravu nezapisujte do úložiště. Držte ji ve formě záplaty do té doby, než přijde datum pro zveřejnění. Pokud byste provedli commit, a to i třeba doprovázený nějakým velmi neškodně znějícím komentářem, někdo by si opravy mohl všimnout a pochopit, k čemu slouží. Nikdy nevíte, kdo všechno vaše úložiště sleduje a z jakého důvodu. Nijak nepomůže, když vypnete e-mailové oznámení commitu – mezera v číslování těchto oznámení bude nápadná a i kdyby nebyla, data stejně v úložišti budou. Pište tedy kód do záplaty a tu uchovávejte někde na soukromém místě, třeba v odděleném, soukromém úložišti, o němž vědí jen ti, kteří vědí i o chybě. (Pokud používáte decentralizovaný systém správy verzí, jako je Arch nebo SVK, můžete práci provést v rámci tohoto systému s tím, že příslušné úložiště bude veřejnosti nepřístupné.)

Čísla CAN/CVE

U bezpečnostních chyb už jste se možná setkali s *číslem CAN* nebo *číslem CVE*. Většinou vypadají nějak jako „CAN-2004-0397“ nebo „CVE-2002-0092“.

Obě čísla reprezentují stejný typ entity, záznam v seznamu „Common Vulnerabilities and Exposures“ (Běžné bezpečnostní nedostatky a rizika), který je uchováván na **http://cve.mitre.org/**. Smyslem tohoto seznamu je poskytovat standardizované názvy pro všechny známé bezpečnostní problémy, aby bylo každému přiděleno jedinečné kanonické jméno, které lze použít v diskuzích, a aby existovalo centralizované místo, kde lze získat více informací. Jediný rozdíl mezi číslem „CAN“ a „CVE“ je v tom, že to první znamená kandidátní záznam, tedy takový, který rada CVE ještě neschválila do oficiálního seznamu, a to druhé znamená schválený záznam. Oba typy jsou ale veřejně viditelné a číslo záznamu se po schválení nemění – pouze je „CAN“ nahrazeno „CVE“.

Záznam CAN/CVE samotný neobsahuje plný popis chyby ani to, jak se před ní chránit. Místo toho obsahuje krátké shrnutí a seznam odkazů na externí zdroje (jako jsou archivy mailing listů), kde lze

získat další informace. Skutečným smyslem **http://cve.mitre.org/** je poskytnout dobře organizovaný seznam, v němž má každá chyba své jméno a kde lze získat podrobnosti. Jako příklad se můžete podívat na záznam na **http://cve.mitre.org/cgi-bin/cvename.cgi?name=2002-0092**. Všimněte si, že jeho reference jsou popsány velmi stroze a že pro zdroje používá systém těžko srozumitelných zkratek. Seznam těchto zkratek najdete na **http://cve.mitre.org/data/refs/refkey.html.**

Pokud vaše chyba splňuje kritéria CVE, můžete pro ni chtít získat číslo CAN. Proces, kterým se to dělá, je úmyslně velmi uzavřený; v zásadě to funguje tak, že musíte někoho znát, nebo znát někoho, kdo zná někoho. To není zas tak šílené, jak to možná zní. Aby nebyla rada CVE zavalena velkým množstvím pochybných nebo špatně napsaných záznamů, berou je jenom od těch zdrojů, které už znají a kterým věří. Abyste tedy dostali svou chybu na seznam, musíte najít spojení mezi lidmi ve vašem projektu a členy rady CVE. Zeptejte se mezi vývojáři – pravděpodobně se najde někdo, kdo zná někoho, kdo už tento CAN proces někdy podstoupil, nebo kdo zase zná někoho dalšího, a tak dál. Výhoda toho, když věci děláte tímto způsobem je, že někde v tomto řetězci se může objevit někdo, kdo o tom ví tolik, že vám může hned říct, že a) podle kritérií MITRE se nejedná o chybu, takže není důvod ji zasílat, nebo b) že tato chyba už své CAN nebo CVE číslo má. Varianta b) se může stát třeba tehdy, pokud už byla chyba publikována na jiném seznamu bezpečnostních expertů, například na **http://www.cert.org/** nebo na mailing listu BugTraqu na **http://www.securityfocus.com/**. (Pokud se ovšem něco takového stalo a nikdo ve vašem projektu o tom nic neslyšel, mělo by to vyvolat obavy, co se ještě může dít, o čem nic nevíte.)

Pokud tedy CAN/CVE číslo vůbec získáte, mělo by to být zkraje celého procesu analýzy chyby, aby všechna další komunikace mohla toto číslo používat. Na záznamy CAN je uvaleno do určitého data embargo. V seznamu bude existovat jen prázdný záznam (abyste nepřišli o jeho jméno), ale vůbec žádné informace o tom, v čem spočívá chyba, a to do doby, než oznámíte chybu a zveřejníte opravu.

Více informací o procesu CAN/CVE lze nalézt na **http://cve.mitre.org/cve/identifiers/candidates_explained.html**; velmi důkladný popis toho, jak tato čísla používá jeden open source projekt, lze nalézt na **http://www.debian.org/security/cve-compatibility.**

Předběžné oznámení

Jakmile váš bezpečnostní tým (tedy ti vývojáři, kteří jsou na bezpečnostním mailing listu nebo kteří byli přizváni k řešení této konkrétní chyby) dokončí opravu, je potřeba rozhodnout, jak ji distribuovat.

Pokud ji jednoduše zapíšete do úložiště nebo ji nějak jinak veřejně oznámíte, pak efektivně každého uživatele vašeho software nutíte, aby buď okamžitě aktualizoval, nebo riskoval hackerský útok. Někdy je proto vhodné několika důležitým uživatelům zaslat *předběžné oznámení*. Zvlášť to platí u software s architekturou klient/server, u nějž mohou existovat známé servery, které jsou pro útočníky lákavým cílem. Administrátoři těchto serverů ocení, pokud jim dáte den nebo dva na to aktualizaci provést, aby v době, kdy bude chyba zveřejněna, už byli chráněni.

Předběžné oznámení jednoduše znamená, že ještě předtím, než celou věc zveřejníte, pošlete těmto administrátorům e-mail, v němž jim řeknete, v čem je problém a jak jej spravit. Toto oznámení byste měli poslat jen těm, jimž věříte, že budou diskrétní. Kvalifikace pro to, získat předběžné oznámení, má tedy dvě stránky: příjemce musí spravovat velký, důležitý server, jehož napadení by byl závažný problém, a zároveň musíte vědět, že to je někdo, kdo existenci bezpečnostní chyby nevyžvaní dřív, než ji zveřejníte vy.

E-maily s předběžným oznámením posílejte příjemcům jednotlivě, jeden po druhém. V žádném případě je neposílejte najednou, aby pak neviděli v hlavičce zprávy jména těch ostatních – tím byste jim totiž všem poskytli informaci, kdo všechno má ve svém serveru bezpečnostní díru. Dobrým řešením není ani poslat tyto e-maily ve skryté kopii (BCC), protože někteří administrátoři chrání své schránky filtry proti spamu, které zprávy s BCC buď úplně blokují, nebo snižují jejich důležitost, protože spamu posílaného tímto způsobem je poslední dobou opravdu hodně.

Tady je příklad e-mailu s předběžným oznámením:

```
Od: Vaše jméno
Komu: admin@velky-znamy-server.com
Reply-to: Vaše jméno (ne adresa bezpečnostního mailing listu)
Předmět: Důvěrné oznámení o bezpečnostní chybě v Scanley

Tento e-mail je důvěrné předběžné oznámení bezpečnostního nedostatku
v serveru Scanley.

Prosím, *nepřeposílejte* žádnou část tohoto e-mailu nikomu jinému. Veřejné
oznámení bude až 19. května a do té doby bychom chtěli udržet informační
embargo.

Tento e-mail jste dostal proto, že (jak si myslíme) provozujete Scanley
server, který bychom byli rádi, kdybyste mohl záplatovat ještě před
zveřejněním chyby 19. května.

Reference:
==========

   CAN-2004-1771: Přetečení zásobníku v dotazech v Scanley

Ohrožení:
=========

   Server může být přinucen ke spouštění libovolných příkazů, pokud je
   parametr locale na serveru nastaven nesprávně a klient zašle špatně
   formátovaný dotaz.
```

```
Závažnost:
=========

   Velmi závažné; v důsledku je možné na serveru spustit libovolný kód.

Metody, jak chybu obejít:
=========================

   Pokud nastavíte v scanley.conf 'natural-language-processing' na 'off',
   chyba se nebude projevovat.

Záplata:
=======

   Záplatu níže lze aplikovat na Scanley 3.0, 3.1, a 3.2.

   Nová verze (Scanley 3.2.1) bude vydána dne 19. května nebo nedlouho před
   tímto datem, aby byla dostupná ve stejný moment, v němž chybu zveřejníme.
   Můžete tuto záplatu aplikovat hned nebo počkat na veřejný release. Jediný
   rozdíl mezi verzemi 3.2 a 3.2.1 bude právě tato záplata.

[…a sem přijde záplata…]
```

Pokud máte CAN číslo, v předběžném oznámení jej uveďte (jak je ukázáno výše), i když je na informace pořád ještě embargo a na stránkách MITRE tedy nebude nic. Díky tomu bude příjemce bezpečně vědět, že chyba, kterou jste mu oznámili, je stejná jako ta, o níž se později dozví z veřejných kanálů, takže nemusí zkoumat, jestli by měl ještě něco dělat nebo ne – což je přesně ten důvod, proč čísla CAN/CVE existují.

Distribuujte opravu veřejně

Posledním krokem v procesu, jak opravit bezpečnostní problém, je tuto opravu veřejně distribuovat. Měli byste to udělat jediným oznámením, v němž popíšete problém, uvedete číslo CAN/CVE (pokud ho máte), popíšete, jak chybu obejít a jak ji trvale opravit. Obvykle zde „opravit“ znamená aktualizovat na novější verzi software, ale někdy to znamená aplikaci záplaty, zejména pokud je tento software stejně obvykle používán ve formě zdrojového kódu. Pokud ale vydáte novou verzi, měla by se od nějaké stávající lišit právě jen tou bezpečnostní záplatou. Tímto způsobem dáváte konzervativním administrátorům možnost aktualizovat, aniž by museli mít obavy, že to bude mít nějaké vedlejší následky; také se nemusí stresovat otázkou dalších aktualizací, protože se rozumí samo sebou, že bezpečnostní záplata bude i ve všech dalších verzích. (Podrobnosti o tom, jak vydávat nové verze, najdete v části **Bezpečnostní release** v kapitole **7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji.**)

Ať už tedy vydáte novou verzi nebo ne, oznámení o chybě by mělo mít zhruba stejnou prioritu, jakou by měl i nový release: pošlete e-mail do mailing listu announce, vydejte novou tiskovou zprávu, aktualizujte záznam na Freshmeatu atd. Ačkoliv byste se rozhodně neměli pokoušet zlehčit existenci bezpečnostní chyby z obav o reputaci projektu, určitě by bylo vhodné nastavit tón bezpečnostního oznámení a jeho umístění tak, aby odpovídaly závažnosti problému. Pokud byla bezpečnostní chyba jen malá, a ne něco, co umožňuje útočníkům převzít kontrolu nad celým počítačem uživatele, pak nemusí stát za to na ni příliš upozorňovat. Můžete se dokonce rozhodnout, že s ní není třeba zatěžovat ani mailing list announce. Koneckonců pokud by se projekt pokaždé tvářil, že se děje něco mimořádně důležitého, mohli by si uživatelé začít myslet, že je mnohem méně bezpečný, než ve skutečnosti je, a také by mohli začít přistupovat k vašem oznámením skepticky – a to i v případech, kdy skutečně půjde o něco velmi závažného. Pro dobrý úvod do problematiky jak posoudit závažnost problému se podívejte na **http://cve.mitre.org/about/terminology.html**.

Obecně se dá říct, že pokud si nejste jisti, jak bezpečnostní problém řešit, najděte někoho zkušenějšího a promluvte si s ním o tom. Odhadování závažnosti problému a opravování bezpečnostních chyb je něco, co vyžaduje hodně zkušeností, a ze začátku je velmi snadné párkrát šlápnout vedle.

7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji

7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji — 191

Číslování verzí — 192
Z čeho se číslo verze skládá — **193**
Jednoduchá strategie — **195**
Strategie sudá / lichá — **196**

Release větve — 197
Jak release větve fungují — **198**

Stabilizace release — 199
Diktatura vlastníka release — **200**
Hlasování o změnách — **201**
Spolupráce na stabilizaci release a jak ji řídit — **202**
Správce release — **203**

Vytváření balíčků — 204
Formát — **204**
Jméno balíčku a adresářový strom — **205**
Velká nebo malá písmena — **207**
Pre-release — **207**
Kompilace a instalace — **208**
Binární balíčky — **209**
Testování a releasing — **210**
Release kandidát — **211**
Oznamování release — **211**

Udržování více release řad — 212
Bezpečnostní release — **213**

Vydávání releasů a každodenní práce — 214
Plánování releasů — **215**

7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji

V této kapitole se podíváme na to, jak projekty svobodného softwaru vytvářejí balíčky (packages) a vydávají nové verze (releases) a jak celkový vývoj projektu ovlivňují právě tyto dva cíle.

Hlavním rozdílem mezi projekty open source a proprietárního softwaru je to, že tým vývojářů není nijak centrálně řízen. Tento rozdíl je zvlášť vidět v situacích, kdy se připravuje nový release – u projektů, které řídí nějaká firma, může být celý tým jednoduše požádán, aby se na tento release soustředil, a tedy prozatím odložil vývoj nových funkcí a opravu drobných chyb, dokud release nebude hotový. Na to ale skupiny dobrovolníků nedrží dost pohromadě. Lidé se účastní projektu z mnoha různých důvodů a ti, kteří nemají zájem pracovat na nějakém konkrétním releasu, si stejně mohou chtít po straně vyvíjet dál na nové verzi zcela nezávisle. Protože vývoj se nikdy nezastavuje, trvá obvykle celý proces vydávání nové verze open source softwaru poněkud déle než v komerční sféře, ale zase méně narušuje chod projektu. Je to trochu podobné, jako když se opravují dálnice. Když chcete opravit silnici, máte dvě možnosti: buď ji úplně uzavřete, takže se na ni mohou všichni údržbáři vrhnout najednou a plně se jí věnovat, dokud problém nevyřeší, nebo můžete v každý daný moment pracovat jen na jednom nebo dvou pruzích a ty zbylé nechat v provozu. První způsob je velmi efektivní z pohledu těch, kdo silnici opravují, ale nikoho jiného – celá silnice je úplně uzavřená, dokud práce neskončí. Druhý způsob trvá mnohem dále a pro údržbáře je výrazně náročnější (mohou nasadit jen menší množství lidí a méně strojů, jejich podmínky jsou stísněné, někdo musí pořád stát na místě a mávat vlaječkami, aby dopravu zpomalil a navedl tím správným směrem, atd.), ale na druhou stranu zůstává silnice pořád použitelná, třebaže ne naplno.

Open source projekty obvykle fungují tím druhým způsobem. Vlastně by se dalo říct, že každý softwarový projekt v pokročilejších fázích vývoje, v němž se pracuje na několika různých release řadách najednou, je ve stavu „menších prací na silnici“ prakticky pořád. Vždy je pár pruhů uzavřeno; v pozadí projektu panuje téměř konstantní, ale celkem nízká úroveň nepohodlí, kterou skupina vývojářů jako celek toleruje, aby mohly releasy vycházet pravidelně.

Model, díky němuž je možné pracovat tímto způsobem, se ale neprojevuje jen ve vytváření releasů. Je to princip paralelizování úkolů, které na sobě vzájemně nesouvisejí – tedy princip, který rozhodně není k vidění pouze v prostředí vývoje open source softwaru, ale jímž se všechny open source projekty nějakým způsobem řídí. Nemohou si dovolit příliš komplikovat život ani údržbářům silnic, ani řidičům, ale také si nemohou dovolit ztrácet čas svých lidí tím, že budou postávat u oranžových kuželů a řídit dopravu. Přirozeně budou tedy směřovat k procesům, u nichž je úroveň administrativních zásahů držena na jedné konstantní úrovni, spíš než aby rostla a zase klesala. Dobrovolníci jsou obvykle ochotni pracovat i v podmínkách určitého menšího, ale konzistentního nepohodlí – předvídatelnost tohoto stavu jim umožňuje klidně pracovat, aniž by se museli obávat, že se jejich harmonogram nějak střetne s tím, co se právě děje v projektu. Pokud by ale byl projekt podřízen nějakému většímu časovému plánu, v němž jsou některé činnosti závislé na jiných, znamenalo by to, že velké množství času by vývojáři neměli co dělat, což by bylo nejen neefektivní, ale také nudné – a tedy i nebezpečné, protože vývojář, který se nudí, vývojářem dlouho nezůstane.

Práce na vytvoření releasu je obvykle tím nejnápadnějším z úkolů, jež sice nejsou přímo vývojem, ale probíhají s ním paralelně, takže metody, které si popíšeme v následujících oddílech, se zaměřují převážně na to, jak zajistit, aby byly releasy vydávány tak, jak by měly. Nezapomínejte ale, že stejné zásady platí i pro jiné, paralelně probíhající úkoly, jako jsou překlady a lokalizace, velké změny v API, které jsou do celého zdrojového kódu zanášeny postupně, atd.

Číslování verzí

Předtím, než si povíme, jak vytvořit release, se podíváme na to, jak jej pojmenovat. K tomu ale potřebujeme vědět, co vlastně release pro uživatele znamená. Release znamená, že:

- Staré chyby byly opraveny. To je pravděpodobně to jediné, na co se uživatelé můžou u každého release spolehnout.
- Přibyly nové chyby. I s tím se dá obvykle počítat, možná s výjimkou bezpečnostních releasů nebo jiných jednorázových aktualizací (viz **Bezpečnostní release** dále v této kapitole).
- Možná byly přidány nové funkce.
- Mohly přibýt i nové konfigurační možnosti, popřípadě se mohl trochu pozměnit význam těch starých. Postup instalace se oproti předchozímu release také mohl trochu změnit, ale při troše štěstí ne.

Mohly být také provedeny některé nekompatibilní změny, takže například datové formáty, které používaly starší verze softwaru, už není možné použít bez nějaké (možná manuálně prováděné) konverze, která funguje jen jedním směrem.

Jak vidíte, ne všechno z toho jsou dobré věci. Proto zkušení uživatelé přistupují k novým releasům obezřetně, zejména pokud se jedná o software, který už funguje nějakou dobu a už převážně dělá to, co chtějí (nebo co si myslí, že chtějí). Dokonce i přidání nových funkcí nemusí být zas tak vítaná událost, protože může znamenat, že se software začne chovat novým, nečekaným způsobem.

Pro číslování verzí tedy jsou dva hlavní důvody – jednak by měla čísla jednoznačně určovat jejich vzájemné pořadí (tedy abychom při pohledu na dvě čísla verzí byli schopni okamžitě určit, která z nich je starší), ale také by měla zhuštěným způsobem podávat informaci o tom, jaký typ změn a jak závažných nový release obsahuje.

To vše že se dá vtěsnat do jediného čísla? V zásadě ano. Číslování verzí je zrovna dobrý příklad přístřešku na kola, který už byl natírán mnohokrát (viz část **Čím jednodušší téma, tím delší debata** v kapitole **6. Komunikace**), a zdá se velmi nepravděpodobné, že by se někdy v dohledné době celý svět shodl na jednom standardním způsobu. Nicméně se časem vyvinulo několik dobrých strategií a jeden obecně uznávaný princip: *buďte konzistentní.* Zvolte si způsob číslování, zdokumentujte jej a pak se jej držte. Vaši uživatelé vám poděkují.

Z čeho se číslo verze skládá

V tomto oddílu si popíšeme formální zvyklosti číslování releasů do podrobností, a to od samých základů. Zařazujeme jej sem především pro úplnost. Pokud už tyto zvyklosti znáte, klidně tento oddíl přeskočte.

Čísla verzí jsou skupiny číslic oddělených tečkami:

Scanley 2.3
Singer 5.11.4

…a tak dále. Tato čísla nejsou desetinné tečky, ale pouhé oddělovače; po „5.3.9“ bude následovat „5.3.10“. Některé projekty se to občas pokusily dělat trochu jinak, nejslavněji asi jádro Linuxu, které používalo sekvenci „0.95“, „0.96“ až „0.99“, jež končila u Linuxu 1.0, ale konvence, že tečky neoddělují desetinná místa, je používána téměř všude a můžete ji považovat za standard. Počet jednotlivých složek čísla verze (tedy číslic mezi tečkami) není nijak omezen, ale většinou nepřekračuje tři nebo čtyři. K důvodům, proč tomu tak je, se dostaneme za chvilku.

Vedle číselných částí se objevuje někdy i slovní popis, například „Alpha“ nebo „Beta“ (viz **Alfa a beta** v části **Stav vývoje** v kapitole **2. Zahájení projektu**), například:

Scanley 2.3.0 (Alpha)
Singer 5.11.4 (Beta)

Dodatek Alpha nebo Beta znamená, že tento release předchází budoucímu release, který bude mít stejné číslo verze, ale už bez tohoto kvalifikátoru. Takže verze „2.3.0 (Alpha)“ časem povede k verzi „2.3.0“. Aby bylo možné takových kandidátních verzí vydat více za sebou, mohou mít tyto kvalifikátory ještě vlastní metakvalifikátory. Takhle například může vypadat úplná řada releasů v pořadí, v jakém se dostávaly na veřejnost:

Scanley 2.3.0 (Alpha 1)
Scanley 2.3.0 (Alpha 2)
Scanley 2.3.0 (Beta 1)
Scanley 2.3.0 (Beta 2)
Scanley 2.3.0 (Beta 3)
Scanley 2.3.0

Všimněte si, že když má verze kvalifikátor „Alpha“, píše se Scanley „2.3.0“ místo „2.3“. Tato dvě čísla jsou zcela ekvivalentní – nulové složky na koncích lze pro zjednodušení vynechat –, ale pokud už verze obsahuje i kvalifikátor, tak už se stejně o zjednodušování nedá mluvit, takže klidně můžeme pro úplnost připsat i tu nulu navíc.

Další kvalifikátory, s nimiž se můžete poměrně často setkat, jsou „Stable“ (stabilní), „Unstable“ (nestabilní), „Development“ (vývojová) a „RC“ (zkratka „Release Candidate“, tedy kandidátní release). Nejčastěji používané jsou jednoznačně „Alpha“ a „Beta“ a v těsném závěsu za nimi „RC“. Povšimněte si ale, že k „RC“ se vždy ještě přidává číselný metakvalifikátor. Obvykle tedy nevydáte verzi „Scanley 2.3.0 (RC)“; takový release se bude místo toho jmenovat „Scanley 2.3.0 (RC 1)“, po něm přijde RC2 atd.

Tato tři označení, tedy „Alpha“, „Beta“ a „RC“, jsou dnes už obecně známá a použití těch ostatních bych nedoporučoval, třebaže se na první pohled mohou zdát lepší, protože to jsou přece jen normální slova a ne žargon. Ale lidé, kteří z releasů instalují software, už tuto trojici dobře znají, a není žádný důvod dělat jen tak z rozmaru věci jinak než všichni ostatní.

Jak už jsme řekli, tečky v číslech verzí neoznačují desetinná místa, ale něco trochu podobného: oddělují pozice s odlišným významem. Všechny releasy číslované „0.X.Y“ předcházejí verzi „1.0“ (což je samozřejmě totéž jako „1.0.0“). „3.14.158“ patří těsně před „3.14.159“; zrovna tak patří, i když už ne těsně, před „3.14.160“, „3.15.cokoliv“ a tak dál.

Konzistentní systém číslování verzí uživatelům umožňuje, aby při pohledu na dvě čísla verzí okamžitě poznali, a to z ničeho jiného než z čísel samotných, jak velký rozdíl mezi těmito dvěma releasy je. V klasickém systému se třemi složkami označuje první část *velké číslo* (major), druhá *malé číslo* (minor) a třetí *mikročíslo*. Takže například verze číslo „2.10.17“ je sedmnáctým mikrorelease v desáté řadě malých release a v druhé sérii velkých release. Slova „řada“ a „série“ v předchozí větě neberte jako termíny, ale intuitivní popis situace. Velká série sdružuje všechny releasy, které mají stejné velké číslo, a malá série (nebo malá řada) obsahuje všechny releasy, které mají stejné malé i velké číslo. To znamená, že „2.4.0“ a „3.4.1“ nepatří do stejné malé řady, třebaže obě mají malé číslo „4“; na druhou stranu pak „2.4.0“ a „2.4.2“ do stejné malé řady patří, třebaže na sebe přímo nenavazují, protože je mezi nimi ještě „2.4.1“.

Význam těchto čísel je přesně takový, jaký byste čekali – zvýšení velkého čísla znamená, že byly provedeny velké změny, zvýšení malého čísla znamená, že byly provedeny malé změny a konečně zvýšení mikročísla znamená, že bylo změněno jen pár maličkostí. Některé projekty přidávají ještě čtvrtou složku, obvykle označovanou jako *číslo záplaty* (patch), používanou pro zvlášť jemné rozlišování mezi verzemi (některé projekty ale poněkud matoucně používají ve svých systémech s třemi složkami „patch“ jako synonymum pro „micro“). Některé projekty pak zase používají poslední složku jako *číslo sestavení* (build), které se zvyšuje pokaždé, když software nově sestavíte, a neohlašuje žádnou jinou změnu než právě nové sestavení. Díky tomu může být v projektu každý bug report propojen s konkrétním sestavením, což je asi nejužitečnější tam, kde k distribuci nejčastěji používáte binární balíčky.

Ačkoliv pro to, jak se jednotlivé složky používají a co přesně znamenají, existuje řada různých zvyklostí, nijak výrazně se od sebe zpravidla neliší – trochu volnosti v tom sice je, ale ne moc. V dalších dvou oddílech se podíváme na ty nejběžnější metody.

Jednoduchá strategie

Většina projektů má svá pravidla, která předepisují, jaké změny lze zahrnout do releasu, který zvyšuje jen mikročíslo; podobně pak mají i zvláštní pravidla pro malá i velká čísla. Tato pravidla opět nejsou standardizována; popíšu tu ale jeden systém, který úspěšně používá řada projektů. Můžete jej využít i ve svém vlastním projektu, ale samozřejmě nemusíte. V každém případě je to dobrý příklad toho, jaké informace by čísla verzí měla obsahovat. Tento systém je upravenou verzí systému, který použil projekt APR, viz **http://apr.apache.org/versioning.html.**

1. Změny v mikročísle (tedy změny v rámci stejné malé linie) musí být plně kompatibilní, tedy jak zpětně, tak dopředně. To znamená, že by se mělo jednat jen o opravy chyb nebo drobná zlepšení stávajících funkcí. Mikroreleasy by neměly obsahovat nové funkce.
2. Změny v malém čísle (tedy změny v rámci stejné velké linie) musí být kompatibilní zpětně, ale už ne nutně dopředně. Malé releasy často přinášejí nové funkce, ale obvykle ne velké množství funkcí najednou.
3. Změny velkého čísla pak vymezují hranice kompatibility. Nový velký release může být nekompatibilní jak dopředu, tak zpět. U velkého release se předpokládá, že bude obsahovat nové funkce, možná i celé větší skupiny nových funkcí.

Co přesně termíny *zpětně kompatibilní* a *dopředně kompatibilní* znamenají, hodně závisí na tom, co váš software dělá, ale v konkrétním kontextu je to obvykle celkem jednoznačné. Například pokud je váš projekt aplikace typu klient/server, pak „zpětně kompatibilní“ znamená, že když aktualizujete server na verzi 2.6.0, nemělo by to pro klienty verze 2.5.4 znamenat, že přijdou o nějakou funkci nebo že se software bude chovat jinak než dřív (samozřejmě kromě toho, že budou opraveny některé chyby). Na druhou stranu pokud aktualizujete jednoho z těchto klientů i server na verzi 2.6.0, může to znamenat, že tento klient bude mít k dispozici nové funkce, které klienti verze 2.5.4 neumí využít. V takovém případě se nejedná o aktualizaci, která by byla „kompatibilní dopředně“ – je jasné, že teď už se nemůžete vrátit zpátky ke klientovi verze 2.5.4 a využívat v něm funkcí verze 2.6.0, protože v něm ještě nejsou.

Z tohoto důvodu se mikroreleasy používají prakticky výhradně pro opravy chyb. Musí totiž zůstat kompatibilní oběma směry – pokud aktualizujete z 2.5.3 na 2.5.4, ale pak si to rozmyslíte a vrátíte se k 2.5.3, neměli byste o žádné funkce přijít. Samozřejmě by se tak vrátily všechny chyby, které byly ve verzi 2.5.4 opraveny, ale neztratily by se žádné funkce, tedy kromě těch, které ve starší verzi sice existovaly, ale kvůli chybám opraveným v 2.5.4 nefungovaly správně.

Protokoly mezi klienty a serverem jsou jen jednou z oblastí, kde se kompatibilita může projevovat. Další jsou datové formáty – zapisuje software data na nějaké trvalé úložiště? Pokud ano, pak musí formáty dat, které je software schopen číst a zapisovat, odpovídat zásadám, které řídí číslování verzí. Verze 2.6.0 musí být schopna přečíst soubory, které vytvořila verze 2.5.4, ale přitom je může potichu aktualizovat na novější formát, který už 2.5.4 nepřečte – protože zpětná kompatibilita se mezi malými verzemi

nevyžaduje. Pokud váš projekt distribuuje knihovny kódu pro použití v jiných programech, pak se kompatibilita týká i API – musíte se ujistit, že pravidla pro kompatibilitu zdrojového kódu i binárních souborů jsou natolik podrobná, aby poučený uživatel nemusel přemítat, zda je aktualizace bezpečná nebo ne. Jednoduše se podívá na čísla a hned se to dozví.

V tomto systému nemáte možnost začít s něčím od začátku, dokud nezvednete velké číslo. To může být někdy dost na obtíž – třeba když chcete přidat nějaké funkce nebo upravit protokoly, ale nemůžete, protože byste tím narušili kompatibilitu. Na to neexistuje žádné kouzelné řešení kromě toho, že budete od začátku projekt navrhovat tak, aby se dal snadno rozšiřovat (což je ale téma, které by snadno vydalo na vlastní knihu a jež rozhodně značně přesahuje záměr této knihy). Nicméně to, že nastavíte zásady pro vydávání releasů a pak se jimi budete řídit, je něco, čemu se při distribuci softwaru nemůžete vyhnout. Jedno nepříjemné překvapení vás může připravit o mnoho uživatelů. Zásady, které jsme tu popisovali, jsou dobré jednak proto, že jsou celkem rozšířené, ale také proto, že je poměrně jednoduché je vysvětlit a zapamatovat si je, a to i pro ty, kteří o nich předtím nikdy neslyšeli.

Další obecně uznávané pravidlo je, že tyto zásady neplatí pro verze nižší než 1.0 (třebaže to byste ve své dokumentaci pravděpodobně také měli pro jistotu zdůraznit). Projekt, který teprve zahájil vývoj, může postupně vydávat releasy 0.1, 0.2, 0.3 a tak dále hned za sebou, dokud nebude hotová verze 1.0, a rozdíly mezi těmito dílčími verzemi mohou být různě veliké. Mikročísla se u releasů nižších než 1.0 někdy vynechávají. V závislosti na tom, jaký váš projekt je a jaké rozdíly mezi těmi jednotlivými verzemi budou, se může ukázat, že by se vám víc hodilo číslování 0.1.0, 0.1.1 atd., ale nemusí. Zvyklosti pro čísla nižší než 1.0 jsou celkem uvolněné, především proto, že všichni chápou, že pokud byste se omezovali otázkami kompatibility, mohlo by to prvotní vývoj příliš zpomalit, a také proto, že lidé, kteří používají takto rané verze, obvykle bývají dost tolerantní.

Pamatujte si, že tato pravidla platí jen pro tento konkrétní systém se třemi složkami. Ve svém projektu si klidně můžete vymyslet vlastní systém se třemi složkami, nebo se můžete rozhodnout, že takto podrobné dělení vůbec nepotřebujete a že vám budou stačit složky dvě. Důležité je ale tato rozhodnutí udělat hned zkraje, zveřejnit, co přesně číslování verzí znamená, a pak se toho držet.

Strategie sudá / lichá

Některé projekty využívají malého čísla k tomu, aby vyjádřily stabilitu softwaru: sudé číslo znamená stabilní verzi, liché nestabilní. To se týká jenom malých čísel, ne velkých a mikro. Zvýšené mikročíslo stále znamená opravu chyb (žádné nové funkce) a zvýšení velkého čísla pořád znamená velké změny, nové sady funkcí atd.

Výhodou systému sudá / lichá, který používal například projekt jádra Linuxu, je to, že umožňuje vydat nové funkce k otestování, aniž byste vystavili uživatele, kteří software nasazují v ostrém provozu, potenciálně nestabilnímu kódu. Z čísel je na první pohled patrné, že verzi „2.4.21“ lze klidně nainstalovat na běžící webserver, ale že verze „2.5.1“ slouží jen k experimentům na domácích stanicích uživatelů. Vývojový tým zpracovává bug reporty, které přicházejí z nestabilní řady (tedy té s lichými čísly),

a jakmile se začne po několika mikroreleasech ze stejné řady situace zklidňovat, zvýší malé číslo (aby bylo sudé), mikročíslo resetují na nulu a vydají balíček, o němž se předpokládá, že je stabilní.

Tento systém zachovává stejné principy, jaké jsme popsali výše, nebo s nimi přinejmenším není v rozporu. Pouze dodává malému číslu trochu informací navíc. Díky tomu se pak malé číslo zvyšuje dvakrát častěji, než by jinak bylo nutné, ale to ničemu nevadí. Systém sudá / lichá je asi nejlepší pro projekty, jejichž release cyklus je spíše pomalejší a které už ze své podstaty mají velké množství konzervativních uživatelů, kteří více ocení stabilitu než nové funkce. Není to ale jediný způsob, jak nějaké funkce nechat otestovat v reálném provozu. V části **Stabilizace release** dále v této kapitole popisuje další, možná běžnější způsob, jak zveřejnit potenciálně nestabilní kód, samozřejmě s příslušným označením, aby všichni už na první pohled poznali, jaká jsou jeho potenciální rizika a výhody.

Release větve

Z hlediska vývojářů je projekt svobodného softwaru ve stavu nepřetržitých release. Vývojáři obvykle spouštějí ten nejnovější kód, protože v něm chtějí nalézt chyby a protože projekt sledují natolik pečlivě, aby věděli, které oblasti funkcí jsou zrovna v nestabilním stavu, a oni by se jim tedy měli vyhnout. Často aktualizují svou kopii softwaru každý den, někdy i několikrát denně, a pokud zapíšou nějakou změnu, není nerozumné očekávat, že ji všichni ostatní vývojáři do 24 hodin budou mít.

Jak tedy má projekt vytvořit formální release? Měl by jednoduše vzít snímek celého stromu v daný moment, zabalit jej a předat světu s tím, že je to verze řekněme „3.5.0“? Zdravý rozum říká, že ne. Jeden důvod je, že nemusí existovat okamžik, v němž by byl celý vývojový strom vyčištěný a připravený na vydání. Mohou se v něm povalovat čerstvě přidané funkce, které ještě nejsou úplně hotové. Někdo mohl zapsat velkou změnu, která opravuje nějakou chybu, ale tato oprava je třeba kontroverzní a v době vytvoření snímku se prodiskutovává. V takovém případě by nebylo efektivní vytvoření snímku oddalovat, dokud diskuse neskončí, protože by mezitím mohla začít jiná, nijak nesouvisející, a pak byste museli čekat zase na ni. A tento cyklus nemusí skončit nikdy.

Navíc také platí, že použití snímků celého stromu pro vytvoření release by mohlo narušit probíhající vývoj i tehdy, kdy by tento strom bylo možné do zveřejnitelného stavu dostat. Řekněme, že tento snímek bude „3.5.0“. Předpokládá se tedy, že další snímek bude „3.5.1“ a bude obsahovat převážně opravy chyb, které budou nalezeny v release 3.5.0. Pokud ale budou oba tyto snímky pocházet ze stejného stromu, co mají mezi těmito dvěma releasy dělat vývojáři? Přidávat nové funkce nemohou kvůli pravidlům kompatibility, ale opravování chyb v kódu 3.5.0 zase nebude práce, která by někoho nadchla. Někdo má třeba rozdělané nové funkce, které by chtěl dokončit, a jenom ho otrávíte, když ho budete nutit, aby buď nedělal nic, nebo pracoval na něčem, co ho nebaví, a to jenom proto, že procesy vedoucí k releasům žádají, aby vývojový strom zůstal v nepřirozeně strnulém stavu.

Řešením těchto problémů je vždy používat *release větev*. Taková větev je jen jednou z větví v systému správy verzí (viz část **větev (branch)** v kapitole **3. Technická infrastruktura**), na níž lze izolovat kód, určený pro konkrétní release, od hlavního vývojového proudu. Koncept vývojových větví rozhodně

není něco, co by využíval pouze svobodný software – používá je i mnoho komerčních vývojářských společností. V komerčním prostředí se ale někdy tyto větve považují za jistý luxus – jakýsi formální „osvědčený postup“, který ale v situacích, kdy se blíží významný termín, lze odložit stranou, zatímco se celý tým snaží stabilizovat hlavní strom.

V open source projektech jsou ale release větve téměř nepostradatelné. Viděl jsem několik projektů, které je při vytváření releasů nepoužívaly, ale vždy to vedlo k tomu, že někteří vývojáři nedělali nic, zatímco jiní – a obvykle jich bylo méně – se snažili release dokončit. To ovšem nevede k dobrým výsledkům, a to hned z několika důvodů. Za prvé tak zpomalíte průběh celého vývoje. Za druhé je tento release pak méně kvalitní, než by mohl být, protože na něm pracovalo jen pár lidí, kteří navíc spěchali, aby se mohli ostatní vrátit ke své práci. Za třetí taková situace psychologicky štěpí celý váš tým, protože staví jednotlivé pracovní činnosti do zbytečných vzájemných konfliktů. Vývojáři, kteří nedělají nic, by pravděpodobně byli ochotní k doladění release větve něčím přispět, pokud by si mohli vybrat na základě toho, kolik na to mají času a co je zajímá. Když ale taková větev neexistuje, otázka pro ně zní „Budu se dnes projektu účastnit, nebo ne?“, místo toho, aby zněla „Budu dnes pracovat na releasu, nebo na té nové funkci, kterou vyvíjím v hlavním kódu?“

Jak release větve fungují

Přesný postup pro vytvoření release větve samozřejmě závisí na vašem systému pro správu verzí, ale v hrubých rysech je všude téměř stejný. Větev se obvykle zakládá z nové větve nebo z kmenu. Tradičně se jako kmen označuje ta část, v níž probíhá hlavní vývoj kódu, bez ohledu na jakákoliv omezení releasů. První release větev, tedy ta, která povede k verzi „1.0“, se vyděluje z kmene. V CVS by příkaz pro vytvoření větve vypadal asi takto

```
$ cd trunk-working-copy
$ cvs tag -b RELEASE_1_0_X
```

zatímco v Subversion zhruba takhle:

```
$ svn copy http://.../repos/trunk http://.../repos/branches/1.0.x
```

(Ve všech příkladech předpokládám systém číslování verzí s třemi složkami. Protože nemůžu uvádět přesné příkazy pro všechny systémy správy verzí, budu používat jako příklad CVS a Subversion a doufat, že pro ostatní systémy se to dá z těchto dvou nějak odvodit.)

Všimněte si, že jsme vytvořili větev „1.0.x“ (a je tam skutečně „x“) místo „1.0.0“. To proto, že stejná mikrořada – a tedy stejná větev – bude využita pro všechny mikrorelease této řady. To, jak přesně probíhá stabilizace větve pro release, probereme v části **Stabilizace release** dále v této kapitole. V tomto oddílu nás zajímá, pouze jak fungují vzájemné interakce mezi procesem vytváření release a systémem správy verzí. Když je release větev stabilizovaná a připravená, je na čase z této větve udělat snímek použitím tagu:

```
$ cd RELEASE_1_0_X-working-copy
$ cvs tag RELEASE_1_0_0
```

nebo

```
$ svn copy http://.../repos/branches/1.0.x http://.../repos/tags/1.0.0
```

Tento tag pak reprezentuje přesný stav zdrojového stromu projektu pro release 1.0.0 (což se může hodit, pokud někdy někdo bude potřebovat získat starou verzi poté, co už byly její distribuční balíčky a binární soubory smazány). Další mikrorelease ve stejné řadě je také vytvářen na větvi 1.0.x; když je hotov, je vytvořen tag pro 1.0.1. To samé pak provedete s 1.0.2 a tak pořád dál. Až bude na čase začít pomýšlet na release 1.1.x, vytvoříte z kmene novou větev:

```
$ cd trunk-working-copy
$ cvs tag -b RELEASE_1_1_X
```

nebo

```
$ svn copy http://.../repos/trunk http://.../repos/branches/1.1.x
```

Údržba pak může pokračovat na 1.0.x a 1.1.x paralelně a releasy můžete vydávat z obou řad nezávisle na sobě. To, že nedlouho po sobě vyjdou releasy ze dvou různých řad, není ve skutečnosti nijak neobvyklé. Starší řada je určena konzervativnějším administrátorům, kteří se na ten velký skok na verzi (řekněme) 1.1 chtějí důkladně připravit. Ti odvážnější si pak obvykle stáhnou poslední release z nejvyšší řady, aby dostali ty nejnovější funkce, i když tím mohou riskovat nižší stabilitu.

To ale samozřejmě není jediná strategie pro vytváření release větví. V některých situacích to ani nemusí být ta nejlepší, třebaže v těch projektech, jichž jsem se účastnil, fungovala docela pěkně. Použijte jakoukoliv strategii, která vám přináší dobré výsledky, ale nezapomeňte na to nejdůležitější: smyslem release větve je izolovat práci na releasech od každodenního vývoje a dát celému projektu něco hmatatelného, kolem čeho lze release proces organizovat. Tento proces samotný si podrobně popíšeme v dalším oddílu.

Stabilizace release

Stabilizace je proces, jímž release větev dostanete do stavu, který lze vydat. To znamená, že je potřeba rozhodnout, které změny budou do release zařazeny a které ne, a podle toho upravit obsah větve.

Pod slovem „rozhodnout“ se tady často skrývá mnoho trápení. V softwarových projektech založených na spolupráci je velmi běžnou reakcí vývojářů divoké přidávání funkcí na poslední chvíli – když vidí, že se blíží release, začnou usilovně dodělávat změny, které mají rozpracované, aby jim neujel vlak.

Což je ale samozřejmě přesně to, co v době dokončování release vůbec nechcete. Bylo by mnohem lepší, kdyby na těchto funkcích pracovali tempem, jaké jim vyhovuje, a nestresovali se tím, zda je stihnou přidat do tohoto releasu nebo až do dalšího. Čím více změn se pokusíte do nové verze nacpat na poslední chvíli, tím více kód destabilizujete a tím více do něj také obvykle dostanete chyb.

Většina vývojářů se v teoretické rovině shodne na přibližných kritériích, které určí, jaké typy změn lze v release řadě v průběhu stabilizace provést. Samozřejmě lze přidávat opravy závažných chyb, zejména těch, které nelze nijak obejít. Aktualizace dokumentace jsou také v pořádku, stejně jako opravy chybových zpráv (s výjimkou situací, kdy jsou považovány za součást rozhraní a musí zůstat neměnné). V mnoha projektech se při stabilizaci také povolují některé změny, které buď představují jen malé riziko, nebo se týkají něčeho okrajového; pro to, jak se určí úroveň rizika, mohou mít i formální postupy. Nicméně žádná formalizace nedokáže úplně nahradit lidský úsudek. Vždy budou existovat případy, kdy se projekt zkrátka musí rozhodnout, jestli danou změnu do release zařadit nebo ne. Nebezpečí je v tom, že protože každý bude chtít, aby se do release dostaly jeho oblíbené novinky, budou všichni motivováni k tomu, aby změny spíše přijímali, a jen velmi málo lidí je bude odmítat.

Proces stabilizace release tedy převážně spočívá v tom, vytvořit mechanismy pro to, jak říct „ne“. V open source projektech zvlášť je tedy vaším cílem najít způsoby, jak říct „ne“, které nebudou mít za následek, že se budou vaši vývojáři cítit ublížení nebo zklamaní, a které také nebudou zbytečně zabraňovat tomu, aby se do release dostaly změny, které si to zaslouží. To se dá dělat mnoha způsoby. Navrhnout systém, který tato kritéria dokáže splnit, je poměrně lehké; stačí, aby se na ně tým zaměřil jako na ty zásadní. V dalším oddílu krátce popíšu dva systémy, které patří mezi ty nejoblíbenější. Tyto systémy tvoří opačné konce poměrně širokého spektra možností; pokud chcete, můžete být ve svém projektu kreativnější. Variant se nabízí opravdu hodně – tohle jsou jenom dva příklady, s nimiž jsem se setkal v praxi a které fungovaly.

Diktatura vlastníka release

Skupina se shodne na tom, že jedna osoba bude jmenována vlastníkem release. O tom, které změny budou zahrnuty, pak rozhoduje vlastník příslušného release a jeho rozhodnutí jsou konečná. Samozřejmě je běžné a očekává se, že se o změnách bude i diskutovat, ale nakonec je to vždy vlastník, kdo získal od skupiny oprávnění tuto debatu uzavřít. Aby tento systém fungoval, je potřeba vybrat někoho, kdo má dostatek technických znalostí na to, aby všechny změny dokázal pochopit, a zároveň má ve skupině dostatečně dobrou pozici a umí natolik zacházet s lidmi, aby dokázal diskuse vedoucí k releasu řídit tak, aby se pokud možno nikdo necítil ublíženě.

Často vlastník releasu řekne: „Nemyslím si, že by na této změně bylo něco špatně, ale ještě jsme neměli dost času ji prozkoušet, takže by do tohoto release přijít neměla.“ Velmi pomáhá, pokud je vlastníkem releasu někdo, kdo projektu po technické stránce opravdu dobře rozumí a dokáže odůvodnit, proč by mohla nějaká změna kód potenciálně destabilizovat (například kvůli své provázanosti s jinými částmi softwaru nebo z obav o to, zda půjde přenést na jiné platformy). Lidé pak někdy budou chtít tato

rozhodnutí odůvodnit nebo budou tvrdit, že nějaká změna není zas tak riskantní, jak vypadá. Takové debaty nemusí být nutně konfliktní, pokud dokáže vlastník release všechny argumenty posoudit objektivně a nebude reagovat tak, že se automaticky vzepře.

Rád bych zdůraznil, že pro roli vlastníka release nemusíte nutně vybrat vedoucího projektu (pokud tedy váš projekt nějakého vedoucího vůbec má – viz **Benevolentní diktátoři** v kapitole **4. Společenská a politická infrastruktura**). Někdy dokonce bude mnohem lepší, když to bude někdo jiný. Schopnosti, které potřebuje dobrý vedoucí vývojář, nemusí být nutně stejné jako ty, jež dělají dobrého vlastníka release. U něčeho tak důležitého, jako je proces vytváření release, může být rozumné mít jednu osobu, která bude fungovat jako protiváha k rozhodnutím vedoucího projektu.

Roli vlastníka release si porovnejte s méně diktátorskou rolí, která je popsána v části **Správce release** dále v této kapitole.

Hlasování o změnách

Na zcela opačném konci spektra než diktatura vlastníka release je možnost nechat vývojáře o tom, které změny budou do release zahrnuty, jednoduše hlasovat. Nicméně vzhledem k tomu, že nejdůležitější funkcí stabilizace release je z něj více změn vyloučit než přidat, je potřeba navrhnout hlasovací systém tak, aby zařazení změny muselo aktivně schválit větší množství vývojářů. Začlenění změny do release by mělo vyžadovat víc než prostý souhlas většiny (viz **Kdo hlasuje?** v kapitole **4. Společenská a politická infrastruktura**). V takovém případě by totiž pro zařazení změny stačil jeden hlas pro a žádný proti, což by mělo za následek vytvoření nešťastného stavu, kdy by každý vývojář dal hlas svým vlastním změnám, ale přitom byl velmi neochoten hlasovat proti změnám ostatních z obav, že by mu to pak vrátili. Abyste se tomu vyhnuli, musíte celý systém nastavit tak, že bude potřeba spolupráce více vývojářů najednou, aby se nějaká změna do release dostala. To bude mít za následek jednak to, že každou změnu bude revidovat více lidí, jednak budou jednotliví vývojáři méně váhat, než dají něčemu negativní hlas, protože budou vědět, že si to nikdo z těch, kdo hlasovali pro, nebude brát osobně. Čím více lidí do debaty zapojíte, tím méně se diskuse týká jednotlivců a tím více se soustředí na změnu samotnou.

Systém, který používáme v projektu Subversion, se časem ukázal jako celkem šťastné řešení, takže jej doporučím i zde. Aby se mohla nějaká změna dostat do release větve, jsou potřeba alespoň tři hlasy pro a žádný hlas proti. Jediný hlas říkající „ne“ znamená, že změna zahrnuta nebude – v tomto případě je hlasování „ne“ totéž, co veto (viz část **Veto** v kapitole **4. Společenská a politická infrastruktura**). Samozřejmě je nutné každý takový hlas nějak odůvodnit; pokud by mělo větší množství lidí pocit, že je toto odůvodnění nerozumné, teoreticky by bylo možné jej zpochybnit a přehlasovat. V praxi se to ovšem nikdy nestalo a nepředpokládám, že by k tomu kdy došlo. Lidé jsou, co se týče releasů, obvykle dost konzervativní, a pokud má někdo na určitou změnu natolik vyhraněný názor, že ji vetuje, pak k tomu obvykle má dobrý důvod.

Protože je celý systém úmyslně nastaven tak, aby tyto konzervativní sklony podporoval, jsou odůvodnění negativních hlasů někdy spíše procedurálního než technického rázu. Například může mít někdo pocit, že třebaže je příslušná změna dobře napsaná a pravděpodobně nevyvolá žádné nové chyby, do mikrorelease by zařazena být neměla, protože je zkrátka příliš velká – třeba přidává novou funkci nebo je v ní nějaká maličkost, která překračuje pravidla pro kompatibilitu. Párkrát jsem viděl, jak nějaký vývojář vetoval změnu proto, že měl instinktivní pocit, že je ji ještě potřeba dál testovat, třebaže v ní žádné chyby nenašel. Lidé sice trochu protestovali, ale veto nikdo nenapadl a změna do release zařazena nebyla (už si ale nepamatuji, jestli se v ní nakonec nějaké chyby při testování našly nebo ne).

Spolupráce na stabilizaci release a jak ji řídit

Pokud se v projektu rozhodnete o změnách hlasovat, je klíčové, aby byl systém sestavení hlasovacích lístků a samotného hlasování co nejjednodušší. Třebaže existuje řada open source softwaru pro hlasování, v praxi se jako nejjednodušší ukazuje do release větve přidat textový soubor se jménem `STATUS` nebo `HLASOVANI` nebo něco takového. V tomto souboru se nachází seznam všech navrhovaných změn (navržení změny může provést kterýkoliv vývojář) a všechny hlasy pro a proti plus další poznámky a komentáře. (Mimochodem když navrhujete změnu, nemusíte pro ni hned hlasovat, i když tomu tak často je.) Záznam v takovém souboru pak může vypadat třeba takhle:

```
* r2401 (issue #49)
  Zabránit tomu, aby handshake mezi klientem a serverem probíhal dvakrát.
  Odůvodnění:
    Předchází zbytečnému zatěžování sítě; je to malá změna, snadno
    revidovatelná.
  Poznámky:
    Tento problém byl prodiskutován v http://.../mailing-lists/message-7777.html
    a dalších zprávách ve stejném vláknu.
  Hlasy:
    +1: jsmith, kimf
    -1: tmartin (nekompatibilní s některými servery na verzích pod 1.0;
                 je sice pravda, že v těch serverech jsou chyby, ale proč
                 být nekompatibilní, když nemusíme?)
```

V tomto případě získala změna dva pozitivní hlasy, ale vetoval ji tmartin, který své odůvodnění napsal do závorky. Na přesném formátování tohoto zápisu nezáleží, můžete si klidně v projektu vymyslet něco jiného – třeba by mělo být vysvětlení od tmartina zařazeno do oddílu „Poznámky:“, nebo by měl vytvořit další oddíl s hlavičkou „Popis“, aby to lépe odpovídalo zbytku záznamu. Důležitější je, aby byly všechny informace, které k vyhodnocení změny potřebujete, dobře dostupné a aby byl mechanismus pro hlasování co nejjednodušší. Navrhovaná změna je označena číslem revize v úložišti (v tomto případě je to jen jedna revize, r2401, ale zrovna tak může změna zahrnovat několik dílčích revizí). Rozumí se samo sebou, že revize odkazuje na změnu v kmeni – pokud by tato změna byla už v release

větvi, nebyl by důvod o ní hlasovat. Pokud ve vašem systému pro správu verzí neexistuje jednoznačný způsob, jak odkázat na jednotlivé změny, měli byste si v projektu nějaký vymyslet. Aby bylo hlasování praktické, mělo by být možné každou změnu jednoznačně identifikovat.

Ti, kdo změnu navrhují nebo pro ni hlasují, ručí za to, že ji lze na release větev aplikovat bez vytvoření konfliktů (viz část **konflikt (conflict)** v kapitole **3. Technická infrastruktura**). Pokud ke konfliktům dojde, pak by měl záznam buď odkázat na záplatu, která je vyřeší, nebo na dočasnou větev, v níž se nachází upravená podoba změny, například takto:

```
* r13222, r13223, r13232
   Přepsání libsvn_fs_fs algoritmu pro automatické spojování
   Odůvodnění:
     nepřijatelně pomalé (>50 minut u malého commitu)
     v úložišti s 300 000 revizemi
   Větev:
     1.1.x-r13222@13517
   Hlasy:
     +1: epg, ghudson
```

Tento příklad pochází ze života – vzal jsem jej ze souboru `STATUS`, vytvořeného pro release Subversion 1.1.4. Všimněte si, že původní revize jsou zde použity jako kanonická označení změny, třebaže existuje i větev s verzí bez konfliktů (tato větev také slučuje tři revize kmene do jedné, r13517, aby bylo přidání této změny do releasu jednodušší – pokud tedy bude schválena). Původní revize jsou zde uvedeny proto, že se nejsnadněji revidují, neboť u sebe stále ještě mají původní zprávy z logu. Dočasná větev tyto zprávy mít nebude – aby se předešlo duplikování informací (viz část **Jedinečnost informace** v kapitole **3. Technická infrastruktura**), záznam v logu této větve pro r13517 bude obsahovat pouze „Úprava r13222, r13223 a r13232 pro zpětné napojení na větev 1.1.x.“ Všechny ostatní informace o změnách lze nalézt u původních revizí.

Správce release

Samotný proces slučování (viz **merge** v části **Slovníček pojmů správy verzí** v kapitole **3. Technická infrastruktura**) schválených změn s release větví může provést kterýkoliv vývojář. Není potřeba dát to na starost jedinému člověku – pokud je těch změn víc, bude lepší je rozdělit mezi víc osob.

Nicméně přestože je hlasování i slučování decentralizované, v praxi se obvykle vyskytne jeden nebo dva vývojáři, kteří celý proces řídí. Někdy je tato role formálně posvěcena jako oficiální *správce release* – což je něco úplně jiného, než je vlastník release (viz část **Diktatura vlastníka release** o něco výše v této kapitole), který rozhoduje o změnách. Správci release sledují, o kolika změnách se ještě rozhoduje, kolik jich bylo schváleno, kolik jich ještě pravděpodobně schváleno bude atd. Pokud budou mít pocit, že se nějaké důležité změně nevěnuje dost pozornosti a že o jejím nezařazení

do release rozhodne pouhý nezájem, mohou ostatním vývojářům jemně připomenout, aby ji nezapomněli revidovat a hlasovat o ní. Když jsou pak všechny změny schváleny, vezmou si tito vývojáři obvykle na sebe úlohu sloučit je s release větví. Není nic špatného na tom, když jim ostatní tento úkol přenechají, ale všichni by si přitom měli uvědomovat, že to není jejich povinnost, pokud se k tomu výslovně nezavázali. Jakmile přijde čas release vydat (viz část **Testování a releasing** dále v této kapitole), je také úkolem správců release zajistit všechny zbývající kroky – vytvoření finálních balíčků, sesbírání digitálních podpisů, uploadování balíčků a vydání veřejného oznámení.

Vytváření balíčků

Základní formou distribuce svobodného softwaru je zdrojový kód. To platí jak tehdy, když software obvykle běží ve formě zdrojového kódu (tedy je interpretovatelný – příkladem může být Perl, Python, PHP atd.), tak i pokud musí být nejprve zkompilován (jako C, C++, Java atd.). Software, který je potřeba kompilovat, si pravděpodobně nebudou uživatelé ze zdroje kompilovat sami, ale využijí místo toho už sestavené binární balíčky (viz část **Binární balíčky** dále v této kapitole). Nicméně i tyto balíčky vycházejí ze stejného zdroje, tedy distribuovaného kódu. Smyslem balíčku zdrojového kódu je release jednoznačně definovat. Pokud projekt distribuuje „Scanley 2.5.0“, znamená to konkrétně „Strom souborů zdrojového kódu, které po kompilaci (pokud je nutná) a instalaci vytvoří Scanley 2.5.0.“

Pro to, jak by měl release zdrojového kódu vypadat, platí poměrně přísné standardy. Občas se setkáte s tím, že tyto standardy někdo nedodrží, ale spíš výjimečně. Pokud tedy nemáte nějaký opravdu dobrý důvod je porušit, měli byste se jich držet i ve svém projektu.

Formát

Zdrojový kód by měl být dodán ve standardních formátech pro uložení adresářových stromů. V prostředí Unixu a operačních systémů podobných Unixu je zvykem použít formát TAR, komprimovaný pomocí compress, gzip, bzip nebo bzip2. V MS Windows je standardní metodou pro distribuci adresářových stromů formát *zip*, který zároveň provádí i kompresi, takže po vytvoření archivu už není potřeba jej dále komprimovat.

Soubory TAR

TAR je zkratka z „Tape ARchive“, tedy „páskový archiv“, protože jsou v tomto formátu adresářové stromy zaznamenány jako lineární tok dat, který je ideální pro uložení na pásek. Díky této vlastnosti se také stal standardem pro distribuci adresářových stromů v jediném souboru. Vytvoření komprimovaných tar souborů (také označovaných jako *tarballs*) je celkem jednoduché. V některých systémech lze komprimovaný archiv vytvořit už pomocí samotného příkazu tar; jinde je třeba použít pro kompresi zvláštní program.

Jméno balíčku a adresářový strom

Jméno balíčku by mělo obsahovat jméno softwaru plus číslo release plus přípony formátu odpovídající typu archivu. Například Scanley 2.5.0 zabalený pro Unix s využitím komprese GNU Zip (gzip) bude vypadat takto:

scanley-2.5.0.tar.gz

a pro Windows s použitím komprese zip:

scanley-2.5.0.zip

Oba tyto archivy po rozbalení vytvoří v aktuálním adresáři nový adresářový strom, který se bude jmenovat `scanley-2.5.0`. V něm se bude nacházet zdrojový kód ve formě, která je připravena ke kompilaci (pokud je potřeba) a instalaci. Na nejvyšší úrovni tohoto nového stromu by se měl nacházet soubor `README` ve formátu prostého textu, který popisuje, co software dělá a jaká je to verze, a obsahuje odkazy na další zdroje, jako jsou webové stránky projektu, další zajímavé soubory atd. Mezi těmito ostatními soubory by měl být i soubor `INSTALL`, blízký příbuzný souboru `README`, v němž se nacházejí instrukce, jak software sestavit a nainstalovat ve všech podporovaných operačních systémech. Jak bylo zmíněno v části **Jak licenci aplikovat** v kapitole **2. Zahájení projektu**, měl by zde existovat i soubor `COPYING` nebo `LICENSE`, v němž budou uvedeny právní podmínky pro distribuci softwaru.

Také by zde měl být soubor `CHANGES` (někdy se jmenuje `NEWS`), který popisuje, co je v tomto release nového. Soubor `CHANGES` obsahuje i seznamy změn všech minulých releases v obráceném chronologickém pořadí, takže seznam pro aktuální release se nachází na jeho začátku. Vytváření tohoto seznamu se obvykle v procesu stabilizace release dělá jako poslední. Některé projekty jej píšou průběžně už při vývoji, jiné si to raději schovávají až na konec a seznam pak píše jediný člověk s použitím logů ze systému správy verzí. Výsledný seznam pak může vypadat třeba takhle:

```
Verze 2.5.0
(20. prosince 2004, z /branches/2.5.x)
http://svn.scanley.org/repos/svn/tags/2.5.0/

  Nové funkce, vylepšení:
     * Přidány dotazy pomocí regulárních výrazů (issue #53)
     * Přidána podpora dokumentů ve formátech UTF-8 a UTF-16
     * Dokumentace přeložena do polštiny, ruštiny, malgaštiny
     * ...

  Opravené chyby:
     * opravena chyba s obnovou indexů (issue #945)
     * opraveno několik chyb v dotazování (issue #815, #1007, #1008)
     * ...
```

Tento seznam může být libovolně dlouhý, ale nemusíte do něj zanášet každou maličkou opravenou chybu nebo drobné zlepšení. Jeho smyslem je poskytnout uživatelům přehled toho, co aktualizací na tento release získají. Také je zvykem, aby byl tento seznam změn obsažen v e-mailu, jímž release oznamujete (viz část **Testování a releasing** dále v této kapitole), což byste měli vzít při psaní na vědomí.

CHANGES nebo ChangeLog

Tradičně se do souboru se jménem *ChangeLog* zapisují všechny změny, které kdy byly v projektu provedeny – tedy všechny revize, které byly zapsány do systému správy verzí. Pro soubory ChangeLog existují různé formáty, což zde ovšem není důležité, protože vždy obsahují totéž: datum změny, jejího autora a krátké shrnutí (nebo jenom záznam z logu).

Soubor `CHANGES` je něco jiného. I to je seznam změn, ale jen těch, které jsou podle vašeho uvážení důležité i pro veřejnost, a obvykle jsou v něm vynechána metadata, jako je datum a autor změny. Abyste předešli nedorozumění, nepoužívejte tyto dva termíny jako synonyma. Některé projekty místo „CHANGES“ používají „NEWS“. Tím sice vyloučí možnost, že by si soubor někdo spletl s ChangeLogem, ale je to poněkud nepřesné, protože v souboru CHANGES se udržují informace o změnách ve všech předchozích releasech, takže kromě skutečných novinek obsahuje i mnoho méně aktuálních informací.

Je ale možné, že soubory ChangeLog jako takové jsou na ústupu. Byly užitečné v dřívějších dobách, kdy se pro správu verzí používalo pouze CVS, protože data o změnách se z CVS exportovala jen obtížně. U novějších systémů pro správu verzí lze ale data, která ChangeLog obsahuje, vyžádat z úložiště správy verzí kdykoliv, takže je zbytečné udržovat je v nějakém statickém souboru – dokonce víc než zbytečné, protože takový ChangeLog pak pouze duplikuje zprávy z logů, které už v úložišti jsou.

Způsob, jakým je zdrojový kód v adresářovém stromu rozložen, by měl být stejný nebo přinejmenším co nejpodobnější tomu, jaký používá projekt přímo v úložišti správy verzí. Obvykle se zde vyskytne několik rozdílů, například proto, že balíček obsahuje ještě několik vygenerovaných souborů, které jsou nutné pro konfiguraci a kompilaci (viz část **Kompilace a instalace** dále v této kapitole), nebo proto, že zahrnuje i software třetích stran, jenž projekt nespravuje, ale který je k jeho provozu nutný a který uživatelé pravděpodobně nemají. Nicméně ani tehdy, když distribuovaný strom plně odpovídá vývojovému stromu v úložišti správy verzí, by distribuce samotná neměla být jen pracovní kopií (viz **pracovní kopie (working copy)** v části **Slovníček pojmů správy verzí** v kapitole **3. Technická infrastruktura**). Release má být statickým referenčním bodem – konkrétní, neměnnou konfigurací zdrojových souborů. Pokud by to byla pracovní kopie, hrozilo by nebezpečí, že ji uživatel aktualizuje a pak si bude myslet, že má pořád stejný release, i když už to ve skutečnosti bude něco jiného.

Pamatujte si, že balíček je vždy stejný, ať už je zabalený jakkoliv. Release – tedy ta přesná entita, na kterou někdo odkáže, když řekne „Scanley 2.5.0“ – je strom, který vytvoříte rozbalením souboru zip nebo tarballu. Projekt tedy může ke stažení nabízet všechny tyto možnosti:

scanley-2.5.0.tar.bz2
scanley-2.5.0.tar.gz
scanley-2.5.0.zip

…ale zdrojový strom, který rozbalením těchto souborů dostanete, musí být stejný. Právě tento strom je totiž distribucí jako takovou – to, v jaké formě byl stažen, je zcela nepodstatné. Jisté triviální změny mezi balíčky zdrojového kódu se ovšem povolují: například v balíčku pro Windows by měly být v textových souborech konce řádků vyznačené sekvencí CRLF (Carriage Return a Line Feed), zatímco v balíčcích pro Unix bude jen LF. I stromy samotné mohou v balíčcích určených pro různé operační systémy vypadat trochu jinak, pokud tyto operační systémy mají pro kompilaci nějaké zvláštní požadavky. Nicméně to jsou všechno jen detaily. Základní soubory se zdrojovým kódem by měly být ve všech balíčcích daného release stejné.

Velká nebo malá písmena

Když lidé uvádějí jméno nějakého projektu, používají obvykle velká písmena, jako by to bylo vlastní jméno, a velkými písmeny píšou i zkratky, pokud ve jméně jsou: „MySQL 5.0“, „Scanley 2.5.0“, atd. To, zda budou velká písmena zachována i ve jméně balíčku, záleží na projektu. Varianta `Scanley-2.5.0.tar.gz` i varianta `scanley-2.5.0.tar.gz` jsou zcela v pořádku (já osobně preferuji tu druhou, protože nechci uživatele nutit k použití klávesy shift, ale celá řada projektů u svých balíčků velká písmena píše). Důležité je, aby adresář, který vytvoříte, když tarball rozbalíte, používal velká písmena totožně. Neměla by tu být žádná překvapení – uživatel musí být schopen přesně předpovědět, jak se bude jmenovat adresář, který rozbalením vaší distribuce vznikne.

Pre-release

Když publikujete tzv. pre-release, tedy předběžný release, nebo kandidátní release, měla by být tato informace obsažena i ve jménu balíčku, protože je fakticky součástí čísla verze. Například při použití sekvence verzí alpha a beta, kterou jsme popsali výše v části **Z čeho se číslo verze skládá**, by jména balíčků vypadala takto:

scanley-2.3.0-alpha1.tar.gz
scanley-2.3.0-alpha2.tar.gz
scanley-2.3.0-beta1.tar.gz
scanley-2.3.0-beta2.tar.gz
scanley-2.3.0-beta3.tar.gz
scanley-2.3.0.tar.gz

První z nich se rozbalí do adresáře jménem `scanley-2.3.0-alpha1`, druhý do adresáře `scanley-2.3.0-alpha2` a tak dál.

Kompilace a instalace

U softwaru, který vyžaduje kompilaci nebo instalaci ze zdroje, existují obvykle určité standardní postupy, které by zkušení uživatelé měli být schopni následovat. Například u programů napsaných v C, C++ nebo jiných jazycích, které je potřeba kompilovat, je standardem u systémů podobných Unixu, že uživatel napíše:

```
$ ./configure
$ make
# make install
```

První příkaz automaticky zjistí tolik informací o prostředí, kolik dokáže, a připraví jej na proces sestavení. Druhý příkaz sestaví software na místě (ale neinstaluje jej) a poslední příkaz jej do systému nainstaluje. První dva příkazy spouštíte jako běžný uživatel, třetí jako root. Více informací o tom, jak tento systém nastavit, najdete ve výtečné knize Vaughana, Ellistona, Tromeye a Taylora jménem *GNU Autoconf, Automake, and Libtool*. V tištěné podobě ji vydává New Riders a její obsah je volně dostupný i online na **http://sources.redhat.com/autobook/.**

Není to jediný standard, který existuje, ale je to jeden z nejrozšířenějších. Poslední dobou je zejména u projektů psaných v Javě stále populárnější systém jménem Ant (**http://ant.apache.org/**), který má pro sestavování a instalaci vlastní standardizované postupy. Také některé programovací jazyky, například Perl a Python, doporučují, aby většina programů psaná v těchto jazycích používala stejný postup (například moduly Perl používají příkaz perl Makefile.PL). Pokud vám není jasné, jaké standardy se na váš projekt vztahují, zeptejte se nějakého zkušeného vývojáře. Můžete si být celkem jisti, že nějaké se na vás budou vztahovat určitě, i když nevíte, které to jsou.

Ať už jsou ale standardy, které se na váš projekt vztahují, jakékoliv, neporušujte je, pokud to nebude nezbytně nutné. Standardní instalační postupy provádí mnoho systémových administrátorů už zcela automaticky. Pokud uvidí v souboru `INSTALL` vašeho projektu povědomé instrukce, řeknou si, že dodržujete zavedené postupy, takže je i celkem pravděpodobné, že víte, co děláte. Jak už jsme také probrali v části **Stahování** v kapitole **2. Zahájení projektu**, standardizovaný postup pro sestavení potěší i potenciální vývojáře.

V prostředí Windows nejsou standardy pro sestavování a instalaci tak pevně zakořeněné. U projektů, které vyžadují kompilaci, bývá zvykem distribuovat strom, který lze zapojit do modelu pracovního prostředí / projektu, jejž používají standardní vývojářská prostředí Microsoftu (Developer Studio, Visual Studio, VS.NET, MSVC++ atd.). V závislosti na vašem softwaru může být možné nabídnout ve Windows možnost sestavování podobnou unixové použitím prostředí Cygwin (**http://www.cygwin.com/**).
A samozřejmě pokud používáte jazyk nebo programovací framework, který má vlastní zvyklosti pro sestavování a instalaci – např. Perl nebo Python, měli byste použít tu metodu, která je pro daný framework standardní, ať už pro Windows, Unix, Mac OS X nebo jiný operační systém.

Může se stát, že vás to, aby projekt vyhovoval zavedeným standardům pro sestavení a instalaci, bude stát hodně práce navíc, ale tato práce rozhodně stojí za to. Sestavení a instalace jsou vstupním bodem. Pokud je bezpodmínečně nutné, aby všechno, co následuje potom, bylo poněkud složitější, nevadí to tolik, jako kdyby vůbec první zkušenost nějakého uživatele nebo vývojáře s vaším softwarem vyžadovala neočekávané kroky.

Binární balíčky

Ačkoliv z formálního hlediska je releasem balíček zdrojového kódu, většina uživatelů použije k instalaci binární balíčky, které buď získají prostřednictvím mechanismu pro softwarovou distribuci, jenž je součástí jejich operačního systému, nebo ručně ze stránek projektu či nějaké třetí strany. „Binární“ zde neznamená nutně totéž, co „zkompilovaný“. Označuje se tak jakákoliv již nakonfigurovaná forma balíčku, který může uživatel na svůj počítač nainstalovat, aniž by následoval klasické postupy pro sestavování a instalaci. V systémech RedHat GNU/Linux je tímto systémem RPM, u Debian GNU/Linux je to systém APT (`.deb`) a v MS Windows obvykle buď soubor `.msi`, nebo samoinstalační soubory `.exe`.

Ať už byly tyto binární balíčky sestaveny někým, kdo s projektem úzce spolupracuje, nebo nějakou třetí stranou, budou s nimi uživatelé zacházet stejně jako s oficiálními releasy, a do bug trackeru projektu budou zapisovat nedostatky právě těchto balíčků. Je proto v zájmu projektu dodávat těm, kdo balíčky sestavují, podrobné instrukce a úzce s nimi spolupracovat, abyste měli jistotu, že to, co vytvoří, bude kvalitní a přesná reprezentace vašeho softwaru.

To nejdůležitější, co by tvůrci balíčků měli vědět, je, že by tyto binární balíčky měly být vždy založeny na oficiálním release zdrojového kódu. Někdy budou v pokušení získat z úložiště nějakou pozdější verzi nebo zahrnout změny, které byly zapsány až po vydání release, aby uživatelům poskytli novější opravy chyb nebo nějaká jiná zlepšení. Ten, kdo balíček vytváří, je přesvědčen, že tím uživatelům pomáhá, protože jim nabízí novější produkt, ale ve skutečnosti to může vyvolat velké zmatky. Projekty jsou vedené tak, aby přijímaly hlášení chyb, které se vyskytují ve vydaných verzích, a těch, které se nacházejí v nejnovějším zdrojovém kódu kmene a velkých větví (ty nacházejí uživatelé, kteří vědomě spouštějí ten vůbec nejaktuálnější kód). Když přijde z jednoho z těchto zdrojů bug report, je často možné potvrdit, že je to známá chyba, která se v tomto snímku opravdu vyskytuje, a možná i to, že od té doby již byla opravena a uživatel by měl tedy buď aktualizovat, nebo počkat na příští release. Pokud je to chyba, která dosud známá není, pak je možné ji díky informaci o přesném releasu relativně snadno zopakovat a v trackeru zařadit na správné místo.

Projekty ale nejsou připraveny na to, že jim přijdou bug reporty, které vycházejí z nějaké nespecifikovatelné hybridní meziverze. Takové chyby může být těžké reprodukovat; kromě toho se také může stát, že je způsobují neočekávané interakce několika izolovaných změn, které byly do této hybridní verze vtaženy z pozdějšího vývoje, což ale v žádném případě nelze brát jako chybu na straně vývojářů. Už jsem viděl situace, v nichž se promrhalo neskutečné množství času jenom proto, že nějaká chyba paradoxně chyběla tam, kde ji všichni očekávali – někdo spouštěl částečně záplatovanou verzi, která byla

založena na oficiálním release, ale nebyla s ním identická; když se pak ukázalo, že se tomuto uživateli neprojevuje jedna známá chyba, trvalo velmi dlouho, než někdo zjistil proč.

Někdy ale dojde k situaci, kdy autor balíčku bude trvat na tom, že jsou nějaké úpravy zdrojového kódu nutné. Bylo by dobré, kdyby v takovém případě celou věc probral s vývojáři projektu a své plány jim popsal. Možná to vývojáři schválí, ale i pokud ne, alespoň o věci budou vědět, takže se pak mohou mít na pozoru před neobvyklými bug reporty. Vývojáři mohou také zareagovat tím, že na webové stránky projektu připojí krátké varování a poprosí tvůrce balíčku, ať udělá na svých stránkách totéž, aby ti, kdo jeho balíček stahují, věděli, že jeho obsah není zcela identický s tím, co oficiálně vydal projekt. V takových situacích není důvod k žádným sporům, i když k nim bohužel často dochází. Rozdíl je hlavně v tom, že tvůrci balíčků mají trochu jiné cíle než vývojáři. To, co chtějí oni, je zajistit svým uživatelům co nejlepší funkčnost už při prvním spuštění. To samozřejmě chtějí i vývojáři, ale kromě toho také chtějí mít kontrolu nad tím, kolik verzí jejich softwaru existuje, aby mohli získávat od uživatelů smysluplné bug reporty a ručit za kompatibilitu. Někdy se ale tyto dva cíle střetnou. V takové situaci je dobré si uvědomit, že projekt nemá nad tvůrci balíčků žádnou moc a že tu fungují závazky na obě strany. Je pravda, že projekt dělá tvůrcům balíčků službu jednoduše tím, že produkuje software. Ale na druhou stranu tvůrci balíčků zase pomáhají projektu, neboť na sebe berou celkem nepopulární práci jenom proto, aby byl software dostupnější pro víc lidí – v některých případech až řádově víc. Můžete s nimi samozřejmě nesouhlasit, ale nerozhádejte si je. Zkuste se zkrátka dobrat k tomu nejlepšímu řešení.

Testování a releasing

Jakmile je ze stabilizované release větve vytvořen tarball zdrojového kódu, začíná veřejná část celého release procesu. Ale ještě předtím, než tento tarball dáte k dispozici celému světu, by měl být otestován a schválen nějakým minimálním počtem vývojářů – obvykle jsou to tři a víc. Toto schválení nespočívá jenom v tom, že se pokusí v tarballu najít zjevné chyby – ideálně by jej vývojáři měli stáhnout, sestavit a nainstalovat na čistý systém, spustit regresivní testy (viz **Automatizované testování** v kapitole **8. Řízení dobrovolníků**) a provést manuální testování. Pokud tyto kontroly proběhnou úspěšně a budou splněna i všechna ostatní kritéria, která projekt pro vydání nové verze má, je tarball svými vývojáři digitálně podepsán s použitím GnuPG (**http://www.gnupg.org/**), PGP (**http://www.pgpi.org/**) nebo nějakého jiného programu, který vytváří podpisy kompatibilní s PGP.

U většiny projektů vývojáři jednoduše použijí své osobní digitální podpisy místo sdíleného projektového klíče; podepsat se může tolik vývojářů, kolik bude chtít (to znamená, že existuje minimum, ale ne maximum). Více podpisů znamená, že tento release podstoupil více testování, ale také větší pravděpodobnost, že uživatel, který dbá na zabezpečení, dokáže najít důvěryhodnou digitální cestu mezi sebou a vaším tarballem.

Schválený release (tedy všechny tarbally, soubory zip a ostatní formáty, které distribuujete) by měl být umístěn ke stažení do příslušné sekce vašich stránek spolu s digitálními podpisy a kontrolními součty MD5/SHA1 (viz **http://en.wikipedia.org/wiki/Cryptographic_hash_function**). I zde existuje několik

různých standardů. Jedním způsobem je ke každému vydanému balíčku dodávat soubor s odpovídajícími digitálními podpisy a další soubor s kontrolním součtem. Například pokud je jedním z vydaných balíčků `scanley-2.5.0.tar.gz`, umístěte do stejného adresáře také soubor `scanley-2.5.0.tar.gz.asc`, který obsahuje digitální podpis tohoto tarballu, soubor `scanley-2.5.0.tar.gz.md5`, jenž obsahuje jeho kontrolní součet MD5, případně ještě další soubor, `scanley-2.5.0.tar.gz.sha1`, v němž je kontrolní součet SHA1. Kontrolu můžete zajistit také tak, že všechny podpisy pro všechny vydané balíčky shrnete do jediného souboru, `scanley-2.5.0.sigs`; totéž můžete udělat i s kontrolními součty.

Na tom, jak přesně to uděláte, moc nezáleží. Důležité je, aby toto schéma nebylo příliš složité, bylo dobře popsané a konzistentní napříč jednotlivými releasy. Smyslem všeho tohoto podpisování a poskytování kontrolních součtů je dát uživatelům možnost si ověřit, že kopie, kterou získali, nebyla po cestě upravena někým s nekalými úmysly. Uživatelé tento kód budou spouštět na svých počítačích – pokud s ním někdo něco prováděl, mohli by mu tak nechtěně poskytnout přístup ke všem svým datům. Více o paranoie najdete části **Bezpečnostní release** dále v této kapitole.

Release kandidát

U důležitých release, které obsahují mnoho změn, projekty nejprve vydávají takzvané *release kandidáty*, např. `scanley-2.5.0-beta1` před `scanley-2.5.0`. Smyslem kandidátního release je dát kód k dispozici k širšímu testování předtím, než bude označen za oficiální release. Pokud při tom budou nalezeny nějaké chyby, opraví se v release větvi a vydá se nový release kandidát (`scanley-2.5.0-beta2`). To celé se opakuje až do té doby, než se vychytají všechny nepřijatelné chyby; jakmile se to stane, poslední release kandidát je prohlášen za oficiální release – jediný rozdíl mezi posledním kandidátem a skutečným release je tedy to, že se odmaže kvalifikátor z čísla verze.

V ostatních ohledech byste měli s release kandidátem zacházet povětšinou stejně jako se skutečným releasem. Kvalifikátor *alpha, beta* nebo *rc* obvykle slouží jako dostatečné varování konzervativním uživatelům, že mají počkat na plnohodnotný release, a oznamovací e-maily kandidátních release by samozřejmě měly zdůraznit, že jeho smyslem je získat zpětnou vazbu. Jinak ale přistupujte k release kandidátům se stejnou péčí, jakou věnujete pravidelným releasům. Koneckonců je ve vašem zájmu, aby lidé kandidáty používali, protože to je nejlepší způsob, jak najít chyby, a také proto, že nikdy nevíte, který kandidát se nakonec stane oficiálním release.

Oznamování release

Oznamování vydání nové verze je stejné jako u jakékoliv jiné události a mělo by se řídit postupem popsaným v části **Publicita** v kapitole **6. Komunikace**. Vydávání nového release ale má i svá specifika.

Vždy, když někde uvádíte URL pro stažení tarballu, nezapomeňte připojit i kontrolní součty MD5/SHA1 a odkaz na soubor s digitálními podpisy. Vzhledem k tomu, že se oznámení objeví na několika různých fórech (mailing list, sekce novinek atd.), bude existovat i více zdrojů pro kontrolní součty, což uživate-

lům, kteří obzvlášť dbají na své zabezpečení, přináší další ujištění, že ani s těmito kontrolními součty nikdo nic zákeřného neprováděl. Pokud poskytnete odkaz na soubor s digitálními podpisy víckrát, na zabezpečení to žádný vliv mít nebude, ale alespoň to lidem (a to zejména těm, kteří projekt moc pečlivě nesledují) připomene, že bezpečnost berete vážně.

V e-mailu s oznámením a na stránkách, které obsahují podrobnější informace o novém release, by se měla objevit i relevantní pasáž souboru `CHANGES`, aby všichni věděli, proč by měli aktualizovat. To je zrovna tak důležité u kandidátních release jako u finálních – seznam opravených chyb a nových funkcí často uživatele přiláká k tomu kandidátní release vyzkoušet.

Také nezapomeňte poděkovat vývojářům, testerům a všem, kdo si našli čas poslat bug report. Neuvádějte ale konkrétní jména; výjimku můžete udělat tehdy, když je za obrovskou část práce na novém releasu zodpovědná jen jediná osoba a všichni v projektu význam této práce uznávají. Dejte si ale pozor, aby se vám tato poděkování nevymkla z ruky (viz části **Uvádění tvůrců a spoluautorů (Credit)** v kapitole **8. Řízení dobrovolníků**).

Udržování více release řad

Většina zavedených projektů udržuje najednou několik release řad. Například poté, co vyjde verze 1.0.0, by tato řada měla pokračovat mikroreleasy (opravujícími chyby) 1.0.1, 1.0.2 atd. do té doby, než se projekt rozhodne řadu ukončit. Ovšem to, že byla vydána verze 1.1.0, není dostatečný důvod k tomu řadu 1.0.x ukončit. Někteří uživatelé například zásadně neaktualizují na první verzi nové velké nebo malé série – počkají si tedy například, až ostatní uživatelé vychytají chyby z verze 1.1.0 a vyjde 1.1.1. Jejich jednání nemusí být nutně sobecké (nezapomínejte, že to také znamená, že využijí nové funkce a opravené chyby této verze až se zpožděním); mají zkrátka nějaký důvod k obezřetnosti při aktualizacích. Z toho tedy plyne, že pokud se projekt dozví o závažné chybě ve verzi 1.0.3 těsně před vydáním 1.1.0, bylo by poněkud drzé tuto chybu opravit jen ve verzi 1.1.0 a všem uživatelům řady 1.0.x říct, ať aktualizují. Zrovna tak můžete vydat najednou 1.1.0 a 1.0.4, čímž uspokojíte všechny.

Poté, co už bude řada 1.1.x dostatečně rozjetá, můžete vydat prohlášení, že řadě 1.0.x *končí životnost*. Toto oznámení by mělo být učiněno oficiálně. Může být buď zcela samostatné, nebo tvořit součást oznámení release 1.1.x. Ať už to uděláte jakkoliv, musíte svým uživatelům oznámit, že končí podpora staré řady, aby si mohli podle toho naplánovat aktualizace.

Některé projekty předem oznamují, v jakém časovém období budou dodávat podporu předchozí řadě. V kontextu open source prostředí znamená „poskytovat podporu“ to, že budou z této řady přijímat bug reporty a že pokud bude nalezena nějaká závažná chyba, vyjde nový release, který ji opraví. Ostatní projekty toto časové období předem neohlašují, ale sledují přicházející bug reporty, aby zjistili, kolik lidí ještě používá starou řadu. Jakmile jejich procento klesne pod určitou hranici, vyhlásí, že staré řadě končí životnost a s ní i podpora.

U každého release byste měli mít v bug trackeru příslušnou *cílovou verzi* nebo *cílový milník*, aby ti, kteří nahlašují chyby, mohli uvést, pro jaký release to je. Nezapomeňte také zanést cíl se jménem „vývoj“ nebo „poslední“ pro nejčerstvější kód, protože se najdou i jiní lidé než aktivní vývojáři, kteří budou raději používat aktuální vývojovou verzi než poslední release.

Bezpečnostní release

O tom, jak se vypořádat s bezpečnostními chybami, už jsme psali v části **Ohlášení bezpečnostních chyb** v kapitole **6. Komunikace**; zbývá ale ještě probrat pár věcí týkajících se bezpečnostních releasů.

Bezpečnostní release je takový, který byl vydán pouze proto, aby uzavřel nějakou bezpečnostní trhlinu. Kód, který takovou chybu opravuje, nemůže být zveřejněn, dokud není tento release k dispozici, což znamená nejen to, že opravy nelze do úložiště zapsat dříve než v den vydání, ale také to, že release nemůže být před vydáním veřejně testován. Vývojáři samozřejmě mohou opravu zkontrolovat sami a release otestovat soukromě, ale široké testování v reálném prostředí jednoduše není možné.

Právě z tohoto důvodu by měl bezpečnostní release obsahovat nějaký stávající release plus opravu bezpečnostní chyby a vůbec nic dalšího. To proto, že čím více neotestovaných změn vydáte, tím větší bude pravděpodobnost, že jedna z nich způsobí novou chybu, možná dokonce opět bezpečnostní. Tento konzervativní přístup je také vstřícný vůči administrátorům, kteří budou potřebovat bezpečnostní opravu nasadit, ale kteří by vzhledem ke svým zásadám pro aktualizaci raději zatím nic jiného neměnili.

Vytvoření bezpečnostního release s sebou někdy nese trochu záměrného matení uživatelů. Projekt například může mít rozpracovanou verzi 1.1.3, o níž už jste veřejně oznámili, že opraví některé konkrétní chyby verze 1.1.2, když ale náhle přijde hlášení o bezpečnostní chybě. Vývojáři se samozřejmě o této chybě nemohou zmiňovat, dokud ji neopraví; do té doby tedy budou o verzi 1.1.3 veřejně mluvit dál, jako by se vůbec nic nezměnilo. Když pak ale verze 1.1.3 skutečně vyjde, bude se od 1.1.2 lišit pouze tou bezpečnostní opravou; všechny ohlašované změny totiž budou odsunuty do verze 1.1.4 (která bude ale samozřejmě obsahovat i tu bezpečnostní opravu, jako ostatně i všechny další releasy).

Jiným řešením by bylo ke stávajícímu číslu verze přidat další složku, která bude znamenat, že jde pouze o bezpečnostní opravu. Uživatelé by pak například byli schopni z čísla 1.1.2.1 poznat, že jde o bezpečnostní release založený na verzi 1.1.2 a že všechny „vyšší“ verze (např. 1.1.3, 1.2.0 atd.) budou tuto bezpečnostní opravu obsahovat také. Těm, kteří tento systém znají, lze takto sdělit hodně informací. Na druhou stranu ale mohou být ti, kteří vývoj projektu zas tak pečlivě nesledují, celkem zmateni, když uvidí mezi čísly se třemi složkami, na něž jsou zvyklí, zdánlivě bezdůvodně zamíchaná čísla, která mají složky čtyři. Většina projektů, co jsem kdy viděl, raději zůstává konzistentní a pro bezpečnostní releasy používá další číslo v řadě, i pokud by to znamenalo, že se všechny ostatní plánované verze o jedno posunou.

Vydávání releasů a každodenní práce

To, že existuje několik release řad najednou, má své důsledky i pro to, jak probíhá každodenní práce vývojářů. Zejména to znamená, že pravidlo, které by jinak bylo pouze doporučené, se stává v podstatě nutností: každý commit musí obsahovat jedinou ucelenou změnu a nikdy se v jednom commitu nesmí smísit několik vzájemně nesouvisejících změn. Pokud se jedná o změnu, která je na jeden commit příliš velká nebo má příliš rozsáhlé důsledky, rozdělte ji do N dílčích commitů, které tuto změnu rozumně člení na menší části a neobsahují nic, co s ní nesouvisí.

Zde je příklad málo promyšleného commitu:

```
------------------------------------------------------------------------
r6228 | jnovak | 2004-06-30 22:13:07 -0500 (Wed, 30 Jun 2004) | 8 řádků

Oprava issue #1729: Přidáno varování pro uživatele, pokud se soubor
v průběhu indexace změní.

* ui/repl.py
  (ChangingFile): Nová třída výjimek.
  (DoIndex): Řešení nové výjimky.

* indexer/index.py
  (FollowStream): Vytvoření nové výjimky, pokud se soubor během indexace změní.
  (BuildDir): Nesouvisející změny: odstranění několika zastaralých komentářů,
  změna formátování kódu, oprava chybné kontroly při vytváření adresáře.

Další nesouvisející úpravy:

* www/index.html: Opraveno několik překlepů, nastaveno datum dalšího release.
------------------------------------------------------------------------
```

Problém nastane, jakmile někdo bude potřebovat přidat opravu chybné kontroly v `BuildDir` do větve připravovaného opravného release. Ostatní změny tohoto commitu ale zanášet nechce – řekněme, že oprava issue #1729 pro tuto větev nebyla schválena a úpravy `index.html` by tam byly zcela irelevantní. Jenže systém pro správu verzí nepovoluje sloučit s cílovou větví pouze tu opravu `BuildDir`, protože podle jeho záznamů tato změna logicky souvisí s těmi ostatními. Po pravdě by se ale celý problém stejně projevil už dřív. Už jenom hlasování o této změně by bylo komplikované – nestačilo by totiž uvést jen jméno revize, ale ten, kdo hlasování navrhuje, by musel vytvořit speciální záplatu nebo větev, na níž by izoloval pouze tu část příslušného commitu, o níž se má rozhodnout. To by znamenalo pro všechny zúčastněné spoustu práce navíc, a to jen proto, že se committer neobtěžoval své úpravy rozdělit do nějakých logických skupin.

Správně by měl tento commit sestávat ze čtyř samostatných commitů: jednoho, který opravuje problém `#1729`, dalšího, který odstraňuje zastaralé komentáře a upravuje formátování v `BuildDir`, třetího, který opravuje chybnou kontrolu v `BuildDir`, a konečně čtvrtého, který upravuje `index.html`. Do větve pro opravný release by pak byl začleněn ten třetí v pořadí.

Stabilizace releasů není samozřejmě jediným důvodem, proč je dobré mít v každém commitu jen jednu změnu. Z psychologického hlediska se takový ucelený commit snáze reviduje a je snadné jej i odstranit, pokud to bude nutné (v některých systémech pro správu verzí je ostatně odstranění commitu jen specifickou variantou sloučení). Pokud budou účastníci projektu od začátku disciplinovaní, mohou tím ušetřit všem ostatním spoustu trápení.

Plánování releasů

Jednou oblastí, v níž se open source projekty od proprietárních vždy výrazně lišily, je plánování releasů. Proprietární projekty obvykle mají data vydání nových verzí stanovená celkem pevně. Někdy je to proto, že zákazníkům bylo slíbeno, že k určitému datu bude dostupná aktualizace, někdy proto, že nový release musí být z marketingových důvodů koordinován ještě s něčím jiným, někdy zase proto, že investoři, kteří celou věc financují, chtějí předtím, než uvolní další peníze, vidět nějaké výsledky. Projekty svobodného softwaru byly ale na druhou stranu až donedávna poháněny převážně amatérismem, a to doslova: lidé je psali jen proto, že je to bavilo. Nikdo necítil potřebu vydat novou verzi dřív, než budou hotové všechny nové funkce – ostatně proč také? To, že by tito lidé jinak přišli o práci, jim rozhodně nehrozilo.

Dnes už existuje mnoho open source projektů, které jsou financovány z rozpočtu různých společností, takže na ně má čím dál větší vliv korporátní kultura, která chce vidět výsledky k nějakému danému datu. V mnoha ohledech je to pro ně přínosné, ale na druhou stranu to může způsobit konflikt mezi prioritami vývojářů, kteří jsou za svou práci placeni, a těch, kteří ji dělají dobrovolně. Tyto konflikty se často projevují právě tehdy, když přijde řeč na plánování nových verzí. Vývojáři, kteří dostávají plat, jsou pod jistým tlakem a budou tedy přirozeně chtít, aby se vybralo datum, kdy bude release vydán, a aby se tomuto datu všichni přizpůsobili. Dobrovolníci ale mohou chtít něco jiného – třeba dokončit funkce, které mají rozpracované, nebo provést nějaké testy, které mají v úmyslu; podle jejich názoru by měl release počkat, dokud tyto věci nedodělají.

Pro tento problém samozřejmě neexistuje nějaké obecně přijatelné řešení – pouze diskuse a kompromis. Rozsah a frekvenci třenic, které tento konflikt vyvolává, je ale možné alespoň omezit, pokud od sebe oddělíte dva různé aspekty stejné věci – samotnou existenci nějakého release a datum, kdy má být vydán. Tím myslím asi tohle: zkuste vést diskusi tak, aby se zaměřila na releasy, které projekt vydá v blízké a středně vzdálené budoucnosti, a na to, jaké funkce budou obsahovat, aniž byste zmiňovali konkrétní data – nanejvýš jen v podobě velmi hrubých odhadů, které se mohou značně posouvat.[23]

Tím, že se už na začátku shodnete na tom, jaké funkce by měl release obsahovat, snížíte složitost diskusí u každého jednotlivého release, čímž zvýšíte předvídatelnost celého procesu. Automaticky se tak vytvoří v projektu jistá setrvačnost a s ní i neochota vyhovět někomu, kdo bude chtít k určité verzi přidat nové funkce nebo její vydání nějak jinak komplikovat. Pokud je obsah releasu předem dobře definován, pak je na tomto člověku, aby svůj návrh nějak obhájil, třebaže datum být nastavené ještě nemusí.

Dumas Malone ve svém mnohosvazkovém životopisu Thomase Jeffersona, nazvaném *Jefferson and His Time* (Jefferson a jeho doba) popisuje, jak probíhalo jednání, na němž se rozhodovalo o organizaci budoucí University of Virginia. Založit univerzitu byl původně Jeffersonův nápad, ale jak se často stává (a to nejen v open source projektech), členy rady se rychle stalo i mnoho dalších lidí, kteří sledovali vlastní cíle. Na první setkání, kde se měly probrat podrobnosti, přišel Jefferson už s hotovými, do posledního detailu rozkreslenými architektonickými návrhy, podrobně propracovaným rozpočtem stavby a provozu, hotovými sylaby a jmény pedagogů, které chtěl přizvat z Evropy. Nikdo jiný nebyl ani zdaleka tak dobře připraven, takže jediné, co mohli dělat, bylo před Jeffersonovým návrhem kapitulovat. Univerzita byla nakonec založena na základě těchto plánů. To, že stavba původní rozpočet značně překročila a že se celá řada nápadů z mnoha různých důvodů nakonec nerealizovala, bylo něco, s čím Jefferson pravděpodobně počítal už od samého začátku. Z jeho strany šlo o strategický tah: na jednání přišel s něčím velmi konkrétním, takže všem ostatním nezbývalo nic jiného, než pouze navrhovat dílčí úpravy; celkový tvar projektu a s ním i jeho harmonogram zůstaly v podstatě tak, jak chtěl.

V projektech svobodného softwaru se takováto „setkání“ nepořádají; místo toho se zde objevuje řada menších návrhů, nejčastěji v issue trackeru. Pokud ale jako člověk, který má v projektu jistou důvěru, začnete přiřazovat k budoucím releasům v issue trackeru různé funkce, zlepšení a opravy chyb na základě nějakého předem ohlášeného plánu, většinou se ostatní přidají. Jakmile dostanete věci víceméně tam, kam chcete, budou všechny diskuse o datech vydání probíhat mnohem snáz.

Samozřejmě je naprosto zásadní, abyste jednotlivá rozhodnutí nepředstavovali, jako by to byla jejich definitivní podoba. V komentářích ke každému dílčímu přiřazení nějaké věci ke konkrétnímu budoucímu release byste měli vítat diskusi nebo nesouhlas a být ochotni se nechat přesvědčit, že by se to mělo dělat jinak. Nikdy se nesnažte uplatnit svůj vliv na projekt jenom proto, že můžete. Čím více se budou ostatní podílet na procesu plánování release (viz část **Podělte se o řídící i technické úkoly** v kapitole **8. Řízení dobrovolníků**), tím snazší bude je přesvědčit, aby se u těch otázek, na nichž vám záleží, přiklonili na vaši stranu.

---

[23] Jiný přístup k věci najdete v disertaci Martina Michlmayra, která se jmenuje *Quality Improvement in Volunteer Free and Open Source Software Projects: Exploring the Impact of Release Management* (Zlepšování kvality v dobrovolnických projektech svobodného a open source software: zkoumání dopadu správy vydávání nových verzí) – **http://www.cyrius.com/publications/michlmayr-phd.html**. Zkoumá v ní vliv, které mají release procesy založené na času (místo na funkcích) na velké projekty svobodného softwaru. Michlmayr měl na stejné téma i přednášku pro Google; ta je dostupná na Google Video na **http://video.google.com/videoplay?docid=-5503858974016723264**.

Dalším způsobem, jak snížit napětí v projektu, které vyplývá z plánování releasů, je vydávat nové verze relativně často. Když mezi jednotlivými releasy pokaždé uběhne dlouhá doba, jejich význam v myslích všech zúčastněných roste. O to víc je zdrcující, když se pak něčí změna do nové verze nedostane, protože dotyčný dobře ví, jak dlouho bude trvat, než dostane další příležitost. V závislosti na složitosti procesu vytváření nové verze a na charakteru celého vašeho projektu se bude ideální prodleva mezi releasy pohybovat někde mezi třemi až šesti měsíci; opravné mikroreleasy, pokud je o ně zájem, by měly vycházet o něco častěji.

7. Vytváření balíčků, vydávání releasů a každodenní práce na vývoji

8. Řízení dobrovolníků

8. Řízení dobrovolníků — 221

Jak dostat z dobrovolníků to nejlepší — 222
Delegování — **222**
Jasně rozlišujte mezi poptáváním a zadáváním — **223**
Postup po delegování — **224**
Všímejte si, o co se lidé zajímají — **225**
Pochvala a kritika — **225**
Zabraňte teritorialitě — **226**
Poměr automatizace — **229**
Automatizované testování — **229**
Ke všem uživatelům se chovejte jako k potenciálním dobrovolníkům — **231**

Podělte se o řídící i technické úkoly — 234
Patch Manager (manažer záplat) — **235**
Translation Manager (manažer překladu) — **236**
Documentation Manager (manažer dokumentace) — **238**
Issue manager (manažer problémů) — **239**
FAQ Manager (manažer sekce s nejčastěji kladenými otázkami) — **240**

Personální změny — 241

Committeři — 244
Volba committerů — **244**
Odebrání commit access — **245**
Částečný commit access — **246**
Nečinní committeři — **247**
Nedělejte s ničím tajnosti — **247**

Uvádění tvůrců a spoluautorů (Credit) — 248

Odnože (Forks) — 249
Zvládání odnoží — **250**
Iniciování odnože — **252**

8. Řízení dobrovolníků

Přimět lidi, aby se dohodli na tom, co projekt potřebuje, a společně pak pracovat na dosažení těchto cílů, to vyžaduje více než jen družnou atmosféru, prostou patrných dysfunkcí. Chce to člověka, případně tým lidí, který bude uvědoměle řídit všechny, kdo se na projektu podílejí. Řízení dobrovolníků nemusí být technické řemeslo ve smyslu počítačového programování, přesto to řemeslo je, lze se v něm totiž zdokonalovat učením a praxí.

Tato kapitola obsahuje určitou všehochuť různých specifických technik určených k řízení dobrovolníků. Vychází možná o něco více než předchozí kapitoly z projektu Subversion jako z případové studie, zčásti proto, že jsem při psaní této publikace na tomto projektu pracoval, a měl jsem tak všechny primární zdroje hned po ruce, a zčásti proto, že je obecně přijatelnější, když někdo hází kameny na vlastní skleník než na skleníky ostatních. Užitek, který s sebou přinášelo aplikování—níže uvedených doporučení a naopak následky pramenící z jejich nepoužívání—jsem však viděl i v jiných projektech; bude-li to „politicky“ schůdné, pokusím se zde podat i některé příklady z jiných projektů.

Když už mluvím o politice, myslím, že bychom se klidně mohli na ono tolik znevažované slovo podívat trochu podrobněji. Mnoho techniků si myslí, že politika je něco, v čem se ostatní lidé angažují. „*Já* se jen snažím obhajovat nejlepší cestu, kterou by se měl projekt ubírat, a *ona* tady vznáší čistě politické námitky.“ Myslím, že odpor k politice (nebo přesněji k určité představě politiky), který je zvláště silný mezi technickými pracovníky, je způsoben tím, že technici jednoduše věří tomu, že některá řešení jsou objektivně lepší než jiná. Proto, jakmile někdo jedná způsobem, který odhaluje vnější motivaci—, například udržení vlastní vlivné pozice, zmírnění cizího vlivu, otevřené handlování nebo nechuť ranit něčí city—, ostatní účastníky projektu to může začít obtěžovat. Samozřejmě, že toto východisko jen zřídka někomu zabrání v tom, aby se choval podobně, jakmile začnou být v sázce jeho vlastní životně důležité zájmy.

Pokud považujete slovo „politika“ za vulgarismus a doufáte, že před ní svůj projekt uchráníte, rovnou to vzdejte. Politika je nezbytná ve chvíli, kdy lidé musí společnými silami spravovat sdílené zdroje. Je zcela logické, že jedním z motivů, který zasahuje do každého lidského rozhodování, je otázka, jak může daný krok ovlivnit náš budoucí vliv v rámci projektu. Nakonec, pokud – stejně jako většina programátorů – věříte vlastnímu úsudku a dovednostem, případná ztráta budoucího vlivu by měla být v určitém smyslu chápána jako technický výsledek. Podobná logika platí i pro ostatní chování, které se může na první pohled jevit jako „čistá“ politika. Ve skutečnosti nic takového jako čistá politika neexistuje: je tomu tak proto a jen proto, že různé kroky a jednání mají své následky v reálném světě, následky, kterých si jsou lidé především politicky vědomi. Politika je koneckonců jen uznání faktu, že je nutno brát do úvahy *všechny* důsledky rozhodnutí. Pokud určité rozhodnutí vede k výsledku, který většina účastníků považuje za technicky uspokojivý, ale zahrnuje v sobě také změnu v mocenských vztazích, která způsobí, že se klíčoví členové budou cítit izolováni, jsou oba tyto efekty stejně důležité. Ignorovat tuto skutečnost je projevem krátkozrakosti, nikoli velkomyslnosti.

Při čtení následující rady a při práci s projektem mějte tedy na paměti, že *nikdo* nestojí nad politikou. Budit zdání, že stojíte nad politikou, je pouze určitá politická strategie, která sice bývá velice užitečná, ale nikdy ji nelze pokládat za realitu. Politika je zkrátka stav, k němuž dochází, když se lidé neshodnou, a úspěšné projekty jsou ty, které rozvíjejí politické mechanismy ke konstruktivnímu zvládání neshod.

Jak dostat z dobrovolníků to nejlepší

Proč dobrovolníci pracují na projektech svobodného softwaru?[24]

V odpovědích na tuto otázku většina z nich uvádí, že chtějí vytvořit dobrý software nebo se chtějí osobně zapojit do oprav chyb, které se jich dotýkají. Tyto důvody však obvykle odkrývají jen část pravdy. Koneckonců, dokázali byste si představit dobrovolníka, který by setrvával u určitého projektu, aniž by mu kdo slůvkem vyjádřil ocenění za jeho práci nebo mu naslouchal při diskusích? Samozřejmě, že ne. Lidé nepochybně tráví svůj čas se svobodným softwarem z důvodů, které se skrývají za pouhou abstraktní touhou po vytvoření kvalitního kódu. Pochopení skutečné motivace dobrovolníků vám napomůže uspořádat věci tak, abyste je ke svému projektu nejen přilákali, ale také je u něj udrželi. Touha vytvořit kvalitní software může být součástí této motivace, společně s chutí podílet se na plnění náročného úkolu (výzva) a vzdělávací hodnotou, kterou tato činnost v sobě nese. Lidé však v sobě také mají zakódovánu touhu po práci s ostatními a touhu projevovat a zasluhovat si úctu prostřednictvím skupinových aktivit. Skupiny zapojené do těchto součinných aktivit musí rozvíjet normy chování tak, aby byl dosažen a v rámci všech kroků také zachováván status, který pomůže cílům skupiny.

Tyto normy se vždy nevytvoří samy od sebe. Například u některých projektů zkušení open source vývojáři jsou pravděpodobně schopni několik takových projektů jmenovat jen tak z hlavy – lidé zjevně cítí, že status se získává umísťováním častých a květnatých komentářů/vzkazů. K tomuto závěru nepřišli náhodně; dospěli k němu díky tomu, že jim byl vždy projevován jistý respekt, když vznášeli dlouhé a komplikované argumenty, ať již byly v projektu k užitku či nikoli. Dále uvádím některé techniky pro vytváření atmosféry, v níž jsou kroky k získání statutu také konstruktivními činy.

Delegování

Delegování není jen způsob, jak vhodně rozložit pracovní zátěž; je to také politický a sociální nástroj. Než někoho o něco požádáte, zvažte všechny důsledky. Nejpatrnější důsledek je, pokud vaši prosbu přijme, že úkol splní místo vás. Dalším důsledkem však je, že si dotyčný uvědomí, že jste mu důvěřovali a svěřili daný úkol. Dále, pokud jste svou žádost učinili na veřejném fóru, dotyčný ví, že i ostatní

---

[24] Této otázce se věnovala podrobná studie se zajímavými výsledky, přednáška Karim Lakhani a Roberta G. Wolfa, nazvaná *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects* (Proč hackeři dělají to, co dělají: jak chápat motivaci a snahy v projektech free/open source softwaru). Viz **http://freesoftware.mit.edu/papers/lakhaniwolf.pdf**.

vědí o této důvěře. Může se rovněž cítit pod jistým tlakem, aby úkol přijal, což znamená, že jej musíte žádat způsobem, který mu umožní s díky vaši prosbu odmítnout, pokud opravdu nechce danou práci provést. Pokud si úkol žádá koordinaci s ostatními členy projektu, navrhujete mu, že se do projektu zapojí hlouběji, vytvoří si vazby, které by si za jiných okolností nevytvořil, a možná se stane autoritou v některé subdoméně celého projektu. Hlubší zapojení do projektu může osloveného dobrovolníka buď zastrašit, nebo jej může vést k angažování i jinými způsoby, zvýšeným pocitem celkového závazku.

Kvůli všem těmto účinkům má v mnoha případech smysl požádat někoho jiného, aby něco udělal, ačkoliv dobře víte, že byste daný úkol sami udělali rychleji či lépe. Samozřejmě, že v této souvislosti může také někdy převládnout argument ekonomické efektivnosti: hodnota ušlého zisku by pro vás mohla být příliš vysoká—v ušetřeném čase byste mohli dělat něco ještě důležitějšího. Ale i kdyby se tohoto případu argument ušlého zisku netýkal, *můžete* chtít požádat někoho jiného, aby daný úkol provedl, v dlouhodobé perspektivě jej totiž například chcete vtáhnout hlouběji do projektu, i kdybyste zprvu měli vynaložit dodatečný čas na dohled nad danou osobu. Platí také obrácená technika: pokud dobrovolně provedete práci, kterou někdo jiný nechtěl nebo neměl čas udělat, získáte si jeho dobrou vůli a úctu. Delegování a nahrazování neslouží jen k provedení jednotlivých úkolů; jejich prostřednictvím lze vtáhnout lidi hlouběji do celého projektu.

Jasně rozlišujte mezi poptáváním a zadáváním

Někdy je správné předpokládat, že určitá osoba určitý úkol přijme. Například pokud někdo zapíše do kódu chybu nebo potvrdí kód, který evidentním způsobem nedodržuje směrnice projektu, stačí na problém upozornit a poté se chovat tak, jako byste předpokládali, že se daná osoba o řešení problému postará. Mohou však nastat i jiné situace, kdy není vůbec jasné, že byste mohli nějaké kroky od této osoby očekávat. Osoba může udělat, oč ji požádáte, nebo taky ne. Jelikož nikdo nemá rád, když od něj ostatní něco se samozřejmostí očekávají, měli byste být citliví na rozdíl mezi dvěma výše uvedenými typovými situacemi a uzpůsobit jim své požadavky.

Jedna z věcí, která lidi okamžitě dopálí, je, když jsou požádáni, aby něco udělali, stylem, který naznačuje, že nepochybujete, že je to stejně jejich odpovědnost, ačkoliv oni na to mají úplně jiný názor. Například zadávání došlých problémů (issues) je zvláště úrodnou půdou pro tento druh obtěžování. Účastníci projektu obvykle vědí, kdo je odborník na určité oblasti, čili ve chvíli, kdy přijde bug report, často již všichni vědí o jednom či dvou lidech, kteří budou pravděpodobně schopni danou chybu rychle opravit. Nicméně pokud práci na tomto problému zadáte někomu z těchto dvou lidí bez jejich předchozího svolení, může se dotyčný cítit v dosti bezvýchodné situaci, do níž se navíc ani nedostal vlastním přičiněním. Bude cítit tlak očekávání, ale v důsledku to také bude vnímat jako trest za své odborné dovednosti. Koneckonců, člověk získává odborné dovednosti tím, že opravuje chyby, tak proč by se tohoto konkrétního úkolu nemohl ujmout někdo jiný! (Povšimněte si, že issue trackery, které automaticky zadávají problémy (issues) konkrétním lidem na základě informací obsažených v bug reportu, nejsou tak urážlivé, protože všichni dobře vědí, že toto zadávání úkolů provádí automatizovaný proces, a nejedná se tak o náznaky lidského očekávání.)

Ačkoliv by bylo krásné rozdělovat pracovní zátěž co možná nejrovnoměrněji, někdy byste rádi jen podnítili osobu, která dokáže opravit danou chybu nejrychleji, aby tak učinila. Vzhledem k tomu, že si nemůžete u každého takového zadávání dovolit komunikační řetězec („Mohl byste se prosím podívat na tuto chybu?“ „Ano.“ „Fajn, tento úkol tedy zadávám vám.“ „OK“), měli byste celé zadávání úkolu provést formou poptávky, tím byste nevyvíjeli na účastníky žádný tlak. V podstatě všechny issue trackery umožňují přiřazovat k zadání práce na určitém problému (issue) také komentáře. Do tohoto komentáře můžete zapsat například:

> Řešení tohoto problému zadávám Vám, pane Nováku, protože jste s tímto kódem nejvíce obeznámen. Můžete jej klidně odmítnout, pokud na to nemáte dostatek času. (Byl byste radši, kdybychom Vám podobné žádosti v budoucnu nezasílali?)

Tím jasně rozlišíte mezi *žádostí* o zadání úkolu a příjemcovým *přijetím* tohoto zadání. Tuto komunikaci nesleduje pouze případná osoba, jíž bude úkol zadán: veřejné potvrzení této osoby vidí celá skupina, ale vaše zpráva také dává této osobě jasně na vybranou, zda chce daný úkol přijmout či nikoli.

Postup po delegování

Když požádáte někoho, aby něco udělal, zapamatujte si to a snažte se být s danou osobou v kontaktu. Většina žádostí se podává na veřejných fórech a vypadají zhruba takto: „Mohl/a byste se postarat o X? Každopádně se nám ozvěte; nevadí, pokud nemůžete, jen nám dejte vědět.“ Na takovouto výzvu můžete, ale nemusíte dostat odpověď. Pokud odpověď dostanete a je záporná, smyčka se uzavře—a pro řešení X musíte vyzkoušet jinou strategii. Pokud je odpověď kladná, pečlivě sledujte postup prací na daném problému a komentujte pokroky, které vidíte či nevidíte (každému se pracuje lépe, když ví, že někdo jiný jeho práci oceňuje). Pokud po několika dnech nedostanete žádnou odpověď, zeptejte se ještě jednou nebo vyvěste vzkaz, že jste dosud neobdržel žádnou odpověď a hledáte někoho, kdo by problém vyřešil. Nebo problém vyřešte sami, ale určitě oznamte, že jste nedostali žádnou odpověď na původní výzvu.

Účelem tohoto oznámení *není* někoho ponížit a vaše komentáře by měly být formulovány tak, aby nevyvolaly podobný dojem. Účelem je pouze ukázat, že si pečlivě zaznamenáváte všechny své žádosti a veškeré reakce, které na ně obdržíte. Tím zvýšíte pravděpodobnost, že se na další výzvu někdo ozve, protože dobře uvidí (byť nevědomky), že se snažíte všímat si veškeré práce, kterou dělají, když už jste si všimli tak nepatrné události, jako že nikdo nereagoval na výzvu.

Všímejte si, o co se lidé zajímají

Další věc, která udělá lidem radost, je, když si někdo všimne jejich zájmů—obecně platí, že čím více osobních rysů si u někoho všimnete a zapamatujete, tím snadněji se vám s ním bude komunikovat a tím více bude chtít pracovat se skupinami, na nichž se podílíte.
Například účastníci projektu Subversion se jasně rozdělili na skupinu, která chtěla dosáhnout definitivního vydání verze 1.0 (což jsme nakonec udělali), a skupinu lidí, kteří chtěli především přidávat nové funkční vlastnosti a pracovat na zajímavých problémech, ale o to, kdy verze 1.0 vyjde, se příliš nezajímali. Žádná z těchto názorových skupin není lepší či horší než druhá; jsou to jen dva různé druhy vývojářů a oba tábory odvádějí na projektu velký kus práce. Ale díky tomu jsme se rychle dozvěděli, že je důležité *nevycházet* z předpokladu, že všichni účastníci projektu jsou nutně hnáni touhou po vydání verze 1.0. Elektronická média mohou klamat: může se vám zdát, že cítíte obecnou chuť jít za společným cílem, ve skutečnosti tuto chuť s vámi sdílejí pouze lidé, s nimiž jste náhodou mluvili, zatímco ostatní mají úplně odlišné priority.

Čím více toho víte o tom, co lidé od projektu očekávají, tím účinněji od nich můžete něco žádat. I pouhý projev toho, že víte, co chtějí, aniž byste hned museli od nich něco požadovat, je velice užitečný, potvrzuje totiž každému účastníkovi, že není pouhým zrnkem písku na poušti.

Pochvala a kritika

Pochvala a kritika nejsou protiklady; v mnoha ohledech jsou si podobné. Především se jedná o způsoby projevu pozornosti a nejúčinnější jsou, pokud jsou specifické, nikoli jen obecné. Obě by se měly používat s konkrétním záměrem. Obě podléhají inflačnímu efektu: pokud chválíte příliš nebo často, svou pochvalu tak devalvujete; stejná poučka platí i pro kritiku, ačkoliv v praxi je kritika obvykle reaktivní, a tím pádem o něco odolnější vůči devalvaci.

Důležitým rysem technické kultury je skutečnost, že detailní, věcná kritika je často považována za pochvalu (viz diskuze v části **Jak rozeznat hrubost** v kapitole **6. Komunikace**), neboť napovídá, že práce dané osoby si zaslouží, aby byl její analýze věnován určitý čas. Musí však být splněny obě podmínky—musí se jednat o detailní a věcnou kritiku. Například, když někdo provede nedbalou změnu v kódu, je zbytečné (a v podstatě škodlivé) reagovat na ni prostou poznámkou „To byla nedbalost.“ Nedbalost je totiž především osobní charakteristikou, nikoli vlastností práce a vaše reakce by měly být zaměřeny výhradně na práci. Je mnohem účinnější popsat taktním a neškodolibým způsobem všechny chyby v dané změně. Pokud se jedná o třetí či čtvrtou nedbalou změnu v řadě, provedenou toutéž osobou, je vhodné tuto skutečnost—opět bez emocí—zmínit na konci vaší kritiky, aby bylo jasné, že si tohoto jevu někdo všiml.

Pokud se někdo v reakci na kritiku nenapraví, silnější či častější kritizování nebude správným řešením. Řešením je vyloučení dané osoby ze skupiny z titulu její nekompetentnosti, a to způsobem, který pokud možno minimalizuje raněné city; příklady viz část **Personální změny** dále v této kapitole.

K této situaci však dochází zřídka. Většina lidí reaguje vstřícně na kritiku, která je specifická, podrobná a obsahuje jasné (byť nevyslovené) předpoklady zlepšení.

Chvála samozřejmě ničí city neraní, to však neznamená, že s ní nemusíte nakládat tak uvážlivě jako s kritikou. Chvála je nástroj: než jej použijete, sami si odpovězte na otázku, proč jej chcete použít. V principu platí, že není radno chválit lidi za něco, co dělají zcela běžně nebo za činnosti, které jsou normální nebo které lze od nich v rámci jejich účasti ve skupině očekávat. Pokud s něčím takovým jednou začnete, je velice těžké vědět, kdy přestat: měli byste chválit *všechny* za to, že dělají běžné věci? Nakonec se stane, že když někoho vynecháte, lidi se začnou ptát proč. Je mnohem lepší vyjadřovat chválu a vděk pouze zřídka, jako reakci na nečekané či nezvyklé úsilí s cílem podnítit více takových snah. Pokud se určitý účastník projektu permanentně posouvá k vyšší a vyšší výkonnosti, upravte pro něj náležitě i svou „mez chvály". Opakovaná chvála za zcela běžné chování postupně ztrácí na významu. Namísto toho začne dotyčná osoba cítit, že vysoký stupeň její produktivity je již nyní považován za normální a přirozený a všímáte už si pouze práce, která jde za tento stupeň.

Tím samozřejmě nechci říci, že byste neměli náležitě uznávat přínos dané osoby. Ale pamatujte, že pokud je projekt správně nastaven, vše, co dotyčná osoba provede, je tak či tak viditelné, a skupina tudíž bude o veškeré činnosti oné osoby dobře vědět (a ona osoba bude vědět, že o tom skupina ví). Práci lze ocenit i jiným způsobem než přímou pochvalou. V diskusi o souvisejícím tématu můžete mimochodem zmínit, že daná osoba vykonala v určité oblasti mnoho práce a je v ní skutečným odborníkem; můžete s touto osobou veřejně konzultovat otázky týkající se kódu; a možná bude nejúčinnější, když budete viditelně využívat práci, kterou tato osoba udělala - pak uvidí, že ostatní se klidně spoléhají na výsledky její práce. Pravděpodobně ani nebude nutné provádět tyto kroky nějak násilně. Kdo pravidelně velkou měrou přispívá do projektu, to bude vědět a bude zastávat vlivnou pozici automaticky. Zpravidla není zapotřebí činit explicitní kroky, pokud ovšem nemáte dojem, že je přispěvatel z jakéhokoli důvodu nedostatečně oceněn.

Zabraňte teritorialitě

Dávejte si pozor na účastníky, kteří se snaží vyhradit si exkluzivní práva na určité oblasti projektu a chtěli by snad dokonce provádět veškerou práci v těchto oblastech, a to až do té míry, že agresivním způsobem přebírají práci, kterou začali ostatní. Takovéto chování se dokonce může zprvu jevit jako prospěšné. Zdánlivě to totiž vypadá, že si daná osoba bere na sebe větší odpovědnost a v dané oblasti projevuje zvýšenou aktivitu. Z dlouhodobé perspektivy je však toto chování destruktivní. Jakmile lidé někde vytuší značku „Zákaz vstupu", budou se držet zpátky. Tím v dotyčné oblasti výrazně poklesne kontrola a stoupne křehkost, protože jediným bodem zlomu bude samotný vývojář. A co je ještě horší, podrývá se tím kooperativní rovnostářský duch celého projektu. Teoreticky by měl být vítán každý vývojář, který chce kdykoli vypomoci s jakýmkoli úkolem. V praxi se samozřejmě věci vyvíjejí trochu odlišně: lidé mají oblasti, v nichž jsou více a méně vlivní, a neodborníci se často podrobují odborníkům v určitých oblastech projektu. Ovšem důležité je, že celý tento stav je dobrovolný: na základě

profesní způsobilosti (kompetence) a zralého úsudku je někomu udělena neformální moc, ta by však nikdy neměla být aktivně *převzata*. I v případě, že osoba toužící po moci je kompetentní, je nadále klíčové, aby tuto moc držela neformálně, tj. na základě konsensu v celé skupině, a aby tato moc nikdy nevedla k tomu, že budou z práce v dané oblasti vyloučení ostatní.

Odmítnutí nebo úprava něčí práce z technických důvodů je samozřejmě něco úplně jiného. Zde je totiž rozhodujícím faktorem vlastní obsah práce, nikoli to, kdo náhodou zastává funkci hlídače. Možná, že stejná osoba náhodou vykonává v dané oblasti nejvíce kontrol. Pokud se však nesnaží bránit ostatním v téže práci, vše je pravděpodobně v pořádku.

Z důvodu boje proti zárodkům teritoriality nebo jejímu výskytu se mnoho projektů rozhodlo přijmout kroky zakazující uvádění jmen autorů nebo jmen určených správcům ve zdrojovém souboru. S touto praxí hluboce souhlasím: dodržujeme ji i v projektu Subversion a stala se víceméně oficiální politikou Apache Software Foundation. Sander Striker, člen ASF, k tomu uvádí:

> *V nadaci Apache Software se snažíme zabránit užívání značek autorů (tagů) ve zdrojovém kódu. Kromě případných právních důsledků nás k tomu vede několik různých důvodů. Společný (kolaborativní) vývoj spočívá ve skupinové práci a péči o projekt. Uvádění jmen autorů je správné a mělo by se praktikovat, ale takovým způsobem, aby se zamezilo nesprávnému připisování zásluh, byť jen pouhým naznačením. Neexistuje jasná hranice pro to, kdy přidat a kdy odebrat autorský tag. Uvádíte své jméno, když pozměníte třeba jen komentář? Nebo když přidáte jednořádkovou opravu? Odstraňujete ostatní autorské tagy, když přefaktorujete kód a on pak vypadá z 95% jinak? Co uděláte s těmi, kteří si „hmátnou“ na každý soubor, přičemž v něm provedou jen takové změny, aby naplnily kvótu pro autorský tag a jejich jméno pak bylo všude uvedeno?*
>
> *Existují lepší způsoby, jak uvádět jména autorů, a preferujeme jejich využívání. Z technického hlediska nejsou autorské tagy nezbytné; pokud chcete zjistit, kdo zapsal určitou část kódu, můžete se podívat do systému správy verzí. Autorské tagy také zastarávají. Skutečně byste byli rádi, kdyby vás někdo soukromě kontaktoval kvůli části kódu, kterou jste zapsali před pěti lety a pak velice ochotně zapomněli?*

Soubory se zdrojovým kódem softwarového projektu jsou jádrem jeho identity. Měly by odrážet skutečnost, že komunita vývojářů je za ně odpovědná jako celek, že by neměly být rozdělovány na malá „léna“.

Někdy můžete zaslechnout argumenty na podporu autorských nebo správcovských tagů ve zdrojových souborech vycházející z toho, že tyto tagy jasně uvádějí jména nejpracovitějších účastníků projektu. Tento argument však s sebou přináší dva problémy. Zaprvé, tagy vznášejí nepříjemnou otázku, kolik práce musí dotyčný udělat, aby bylo jeho jméno uvedeno na seznamu autorů. Zadruhé, tagy v sobě spojují problém zásluh a moci: z toho, že jste v minulosti odvedli určitou práci, neplyne, že vám připadlo vlastnictví oblasti, kde jste tuto práci odvedli; je však velice obtížné, ne-li nemožné, se této implikaci vyhnout,

když jsou jednotlivá jména uvedená v záhlavích zdrojových souborů. V každém případě lze informace o spoluautorství získat ze záznamníku správy verzí a dalších vedlejších zdrojů, jako například z archivu mailing listů, takže vyloučením ze zdrojových souborů se žádná informace ani tak neztratí.[25]

Pokud se váš projekt rozhodne neuvádět jednotlivá jména ve zdrojových souborech, dejte si pozor, abyste nezašli do extrému. Mnoho projektů má například tzv. `contrib/`, tj. oblast, kde se uchovávají malé nástroje a pomocné skripty, jež jsou často dílem lidí, kteří se jinak na projektu nijak nepodílejí. U těchto souborů je v pořádku, že obsahují jména autorů, protože nejsou ve skutečnosti udržovány projektem jako takovým. Na druhou stranu, pokud takovýto nástroj či soubor začnou ostatní účastníci projektu vylepšovat, můžete jej přesunout do méně izolované lokace a – za předpokladu, že s tím bude jeho původní autor souhlasit – z něj můžete odstranit i autorovo jméno, kód pak bude vypadat jako každý jiný zdroj udržovaný komunitou. Pokud je původní autor na podobné kroky citlivý, je možno přijmout kompromisní řešení, například:

```
# indexclean.py: Odstraňte stará data z indexu Scanley
#
# Původní autor: K. Maru <kobayashi@yetanotheremailservice.com>
# Aktuálně udržuje: The Scanley Project <http://www.scanley.org/>
#                   a K. Maru.
#
# ...
```

Těmto kompromisům je však vždy lépe se vyhnout, je-li to možné, a většina autorů se nechá přesvědčit, jsou totiž rádi, že se jejich příspěvek stává důležitější součástí projektu.

Musíte však neustále pamatovat na to, že jádro a periférie každého projektu jsou spojité nádoby. Hlavní soubory zdrojového kódu pro software jsou samozřejmě součástí jádra, a měly by tak být chápány jako objekty udržované komunitou. Naopak, doprovodné nástroje nebo části dokumentace mohou být dílem jednotlivých autorů, kteří je v zásadě uchovávají samostatně, ačkoliv mohou být s projektem asociovány nebo dokonce distribuovány. Není zapotřebí aplikovat na každý soubor pravidlo „univerzální velikosti“, pokud se dodržuje zásada, že zdroje udržované komunitou se nemohou stát samostatným teritoriem.

---

[25] Vlákno mailing listu nazvané *"having authors names in .py files"* na adrese **http://groups.google.com/group/sage-devel/browse_thread/thread/e207ce2206f0beee** uvádí dobrý protiargument, zejména příspěvek Williama Steina. Myslím si, že v tomto případě je zásadní skutečnost, že mnoho těchto autorů pochází z prostředí (akademická komunita matematiků), kde je přímé uvádění autorů normou a je vysoce ceněno. Za těchto okolností by nejspíše bylo vhodné uvádět jména autorů do zdrojových souborů, společně s přesným popisem jejich jednotlivých prací, neboť většina potenciálních přispěvatelů bude tento typ ocenění práce očekávat.

Poměr automatizace

Snažte se, aby lidé nemuseli dělat to, co dokážou udělat stroje. Je empiricky prokázáno, že automatizace běžného úkolu ušetří nejméně desetinásobek úsilí, které by musel vývojář vyvinout na jednorázové manuální provedení téhož úkolu. U častějších nebo velmi komplikovaných úkolů může tento poměr narůst nejméně na dvacetinásobek.

Představte si, že jste sami „projektový manažer“, ne pouze jeden z vývojářů, tato představa je v mnoha ohledech užitečná. Jednotliví vývojáři jsou někdy přespříliš zabředlí v práci na nižší úrovni, aby si dokázali udělat širší obrázek o projektu a uvědomili si, že každý z nich mrhá úsilím při manuálním provádění úkolů, které lze automatizovat. I ti, kdo si uvědomují, že řešení problému nemůže zabrat moc času: protože každé jednotlivé provedení úkolu přeci není nijak velká zátěž, nikdo se nenamáhá, aby s tím něco udělal. Co činí automatizaci žádoucí je fakt, že tyto malé zátěže jsou násobeny tím, kolikrát je musí každý vývojář provést a že *toto* číslo se pak ještě násobí počtem vývojářů.

V tomto kontextu používám termín „automatizace“ v širším slova smyslu, tj. označuji jím nejen opakované akce, kde se střídají jen jedna či dvě proměnné, ale i veškeré typy technické infrastruktury, která pomáhá lidem. Minimální standardní automatizace, která je dnes zapotřebí k chodu projektu, byla popsána v kapitole **3. Technická infrastruktura**, ale každý projekt může mít i své vlastní problémy. Například skupina pracující na dokumentaci by ráda využívala webovou stránku, na níž by byly nonstop zobrazeny poslední aktualizované verze dokumentů. Jelikož se dokumentace často zapisuje ve značkovacím jazyku, např. XML, při vytváření zobrazitelného nebo stáhnutelného dokumentu může být zapotřebí určitý kompilační krok, často dosti komplikovaný. Uspořádání webové stránky, kde by toto kompilování probíhalo automaticky při každém potvrzení změny v kódu, může být sice složité a časově náročné—, ale rozhodně se vyplatí, i kdyby vás to stálo jeden či více dnů práce. Celkový prospěch z toho, že máte kdykoli k dispozici aktualizované stránky, je samozřejmě obrovský, byť by se vám mohlo zdát, že cena, kterou platíte za to, že tyto stránky k dispozici *nemáte*, spočívá jen v drobné jednorázové nepříjemnosti pro jednotlivého vývojáře.

Pokud si tyto stránky vytvoříte, nejen že přestanete plýtvat časem, ale zbavíte se také mrzutostí a frustrací, které pramení z lidských chyb (ty jsou nevyhnutelné) při manuálním provádění komplikovaných postupů. Vícestupňové, deterministické operace – přesně pro tohle byly počítače vynalezeny; lidi si šetřete na zajímavější úkoly.

Automatizované testování

Automatizovaný zkušební chod je užitečný u všech softwarových projektů, ale zvláště pro open source projekty, neboť automatizované testy (zejména regresní) umožňují vývojářům snadno měnit kód v oblastech, s nimiž nejsou dobře obeznámeni, a tím podněcuje další průzkumné práce. Jelikož se porušení funkcionality zjišťuje ručně velice obtížně—, musíte v podstatě tipovat, co jste mohli poškodit a ověřovat, že se tak nestalo—automatizovaný způsob detekce těchto chyb ušetří projektu *mnoho* času.

Lidé budou také méně nervózní při refaktorování velkých částí kódu, a přispěje se tak k dlouhodobější udržitelnosti softwaru.

Regresní testování

Regresní testovaní znamená testování na opakovaný výskyt již opravených chyb. Účelem regresního testování je redukovat možnost, aby chyby kódu nečekaně ukončily software. S postupným narůstáním softwarového projektu a jeho rostoucí komplikovaností se stabilně zvyšuje i možnost výskytu takovýchto nečekaných vedlejších účinků. Dobrý design může snížit rychlost, s níž tato možnost narůstá, ale nemůže problém zcela eliminovat.

V důsledku toho mají mnohé projekty *testovací sadu*, tedy samostatný program, který spouští projektový software způsobem, který již dříve vyvolával specifické chyby. Pokud testovací sada úspěšně vyvolá jednu z těchto chyb, označuje se to jako *regrese*, což znamená, že něčí změna nečekaně obnovila dříve opravenou chybu.

Viz také **http://en.wikipedia.org/wiki/Regression_testing.**

Regresní testování však není žádný všelék. Důvodem je, že fungují nejlépe u programů s dávkovým rozhraním (batch-style). Software, který se obsluhuje především prostřednictvím grafického uživatelského rozhraní, je mnohem těžší řídit programaticky. Další problém spočívá v tom, že vlastní framework sady na regresní testování je často velice složitý, a vyžaduje tak nemalé úsilí na jeho naučení a udržování testů.

Redukce této složitosti je jedna z užitečných věcí, kterou byste mohli udělat, ač to může být výrazně časově náročnější. Čím snadněji se do sady přidávají nové testy, tím více vývojářů to bude dělat a tím méně chyb se dožije data vydání softwaru. Veškeré úsilí věnované zjednodušení zápisu testů se vám bude mnohonásobně vyplácet po celou dobu životnosti projektu.

Mnoho projektů se drží pravidla „*Don't break the build!*" (Nenarušujte build), což znamená, že se nemá potvrzovat změna, která znemožní kompilování nebo spuštění softwaru. Osoba, která poruší build, je zpravidla terčem výsměchu a popichování. Projekty s regresní testovací sadou mají často zavedeno pravidlo, které je důsledkem pravidla výše uvedeného: nepotvrzujte žádnou změnu, která způsobuje selhání testů. Tato selhání se nejsnadněji zjišťují, pokud se automaticky na noc spouští celá testovací sada, jejíž výsledky se pak zasílají do vývojářských mailing listů nebo na určený mailing list s výsledky testů; další příklad užitečné automatizace.

Většina dobrovolných vývojářů si ráda prodlouží pracovní dobu a napíše regresní test, pokud je testovací systém srozumitelný a snadno se s ním pracuje. Průvodní změny u testů jsou považovány za velice odpovědnou činnost, je to však také skvělá příležitost ke spolupráci: obvykle si dva vývojáři rozdělí práci na opravě chyby, jeden bude zapisovat vlastní opravu a druhý bude zapisovat test. Druhý vývojář má většinou více práce, a jelikož je zapisování testu už tak méně záživnou činností než vlastní oprava chyby, je velice důležité, aby testovací sada celou tuto činnost neznepříjemnila ještě více.

Některé projekty jdou dokonce dál, vyžadují totiž, aby *každou* opravu chyby nebo novou funkční vlastnost doprovázel nový test. Zda je to dobrý nápad či nikoli, závisí na mnoha faktorech: povaze softwaru, složení vývojového týmu a složitosti zápisu nových testů. Projekt CVS (**http://www.cvshome.org/**) toto pravidlo dlouho dodržoval. Teoreticky je to správná metoda, jelikož je CVS software pro správu verzí, a je tak velice citlivý na riziko svévolného zničení či nesprávné manipulace s uživatelskými daty. V praxi nastává problém, neboť regresní testovací sada CVS je jeden veliký shellový skript (zábavně pojmenovaný `sanity.sh`, tj. příčetnost), který lze obtížně číst, upravovat či rozšiřovat. Obtížnost přidávání nových testů, v kombinaci s požadavkem, aby byly záplaty vždy doplněny novými testy, znamená, že CVS účinně odrazuje od tvorby záplat. Když jsem pracoval na CVS, viděl jsem, jak lidi někdy začali a dokonce dokončili záplatu pro vlastní kód CVS, ale své úsilí vzdali, když se dozvěděli, že je u každé záplaty zapotřebí přidat do `sanity.sh` také nový test.

Běžně se nad tvorbou nového regresního testu stráví více času než nad opravou původní chyby. CVS však tento jev dohnala do extrému: člověk mohl strávit dlouhé hodiny marných pokusů o správné koncipování testu, jediná změna v 35.000 řádcích Bourne shell skriptu s sebou totiž nese veliké množství nepředstavitelných komplikací. Stejně dlouhou dobu vrčeli vývojáři CVS, když měli přidat nový test. Tato situace vznikla díky tomu, že jsme nezvážili poměr automatizace. Je pravda, že nejtěžším úkolem by bylo jednoduché přepnutí na opravdový testovací framework – ať již zhotovený na zakázku nebo běžně zakoupený.[26] Jelikož jsme však tuto otázku zanedbali, během několika let na tom projekt těžce prodělal. Kolik oprav chyb a nových funkčních vlastností dnes není v CVS kvůli tak drobné překážce, jako je směšná testovací sada? Přesné číslo sice neznáme, ale je určitě několikanásobně větší než počet oprav či nových funkčních vlastností, kterých se mohli vývojáři vzdát, jen aby vyvinuli nový testovací systém (nebo správně integrovali některý z běžně prodávaných systémů). Tento úkol by zabral pouze omezený čas, zatímco trest za používání aktuální testovací sady bude trvat věčně, pokud se s tím něco zásadního neudělá.

Problém není v tom, že je špatné mít přísné požadavky na zapsání testů, ani v tom, že by zapisování testovacího systému jako Bourne shell skript bylo nutně špatné. Může fungovat dobře, podle toho, jak jej budete koncipovat a co je zapotřebí testovat. Problém je jednoduše v tom, že když se testovací systém stane výraznou překážkou ve vývoji, je nutno s tím něco udělat. To samé platí i pro jakýkoli rutinní proces, z něhož se stane bariéra nebo kritické místo projektu.

Ke všem uživatelům se chovejte jako k potenciálním dobrovolníkům

Každá interakce s uživatelem je příležitostí, jak získat nového dobrovolníka. Když si uživatel udělá čas a napíše vzkaz na jeden z projektových mailing listů nebo podá bug report, prokáže svůj potenciál podílet se na projektu více než většina ostatních uživatelů (kteří se projektu vůbec neozvou).

---

[26] Všimněte si, že bychom tím pádem nepotřebovali konvertovat všechny stávající testy do nového frameworku; oba by mohly vesele fungovat vedle sebe, staré testy by byly konvertovány pouze, pokud by bylo nutné je měnit.

Tohoto potenciálu se držte: pokud popíše chybu, poděkujte mu za zprávu a zeptejte se jej, zda by se nepokusil ji sám opravit. Pokud napíše, že v seznamu FAQ (nejčastější dotazy) chybí jedna důležitá otázka nebo že dokumentace programu je v určitém směru nedostatečná, problém uznejte (předpokládejme, že skutečně existuje) a zeptejte se jej, zda by měl zájem napsat chybějící materiál sám. Většina uživatelů bude samozřejmě na rozpacích. Za zeptání ale nic nedáte a při každém takovém dotazu ostatním účastníkům fóra připomenete, že do projektu se může skutečně zapojit každý.

Neomezujte se na pouhé získávání nových vývojářů a zapisovatelů dokumentace. Například, i zaškolení lidí na sepisování dobrých bug reportů se dlouhodobě vyplatí, pokud u každého jednoho zájemce nestrávíte *příliš* mnoho času a pokud budou absolventi tohoto školení předkládat v budoucnu více bug reportů – což pravděpodobně budou, pokud se u svého prvního bug reportu dočkají konstruktivní reakce. Konstruktivní reakcí nemusí být zrovna oprava chyby, ačkoliv ta je vždycky ideální; může to však klidně být prosba o další informace nebo pouhé potvrzení, že popsané chování *je* skutečně chyba. Lidé chtějí, aby jim bylo nasloucháno. Až v druhé řadě chtějí, abyste jim opravili chyby. Druhé přání jim nemusíte vždy splnit včas, ale to první jim rozhodně splnit můžete (přesněji ne vy, ale projekt jako celek).

Vývojáři by tudíž neměli projevovat zlost vůči lidem, kteří v dobrém úmyslu podávají vágní či zmatené bug reporty. Tohle mě vždycky dopálí; vidím, že je to častá reakce vývojářů na různých mailing listech v open source projektech, a škoda, která tím vzniká, je citelná. Nešťastný začátečník vyvěsí zcela nepoužitelný vzkaz:

> Zdravím, nedaří se mi spustit Scanley. Při každém zapnutí to prostě spadne. Má s tím ještě někdo stejný problém?

Vývojář—, který podobný vzkaz viděl už tisíckrát a nedochází mu, že onen začátečník ještě nikdy—, odpoví něco ve stylu:

> A co od nás čekáš, když nám k tomu nedáš ani pořádné informace? Jemine. Tak nám pošli aspoň pár podrobností jako třeba verze Scanley, tvůj operační systém a přesnější popis té chyby.

Tento vývojář není schopen nahlížet věci z perspektivy uživatele a vůbec nezvážil, jaký účinek může mít jeho reakce na všechny *ostatní*, kteří tuto komunikaci sledují. Samozřejmě, že uživatel bez programovacích zkušeností a bez předchozích zkušeností s hlášením chyb nebude vědět, jak se má správně napsat bug report. Jak se k takovéto osobě správně zachovat? Vzdělejte ji! A udělejte to tak, aby se ráda vracela a dozvídala se více a více:

> Je nám líto, že máte s programem problémy. Potřebujeme však od Vás více informací, abychom měli lepší představu, co se u Vás na počítači děje. Napište nám prosím, jakou verzi Scanley používáte, jaký máte operační systém a přesný popis dané chyby. Nejlepší bude, když nám zašlete zápis uvádějící přesné příkazy, které spouštíte, a výstupy, které tyto příkazy produkují. Další informace vizte na adrese **http://www.scanley.org/how_to_report_a_bug.html**.

Tento způsob reagování na dotazy je při získávání potřebných informací od uživatele podstatně efektivnější, protože je psán tak, aby odpovídal perspektivě uživatele. Zaprvé, uživateli vyjadřujete pochopení, svou účast: *Máte problém; chápeme, že je to pro vás nepříjemné.* (Tento aspekt není při každé reakci na bug report nezbytný; závisí na vážnosti problému a na tom, jak naštvaně na vás uživatel působí.) Zadruhé, namísto ponižování za to, že neví, jak se má hlásit chyba, je uživatel poučen, jak se to dělá, a to dostatečně podrobně, aby bylo toto hlášení užitečné—například, mnoho uživatelů netuší, že výzva „ukažte nám tu chybu“ znamená: „ukažte nám přesný text chyby, bez vynechávek a zkracování.“ Při první komunikaci a spolupráci s takovýmto uživatelem musíte být v těchto bodech velice přesní. A konečně, tato reakce rovněž nabízí odkaz na detailnější a kompletní pokyny pro hlášení chyb. Pokud bylo vaše jednání s uživatelem úspěšné, většinou si pak udělá čas, přečte si, co je v odkazovaném dokumentu napsáno, a bude to dodržovat. To samozřejmě znamená, že musíte mít dokument předem připraven. Měl by podávat jasné instrukce o tom, jaký druh informací by váš vývojový tým rád viděl v každém bug reportu. V ideálním případě by se měl také postupně dále rozvíjet v reakci na konkrétní typy vynechávek a nesprávných hlášení, kterých se uživatelé vašeho projektu nejčastěji dopouštějí.

Pokyny pro hlášení chyb (bug reports) u projektu Subversion jsou po formální stránce jednoduchým standardním příkladem takových instrukcí (viz v příloze **D. Vzorové pokyny pro hlášení chyb (bugů)**). Povšimněte si, že končí výzvou k zaslání záplaty pro opravu dané chyby. Ne že by tato výzva vedla k vyššímu poměru záplata/hlášení—; většina uživatelů, kteří jsou schopni opravovat chyby, již ví, že každou záplatu vítáme, a nemusíme je na to výslovně upozorňovat. Skutečným cílem této výzvy je zdůraznit všem čtenářům, zejména nově příchozím do projektu nebo nováčkům ve svobodném softwaru vůbec, že celý projekt žije z příspěvků dobrovolníků. V jistém smyslu nenesou aktuální vývojáři projektu větší odpovědnost za opravu chyb než osoby, které chyby hlásí. Jedná se o důležitou skutečnost, o níž mnoho nových uživatelů nemá tušení. Jakmile si ji uvědomí, častěji nám pak pomáhají s tvorbou oprav, když už ne přímými příspěvky ke kódu, tak alespoň důkladnějšími návody pro reprodukci chyby nebo nabídkou otestování oprav, které zaslali ostatní. Našim cílem je, aby si všichni uživatelé uvědomili, že neexistuje žádný *apriorní* rozdíl mezi nimi a lidmi, kteří na projektu pracují—, hlavní totiž je, kolik času a úsilí člověk do projektu vloží, nikoli kým je.

Varování před zlostnými reakcemi na vzkazy a dotazy se samozřejmě netýká vzkazů a dotazů od neslušných uživatelů. Lidé občas zašlou bug report nebo stížnost, která se, bez ohledu na její informační hodnotu či obsah, nese v posměšném až opovržlivém tónu vůči projektu za některé jeho chyby či selhání. Tito lidé střídavě urážejí a lichotí, jako například osoba, která zaslala na mailing list Subversion následující dotaz:

> Jak to, že po téměř 6 dnech nemáte ještě vyvěšeny žádné binární soubory pro Windows platformu?!? Je to pořád to samé a už mi to leze na nervy. Proč nejsou tyhle věci automatizované, aby byly hned k dispozici?? Myslel jsem si, že když vyvěsíte „RC“ Build, chcete, aby ho uživatelé otestovali, ale přitom nám k tomu vůbec neposkytnete žádné nástroje. K čemu je pak doba stabilizace, když uživatelům nenabídnete žádné prostředky k testování??

Počáteční reakce na tento spíše plamenný vzkaz byla překvapivě mírná: lidi poukazovali na fakt, že projekt ve své zveřejněné politice jasně uvádí, že nebude poskytovat žádné oficiální binární soubory, a s různým stupněm rozhořčení dotyčnému doporučovali, aby se tedy práce na souborech ujal dobrovolně sám, když jsou pro něj tak důležité. Věřte nebo ne, jeho další vzkaz začínal takto:

> Především bych chtěl začít tím, že Subversion je dle mého názoru skvělý a cením si úsilí všech, kdo se na něm podíleli. (...)

... a pak zase začal projekt plísnit za to, že neposkytuje binární soubory, aniž by se nabídl, že s tím jako dobrovolník něco udělá. Po tomto vzkazu se na něj obořilo nějakých 50 lidí a nemůžu říct, že by mi to vadilo. Nulová tolerance vůči hrubosti obhajované části **Nezdvořilé chování potlačte hned v zárodku** v kapitole **2. Zahájení projektu** se týká pouze lidí, s nimiž projekt uchovává (nebo by chtěl uchovávat) trvalou interakci. Když ale někdo od prvopočátku jasně demonstruje, že bude pouze dštít síru, nemáme jediný důvod chovat se k němu vstřícně.

K těmto situacím dochází naštěstí jen zřídka a ještě méně u projektů, které se všemožně snaží jednat s uživateli konstruktivně a zdvořile od jejich první interakce.

Podělte se o řídící i technické úkoly

Tíhu řízení i technických úkolů spojených s chodem projektu si můžete vzájemně rozdělit s kolegy. S rostoucí složitostí celého projektu se stále větší část práce týká řízení lidí a informačního toku. Není jediný důvod nerozdělit si tuto pracovní zátěž s kolegy, což nemusí nutně vyžadovat dodržování klasické vertikální hierarchie—v praxi se stále častěji setkáváme se síťovou topologií typu peer-to-peer než s vojenskou příkazovou strukturou.

Někdy jsou manažerské role formalizovány, někdy vzniknou spontánní. V projektu Subversion máme funkce patch manager (manažer záplat), translation manager (manažer překladu), documentation manager (manažer dokumentace), issue manager (manažer problémů) (byť neoficiální) a release manager. Některé tyto role jsme vědomě vytvořili, jiné vznikly spontánně; s rostoucím projektem předpokládám vznik mnohem více rolí. Tyto role a některé další budou na následujících stránkách blíže rozebrány (s výjimkou release managera, o němž jsme pojednali v částech **Správce release** a **Diktatura vlastníka release** v této kapitole).

Při čtení popisu role si všimněte, že žádná z nich nevyžaduje výhradní kontrolu nad příslušnou oblastí projektu. Issue manager nebrání ostatním v provádění změn v issues databázi, FAQ manager (manažer sekce s nejčastěji kladenými otázkami) netrvá na tom, aby tyto otázky mohl upravovat pouze on a tak dále. Tyto role s sebou nesou odpovědnost bez monopolizace. Důležitou součástí práce každého z těchto manažeru je zaznamenat, kdy se na práci v dané oblasti projektu podílejí i ostatní lidé, a naučit je, jak provádět jednotlivé úkoly stejným způsobem jako manažer, tak aby se kolektivní úsilí násobilo a navzájem nekolidovalo. Manažeři jednotlivých oblastí projektu také dokumentují procesy, které používají při provádění své práce; jakmile tedy někdo z nich projekt opustí, jiný na ně může okamžitě navázat.

Někdy dochází ke konfliktu: dva a více lidí chce zastávat stejnou roli. Na tento problém neexistuje jedno správné řešení. Můžete navrhnout, aby každý ze zájemců vyvěsil svůj návrh („žádost“) a všichni committeři pak hlasovali, který je nejlepší. Je to však řešení poněkud těžkopádné a někdy i směšné. Zjistil jsem, že je lepší požádat všechny kandidáty, aby si celou věc ujasnili sami mezi sebou. Obvykle tak učiní a s výsledkem jsou mnohem spokojenější, než kdyby byl celý problém rozhodnut formou nařízení zvenčí.

Patch Manager (manažer záplat)

V projektu svobodného softwaru, který dostává řadu záplat, může být evidence všech došlých záplat a rozhodnutí, která k nim byla učiněna, doslova noční můrou, zejména pokud se tato evidence provádí decentralizovaně. Většina záplat přichází ve formě vzkazů do vývojářského mailing listu daného projektu (některé záplaty se však také mohou nejprve objevit v issue trackeru nebo na externích webových stránkách), záplata se pak může ubírat mnoha různými směry.

Někdy ji může někdo zrevidovat, najít v ní problémy a poslat ji zpět původnímu autorovi, aby ji pročistil. To zpravidla vede k iteračnímu postupu—vše je zobrazeno v mailing listu—kdy původní autor vyvěšuje revidované verze záplaty, dokud k ní kontrolor již nemá žádné další připomínky. Ne vždy lze snadno určit, kdy je celý proces dokončen: pokud kontrolor záplatu potvrdí, pak je samozřejmě celý cyklus uzavřen. Ale pokud záplatu nepotvrdí, může to být proto, že zkrátka neměl čas nebo nemá commit access a nepodařilo se mu najít vývojáře, který by udělal commit za něj.

Další častou reakcí na záplatu je nevázaná diskuse, nikoli nutně o vlastní záplatě, ale o tom, zda je správná celá koncepce, která se za danou záplatou skrývá. Například, určitá záplata je sice schopna opravit některou chybu, ale projekt by raději tuto chybu opravil jiným způsobem, například v rámci řešení obecnějšího typu problémů. Často nebývá tato skutečnost předem známa a až vlastní záplata podnítí tento objev.

Příležitostně se nově vyvěšená záplata setká s naprostým mlčením. Obvykle je to proto, že žádný vývojář nemá aktuálně čas záplatu zkontrolovat, tak doufá, že to udělá někdo jiný. Jelikož není přesně stanoveno, jak dlouho se může čekat na to, až někdo zvedne hozenou rukavici, a mezitím se neustále objevují nové priority, záplata může velice snadno propadnout sítem, aniž by to někdo cíleně chtěl. Projekt tak může snadno přijít o užitečnou záplatu a kromě toho existují i další škodlivé vedlejší účinky: odrazuje autora, který na záplatě pracoval, a projekt se jako celek jeví odtažitě, zejména pro lidi, kteří zvažují, že by tvořili záplaty.

Úkolem patch managera je zajistit, aby žádná záplata „nepropadla sítem“. To se zajistí tak, že každá záplata bude dotáhnuta do určitého stabilního stadia. Patch manager sleduje všechna vlákna mailing listu, která vycházejí z vyvěšení záplaty. Pokud vlákno končí potvrzením záplaty, nedělá nic. Pokud dojde k opakování cyklu kontrola/revize končícího finální verzí záplaty, ale bez potvrzení změn, patch manager zaeviduje issue s odkazem na finální verzi a na související vlákno mailing listu, a vývojáři tak

mají neustále k dispozici záznam, kterému se mohou kdykoli později věnovat. Pokud záplata reaguje na existující issue, patch manager okomentuje toto issue relevantními informacemi, aniž by otevíral nové issue.

Pokud se na záplatu neobjeví žádná reakce, patch manager vyčká několik dní, pak se začne ptát, jestli se někdo chystá záplatu zkontrolovat. Na tento dotaz zpravidla obdrží reakci: vývojář může buď vysvětlit, proč si myslí, že by daná záplata neměla být použita, nebo ji může zrevidovat, v kterémžto případě se celý postup začne odvíjet způsobem výše popsaným. Pokud se přesto žádná reakce nedostaví, patch manager může nebo nemusí dle vlastního uvážení evidovat issue o dané záplatě, původce záplaty však alespoň obdrží *nějakou* reakci.

Díky patch managerovi už vývojářský tým Subversion ušetřil spoustu času a energie. Bez této osoby, která na sebe bere odpovědnost, by se musel každý vývojář neustále trápit otázkami jako: „Pokud nebudu mít čas reagovat na tuto záplatu hned teď, můžu se spoléhat na to, že to někdo udělá? Měl bych na to dohlédnout? Ale když na to budou dohlížet i ostatní ze stejného důvodu jako já, budeme zbytečně zdvojovat úsilí.“ Patch manager za této situace eliminuje nutnost těchto úvah. Každý vývojář může učinit rozhodnutí, které je pro něj ve chvíli, kdy poprvé spatří záplatu, správné. Pokud chce provést revizi záplaty, může ji provést—, patch manager se podle toho zařídí. Pokud chce záplatu úplně ignorovat, žádný problém; patch manager se postará o to, aby záplata neupadla v zapomnění.

Jelikož celý tento systém funguje pouze v případě, že se lidé mohou bezmezně spolehnout na přítomnost patch managera, tato role musí být zastávána formálně. V projektu Subversion jsme osobu pro tuto roli hledali formou inzerátu na vývojovém a uživatelském mailing listu, přihlásilo se několik dobrovolníků a vzali jsme hned prvního, který na inzerci reagoval. Když musela tato osoba z funkce odstoupit (viz část **Personální změny** dále v této kapitole), celý postup jsme zopakovali. Nikdy jsme se nepokoušeli nechat zastávat jednu roli několik lidí, především kvůli nezbytné komunikaci, kterou by mezi sebou museli vést; možná by však při velkém objemu předkládaných záplat měl takovýto víceosobový patch manager smysl.

Translation Manager (manažer překladu)

V softwarových projektech může anglický termín „translation“ odkazovat na dvě různé věci. Může znamenat překlad softwarové dokumentace do jiných jazyků, nebo překlad vlastního softwaru—, tj. programové zobrazení chyb a nápověd v jazyku preferovaném uživatelem. Obě tyto uvedené činnosti jsou složité úkoly, máme-li však zavedenou správnou infrastrukturu, lze je z velké části oddělit od ostatního vývoje. Jelikož jsou si oba úkoly jistým způsobem podobné, může být vhodné (v závislosti na vašem konkrétním projektu) mít jediného translation managera pro řešení obou úloh nebo dva různé translation managery.

V projektu Subversion máme jednoho translation managera, který se stará o obě činnosti. Sám osobně samozřejmě překlady nepíše—, možná s jedním či dvěma někdy pomohl, ale v době, kdy byla sepsána tato publikace, by musel umět mluvit deseti jazyky (dvanácti, včetně dialektů), aby mohl na všech

těchto úlohách pracovat! Namísto toho řídí týmy dobrovolných překladatelů: pomáhá jim ve vzájemné koordinaci a koordinuje tyto týmy se zbývajícími účastníky projektu.

Translation manager je zčásti nezbytný proto, že překladatelé se liší od vývojářů. Mají pouze malé nebo vůbec žádné zkušenosti s prací na úložišti správy verzí nebo se skutečnou prací v rámci širšího týmu dobrovolníků. Ale v jiných ohledech jsou často tím nejlepším druhem dobrovolníků: lidé se specifickými odbornými znalostmi, kteří zaznamenali potřebu těchto znalostí a rozhodli se, že se do projektu zapojí. Obyčejně se rádi učí a pracují. Potřebují jen někoho, kdo jim řekne jak. Translation manager zajistí, aby překlady proběhly způsobem, který nebude zbytečně zasahovat do běžného vývoje. Slouží zároveň jako určitý zástupce překladatelů, jako jejich jednotný orgán, kdykoli je zapotřebí informovat vývojáře o technických změnách, které jsou zapotřebí k podpoře překladatelského úsilí.

Nejdůležitější dovedností této pozice je tedy diplomacie, nikoli technika. Například v projektu Subversion máme zásadu, že na všech překladech musí pracovat alespoň dva lidé, jinak je totiž revize textu nemožná. Když se objeví nový dobrovolník, který projektu Subversion nabízí překlad, například do malgaštiny, translation manager jej buď musí spojit s někým, kdo za posledních šest měsíců vyvěsil vzkaz, v němž vyjádřil svůj zájem překládat do malgaštiny, nebo tohoto zájemce slušně požádá, aby si našel ještě *jednoho* překladatele do malgaštiny, který s ním bude pracovat jako partner. Jakmile máme k dispozici dostatek lidí, manažer je vybaví náležitým commit access, informuje je o obecných zásadách projektu (jako například jak se mají psát zprávy a hlášení) a pak na ně dohlíží, zda tyto podmínky skutečně dodržují.

Rozhovory mezi translation managerem a vývojáři nebo mezi translation managerem a překladatelskými týmy zpravidla probíhají v původním jazyku projektu—, tj. v jazyku, z něhož se pořizují všechny překlady. U většiny projektů svobodného softwaru je tímto jazykem angličtina, je však jedno, jaký jazyk to je, stačí, když se na něm shodnou účastníci projektu. (Angličtina se asi přesto nejvíce hodí pro projekty, které chtějí zaujmout širokou mezinárodní vývojářskou komunitu.)

Rozhovory *v rámci* konkrétního překladatelského týmu zpravidla probíhají v konkrétním jazyce daného týmu, jedním z dalších úkolů translation managera je nastavit mailing list, který bude vyhrazen každému jednotlivému týmu. Překladatelé tak mohou volně diskutovat o své práci, aniž by rozptylovali nebo mátli ostatní lidi na hlavních mailing listech projektu, kteří by jazyku překladu pravděpodobně nerozuměli.

Internacionalizace versus lokalizace

Internacionalizace (I18N) a *lokalizace (L10N)* označují procesy, které upravují program pro práci v jiných jazykových a kulturních prostředích, než pro které byl původně napsán. Tyto termíny se často chápou jako vzájemně zaměnitelné, nejedná se však o úplně stejnou věc. Jak píše **http://en.wikipedia.org/wiki/G11n:**

> Rozdíl mezi těmito termíny je sice nepatrný, zato však velice důležitý: Internacionalizace je úprava produktů pro jejich případné použití prakticky kdekoli, zatímco lokalizace je přidání speciálních funkčních vlastností, které umožní používání produktů na *specifickém* místě.

Například úprava vašeho softwaru na bezztrátovou manipulaci s textovým kódováním Unicode (**http://en.wikipedia.org/wiki/Unicode**) je internacionalizací, jelikož se v ní nejedná o jeden konkrétní jazyk, ale o přijetí textu z jakéhokoli jazyka. Na druhou stranu, úprava softwaru pro tisk všech chybových hlášení ve slovinštině, jakmile software zjistí, že pracuje ve slovinském prostředí, je lokalizací.

Úloha translation managera tedy v zásadě spočívá v lokalizaci, nikoli internacionalizaci.

Documentation Manager (manažer dokumentace)

Průběžné aktualizování softwarové dokumentace je úkol bez konce. Každá nová funkční vlastnost nebo doplněk, které přecházejí do kódu, jsou potenciální příčinou změny dokumentace. Kromě toho, jakmile projektová dokumentace dosáhne určitého stupně kompletnosti, zjistíte, že mnoho zaslaných záplat se vlastně týká dokumentace, nikoli kódu. Důvod je prostý, drtivá většina lidí je totiž schopna opravit chyby v běžném textu, nikoli však v kódu: všichni uživatelé jsou čtenáři, ale jen hrstka z nich jsou programátoři.

Je obvykle podstatně snazší kontrolovat a aplikovat záplaty dokumentace než záplaty kódu. Provádí se jen málo testů, pokud vůbec nějaké, a kvalitu změny lze snadno a rychle vyhodnotit pouhým pročtením. Jelikož je množství veliké, ale obtížnost kontroly dosti nízká, poměr administrativy vůči produktivní práci je u záplat dokumentace vyšší než u záplat kódu. Dále, většina záplat bude pravděpodobně potřebovat jistý druh úprav, aby byl zachován jednotný autorský ráz dokumentace. V mnoha případech se budou záplaty vzájemně překrývat nebo ovlivňovat, a musí být tedy před potvrzením upraveny tak, aby si řádně odpovídaly.

Vzhledem k nezbytnosti zpracování dokumentačních záplat a skutečnosti, že kódová základna musí být průběžně monitorována, aby byla dokumentace stále aktuální, je velice rozumné mít pro tento úkol určenu jednu osobu nebo menší tým. Mohou evidovat, kde a jak přesně dokumentace zaostává za softwarem, a mít zavedené postupy pro zpracování velkého množství záplat jednotlivým způsobem.

To samozřejmě nebrání dalším lidem v projektu aplikovat průběžně, pokud jim to čas dovolí, další dokumentační záplaty, zejména menších rozměrů. A stejný patch manager (viz části **Patch Manager (manažer záplat)** výše v této kapitole) může evidovat kódové i dokumentační záplaty, zakládat je tam, kam si to přeje vývojářský, respektive dokumentační tým. (Pokud celkové množství záplat překročí lidské možnosti zaznamenávání, bude možná jako první krok nejlepší přejít na režim dvou samostatných patch managerů, tj. pro kód a pro dokumentaci.) Smyslem dokumentačního týmu je mít k dispozici lidi, kteří přijímají svou odpovědnost za uchovávání dokumentace v uspořádaném, aktualizovaném a vnitřně konzistentním stavu. V praxi to znamená, že musí dokumentaci důvěrně znát, musí sledovat kódovou základnu, změny, které *ostatní* v dokumentaci potvrdili, sledovat příchozí záplaty dokumentace a používat všechny tyto informační zdroje, aby všemožným způsobem uchovali dokumentaci v řádném stavu.

Issue manager (manažer problémů)

Počet issues v projektovém bug trackeru roste úměrně s počtem lidí, kteří software používají. Proto, i když opravíte chyby a publikujete stále větší program, musíte očekávat, že počet nedořešených issues stále bezmezně poroste. Také se bude zvyšovat počet zdvojených issues a četnost nedokončených či nedokonale popsaných issues.

Issue manažeři pomáhají tyto problémy mírnit, neboť sledují aktuální dění v databázi a pravidelně se jí prokousávají, přičemž vyhledávají specifické problémy. Jejich nejobvyklejší činností je pravděpodobně uspořádávání příchozích issues, buď proto, že odesílatel zprávy nezadal správně některá pole formuláře, nebo proto, že dotyčné issue je již jednou obsaženo v databázi. Samozřejmě, že čím více je issue manager obeznámen s databází projektových chyb, tím účinněji bude schopen odhalit duplicitní issues—mít několik lidí, kteří se specializují na databázi chyb, je rozhodně výhodnější, než aby se o to pokoušel každý *ad hoc*. Když se skupina těchto lidí snaží provádět tuto činnost decentralizovaně, žádný jednotlivec nemůže získat hlubší vhled do obsahu databáze.

Issue manažeři mohou také napomáhat mapování mezi issues a jednotlivými vývojáři. Když přichází mnoho bug reportů, ne každý vývojář je schopen stejně pozorně číst mailing listy s oznámením issues. Nicméně pokud někdo, kdo zná tým vývojářů, hlídá všechny příchozí issues, může pak rozvážně směrovat pozornost určitých vývojářů na specifické chyby. Tuto činnost je samozřejmě zapotřebí provádět s maximálním ohledem na všechny ostatní činnosti probíhající v rámci projektu a na přání a povahu příjemce. Proto je pro issue manažery často nejlepší, když jsou sami také vývojáři.

Podle toho, jak váš projekt používá issue tracker, issue manažeři mohou také upravovat databázi tak, aby odrážela priority projektu. Například v Subversion máme issues načasovány podle jednotlivých budoucích vydání, takže když se někdo zeptá „Kdy bude opravena chyba X?“, odpovíme „V popříštím release“, ačkoliv nemůžeme uvést přesné datum. Releasy jsou v issue trackeru uvedeny jako cílové milníky – pole dostupné v IssueZilla.[27] Každý release Subversion má zpravidla jednu hlavní novou funkční vlastnost a seznam specifických oprav chyb. Vhodný cílový milník přiřazujeme ke všem issues, které jsou pro daný release naplánovány (včetně nové funkční vlastnosti—ta dostává rovněž issue), zájemci tak mohou sledovat databázi chyb z pohledu časového harmonogramu vydání. Tyto cílové milníky však málokdy zůstávají statické. S výskytem nových chyb se někdy posouvají priority a issues je nutno přesunout z jednoho milníku na jiný, aby si každé vydání nadále zachovalo rozumnou velikost. To – zase – dělají nejlépe lidé, kteří mají celkové povědomí o tom, co je v databázi a jak spolu souvisejí jednotlivá issues.

Další činnost, kterou issue manager provádí, je upozorňování na zastaralost issue. Někdy se chyba opraví náhodně, jako součást nesouvisející změny softwaru, někdy se v rámci projektu změní názor na to, zda je určité chování opravdu chybné. Hledání zastaralých issues není snadné: jediný systematický způsob je projít veškeré issues v databázi. Nicméně s postupem času a rostoucím počtem

[27] Jako issue tracker používáme IssueZilla; nástupce BugZilla.

issues je toto procházení stále méně reálně proveditelné. Od určitého bodu lze databázi udržovat v rozumných mezích pouze v duchu pravidla „rozděl a panuj“: kategorizujte issues okamžitě při jejich přijetí a směřujte je k příslušným vývojářům či týmům. Příjemce se pak o issue stará až do konce její životnosti, vede issue k rozřešení nebo zapomnění, podle potřeby. Když je databáze už takto rozsáhlá, z issue managera se stává celkový koordinátor, tráví méně času vlastním řešením issues a stále více času mu zabírá přesměrovávání issues na správné osoby.

FAQ Manager (manažer sekce s nejčastěji kladenými otázkami)

Údržba FAQ je překvapivě složitý problém. Na rozdíl od většiny ostatních dokumentů v projektu, jejichž obsah předem rozplánovali autoři, je FAQ čistě reaktivní dokument (viz část **Údržba FAQ** v kapitole **2. Zahájení projektu**). Bez ohledu na to, jak je FAQ velké, nikdy nevíte, jak bude vypadat další příspěvek či doplněk. A protože se rozrůstá po kouskách, velice snadno se může celý dokument stát nesourodým a neorganizovaným, může dokonce obsahovat duplicitní nebo poloduplicitní položky. I když nemá žádné evidentní problémy tohoto druhu, často se ve FAQ objevují nepovšimnuté vzájemně závislé položky—křížové odkazy, které by sice měly být provedeny, ale ve skutečnosti nejsou—, protože jednotlivé spolu související položky byly do dokumentu vloženy s ročním odstupem.

Role FAQ managera je vlastně dvojitá. Zaprvé, udržuje celkovou kvalitu FAQ tím, že je neustále alespoň rámcově obeznámen s tématy všech otázek, tzn. když lidé přidají novou položku (otázku), která se již ve FAQ nachází nebo s jinou otázkou jakkoli souvisí, provede vhodnou úpravu. Zadruhé, sleduje mailing listy projektu a další fóra, kde se mohou vyskytnout opakující se problémy či otázky, na základě těchto vstupů pak píše nové položky do FAQ. Tento druhý úkol může být dosti komplikovaný: manažer totiž musí být schopen sledovat určité vlákno, rozpoznat základní otázky, které se v něm objevují, vyvěsit navrhovanou položku FAQ, začlenit komentáře ostatních (jelikož FAQ manažer samozřejmě nemůže být odborníkem na veškerá témata projednávaná ve FAQ) a musí umět dobře odhadnout, kdy je proces ukončen, a položku je tak možno přidat.

FAQ manažer se také často stává standardním odborníkem na formátování FAQ. Správné udržování FAQ s sebou nese množství detailní práce (viz část **Se všemi zdroji zacházejte jako s archivy** v kapitole **6. Komunikace**); pokud FAQ upravují náhodní lidé, mohou některé tyto detaily opomenout. Nevadí, pokud máme po ruce FAQ manažera, který po těchto lidech „uklidí“.

Existují různé svobodné softwary, které vám s údržbou FAQ mohou pomoci. Je vhodné tyto softwary používat, pokud tím nesnížíte kvalitu FAQ; vyvarujte se však přehnané automatizace. Některé projekty se snaží plně automatizovat proces údržby FAQ a umožňují všem zájemcům přispívat a upravovat položky FAQ podobným způsobem jako wiki (viz část **Wiki** v kapitole **3. Technická infrastruktura**). Konkrétně jsem to zažil s Faq-O-Matic (**http://faqomatic.sourceforge.net/**), ačkoliv je možné, že případy, které jsem viděl, byly jen prostým zneužitím, které překračuje původní myšlenku Faq-O-Matic.
V každém případě platí, že zatímco úplná decentralizace údržby FAQ skutečně snižuje pracovní zátěž projektu, na druhou stranu má za následek méně kvalitní FAQ. Neexistuje totiž nikdo, kdo by měl širší obrázek o celém FAQ, neexistuje nikdo, koho byste mohli upozornit na potřebu aktualizace některých

položek nebo na jejich naprostou zastaralost, a nikdo, kdo by sledoval vzájemnou provázanost jednotlivých položek. Ve výsledku pak máme FAQ, které není často schopno poskytnout uživatelům odpovědi na otázky a v nejhorších případech je dokonce uvádí v omyl. K údržbě FAQ ve vašem projektu používejte libovolné nástroje, ale nikdy vás nesmí pohodlnost při používání těchto nástrojů svést k tomu, že snížíte nároky na kvalitu FAQ.

Popis a hodnocení open source nástrojů k údržbě FAQ najdete v článku Seana Michaela Kernera, *The FAQs on FAQs*, na adrese **http://osdir.com/Article1722.phtml**.

Personální změny

Někdy může dobrovolník na odpovědném postu (např. patch manager, translation manager atd.) přestat být schopen plnit povinnosti, které z jeho funkce vyplývají. Může to být kvůli tomu, že se jeho konkrétní práce ukázala jako náročnější, než si původně myslel, nebo kvůli vlivům, které přicházejí zcela zvnějšku: manželství, narození dítěte, nová práce, cokoli.

Když je dobrovolník takto zavalen, obvykle si toho zpočátku ani nevšimne. Přichází to pozvolna, neexistuje žádný konkrétní bod, kdy by si uvědomil, že již není schopen plnit povinnosti, které s sebou daná role přináší. Naopak, zbývající účastníci projektu o něm téměř nevědí. Pak se náhle pustí do horečné činnosti, cítí se totiž provinile, že projekt tak dlouho zanedbával a vyšetří si jednu noc, aby všechno dohonil. Pak od něj zase nebudou přicházet žádné zprávy, to už bude trvat trochu déle než prve; a pak možná zase období horečné aktivity (nebo už ne). Zřídka se však setkáte s dobrovolnou formální rezignací. Dobrovolník pracoval na projektu ve svém volném čase, rezignace by tak znamenala, že si otevřeně přiznává, že jeho volný čas se trvale zkrátil. To si lidi přiznávají velice neradi.

Je tedy na vás a ostatních účastnících projektu, abyste si všimli, co se děje—nebo spíše neděje—, a zeptali se na to i dotyčného dobrovolníka. Dotazování musí být přátelské a prosté veškerého obviňování. Hledáte informaci, nechcete nikoho zahanbovat. Obecně by toto dotazování mělo být viditelné i zbývajícím účastníkům projektu, víte-li však o nějakém důvodu, proč by bylo lepší hovořit s dotyčným spíše soukromně, klidně tak učiňte. Hlavní důvod pro veřejné dotazování je ten, že pokud dotyčný dobrovolník odpoví, že již není schopen svěřený úkol plnit, vytvoříte si tím kontext pro *další* veřejný vzkaz: výzvu pro nového dobrovolníka, který by uvolněnou funkci převzal.

Někdy není dobrovolník schopen vykonávat práci, kterou přijal, ale tento fakt si buď neuvědomuje, nebo si jej nedokáže připustit. Samozřejmě, že na začátku může mít problémy kdekdo, zvláště když je pracovní odpovědnost složitá. Nicméně když někdo jednoduše neplní úkol, který přijal, přestože mu všichni ostatní všemožně pomáhají a radí, pak je jediným řešením, aby dotyčný odstoupil a dal šanci někomu jinému. A když na to někdo neumí přijít sám, musíte mu to prostě říct. Podle mého názoru lze celou tuto situaci řešit v podstatě jediným způsobem, proces je však několikastupňový a každý krok je velice důležitý.

Nejprve se ujistěte, že neblouzníte. Soukromě pohovořte s ostatními účastníky projektu a zjistěte, zda si i oni myslí, že je problém tak vážný, jak ho vidíte vy. I když jste si skutečně jistí, že se nemýlíte, tyto soukromé hovory mají za účel informovat ostatní, že zvažujete možnost požádat dotyčnou osobu o odstoupení z funkce. Většinou nebude nikdo nic namítat—, naopak, budou rádi, že jste s touto nanejvýš trapnou záležitostí sami začali a nemuseli to dělat oni.

Potom *soukromě* kontaktujte dotyčného dobrovolníka a řekněte mu zdvořile, ale narovinu o problémech, na které jste narazili. Buďte konkrétní, uveďte co možná nejvíce příkladů. Určitě se zmiňte o tom, jak se ostatní snažili pomáhat, ale problémy nadále přetrvávaly bez změny k lepšímu. Správně očekáváte, že psaní tohoto e-mailu vám zabere mnoho času, u zpráv tohoto druhu musíte mít všechna svá tvrzení podložena důkazy a argumenty, pokud takové důkazy a argumenty nemáte, raději žádnou zprávu neposílejte. Uveďte, že byste pro uvolněnou funkci rád našel nového dobrovolníka, ale také upozorněte, že do projektu lze přispívat mnoha jinými způsoby. V této fázi však nezmiňujte, že jste o tomto problému už hovořil i s ostatními; nikdo nemá rád, když se o něm lidé domlouvají za zády.

Od této chvíle se celá záležitost může ubírat několika cestami. Nejpravděpodobnější reakce je, že s vámi bude dotyčný dobrovolník souhlasit nebo se nebude chtít za každou cenu přít a svolí k odstoupení. V tomto případě navrhněte, aby své rozhodnutí oznámil sám, pak můžete vyvěsit vzkaz, že za něj hledáte náhradu.

Nebo může souhlasit s tím, že se vyskytly určité problémy, ale požádá vás o trochu strpení (nebo o ještě jednu šanci, v případě funkcí s časově nespojitými úkoly, například release manager). Jak na tuto prosbu zareagujete, je zcela na vás, ale ať už uděláte cokoli, nepřistupujte na ni jenom kvůli tomu, že se vám zdá, že takový rozumný návrh prostě nemůžete jen tak odmítnout. To by celou agonii jen prodlužovalo, nikoli mírnilo. Často se najde dobrý důvod, proč tuto prosbu odmítnout, že totiž takových šancí už bylo mnoho a zrovna proto věci došly až tam, kde momentálně jsou. Nabízím vám příklad takového e-mailu, který jsem adresoval osobě, jež zastávala funkci release managera, ačkoli se pro ni skutečně nehodila:

```
> Pokud si přejete nahradit mě někým jiným, velice rád tuto funkci
> uvolním svému následníkovi. Mám jen jednu prosbu, která - jak doufám
> - není nesmyslná. Rád bych se pokusil ještě o jeden release, abych se
> tentokrát skutečně osvědčil.

Vašemu přání dokonale rozumím (byl jsem na tom podobně!), ale v tomto
případě už nebudeme na žádnou „další šanci“ přistupovat.

Nejedná se ani o první, ani o druhý release, ale snad už o šestý nebo sedmý...
A co vím, u všech předchozích jste byl s výsledky také nespokojen
(už jsme se o tom dříve bavili). Takže „ještě jedna šance“ už několikrát
proběhla. Jedna z těchto šancí už prostě musí být poslední... Myslím si, že
tou poslední šancí je právě onen poslední release.
```

V nejhorším případě může dotyčný dobrovolník vyjádřit naprostý nesouhlas. Pak musíte počítat s tím, že celá záležitost bude sice nanejvýš mrzutá až trapná, ale nezbývá vám, než prosadit svou. Nyní uveďte, že jste o celé věci mluvil i s ostatními (neříkejte však s kým, pokud k tomu nemáte jejich svolení, protože tyto rozhovory byly důvěrné) a že si myslíte, že projektu neprospěje pokračovat za současného stavu věcí. Buďte neústupný, ale nevyhrožujte. Neustále mějte na mysli, že u většiny rolí (funkcí) dochází k personální změně až ve chvíli, kdy někdo nový zahájí svou práci, *nikoli* ve chvíli, kdy předchůdce ve funkci přestane pracovat. Například, pokud se spor týká role issue manager, můžete se vy a další vlivní účastníci projektu kdykoli domáhat nového issue managera. V podstatě není nutné, aby osoba, která tuto funkci dříve zastávala, v ní přestala pracovat, pokud nebude (úmyslně či jinak) sabotovat úsilí dobrovolníka, který byl do ní nově jmenován.

Což nás přivádí ke svůdné myšlence: místo, abyste žádali dotyčnou osobu o rezignaci, proč celou záležitost nezaobalit tak, že byste mu rádi poskytli určitou pomoc? Proč bychom nemohli mít dva issue managery nebo patch managery nebo jakékoli jiné dvě osoby v jedné funkci?

Ačkoliv se tato myšlenka může teoreticky jevit jako dokonalá, obecně to není zrovna dobrý nápad. Důvod, proč manažerské role fungují—, proč jsou užitečné—, je jejich centralizace. Záležitosti, které lze vyřizovat decentralizovaně, jsou již zpravidla takto řešeny. Mít dva lidi v jedné manažerské roli vždy vyžaduje jejich vzájemnou komunikaci a hrozí zde velice ošemetné přesouvání odpovědnosti („Myslel jsem, že lékárničku přineseš ty!“ „Já? Ne, já myslel, že ji přineseš *ty*!“). Samozřejmě, že existují výjimky. Někdy spolu dva lidé pracují velice dobře nebo je povaha dané role taková, že ji lze snadno rozdělit mezi několik lidí. Ale tyto případy vám asi sotva pomohou, když musíte sledovat, jak někdo bez rozmyslu setrvává ve funkci, pro niž se nehodí. Kdyby tento problém uznal hned na začátku, už by o takovou pomoc dávno požádal. V každém případě by bylo neuctivé nechávat někoho mrhat vlastním časem na práci, jíž stejně nikdo nebude věnovat pozornost.

Pokud někoho žádáte o rezignaci, nejdůležitějším aspektem celé záležitosti je důvěrnost: dejte mu prostor, aby se rozhodl sám, aniž by cítil, že jej ostatní pozorují a čekají. Jednou jsem udělal chybu—, dnes vím, že zcela evidentní chybu—, když jsem požádal release managera projektu Subversion o přenechání funkce jiným dvěma dobrovolníkům, tuto žádost jsem však poslal hromadným e-mailem všem třem zainteresovaným osobám. S oběma dobrovolníky jsem již předtím soukromě komunikoval a věděl jsem, že jsou ochotni tuto odpovědnost převzít. Naivně a poněkud necitlivě jsem si myslel, že ušetřím čas a nervy, když jeden e-mail rozešlu všem třem zainteresovaným osobám najednou. Předpokládal jsem, že aktuální release manager si byl všech problémů plně vědom, a že tak velice rychle pochopí racionalitu mých argumentů.

Mýlil jsem se. Release manager se hluboce urazil a nutno dodat, že po zásluze. Jedna věc je být požádán o předání funkce, a jiná věc je být o totéž požádán *před* lidmi, kterým tuto funkci chcete svěřit. Když mi konečně došlo, proč se tak urazil, musel jsem se mu omluvit. Nakonec však se vší důstojností odstoupil a dodnes je v projektu zapojen. Jeho city však byly raněny a nemusím snad ani zmiňovat, že pro nové dva dobrovolníky to taky nebyl zrovna šťastný začátek.

Committeři

Jako jediná formálně odlišná skupina lidí, přítomná ve všech open source projektech, si committeři zaslouží zvláštní pozornost. Committeři jsou nevyhnutelný ústupek diskriminaci v systému, který je jinak tak nediskriminační, jak to jen jde. Termín „diskriminace“ však zde není myšlen pejorativně. Funkce, kterou committeři vykonávají, je zcela nezbytná a žádný projekt by bez nich, myslím, nikdy neuspěl. Řízení jakosti vyžaduje, světe div se, řízení. Vždycky se najde hodně lidí, kteří se domnívají, že jsou kompetentní k provádění změn v programu, a jen málo jich skutečně kompetentních je. Projekt se nemůže opírat o vlastní úsudek lidí; musí stanovit určité standardy a commit access udělovat pouze těm, kteří tyto standardy splňují.[28] Na druhou stranu, když spolu v jednom kolektivu pracují lidi, kteří mohou provést změny přímo, a lidi, kteří nemohou, může to vytvářet pocity nestejného vlivu na projekt. Tato možnost musí být brána v potaz, aby projektu neuškodila.

V části **Kdo hlasuje?** v kapitole **4. Společenská a politická infrastruktura** jsme již probírali mechanismus posuzování nových committerů. V této chvíli se podíváme blíže na standardy a normy, podle nichž jsou noví committeři posuzováni, a jak se má tento proces prezentovat širší komunitě.

Volba committerů

V projektu Subversion volíme committery především dle zásad Hippokratovy přísahy: *Zaprvé, neubližovat.* Naším hlavním kritériem nejsou technické dovednosti, dokonce ani znalost kódu, ale čistě otázka, zda je committer schopen zralého úsudku. Úsudek může jednoduše znamenat prostě vědět, co přijmout a co ne. Určitá osoba sice může vyvěšovat jen malé záplaty, opravovat jen jednodušší problémy v kódu, ale pokud jsou tyto záplaty aplikovány čistě, neobsahují chyby a jsou většinou v souladu s projektovými zásadami pro psaní logovacích zpráv a formátování zdrojového kódu a pokud je těchto záplat dost na to, aby se prokázala skutečná zručnost dané osoby, může stávající committer navrhnout tuto osobu na držitele commit access. Pokud s tím budou souhlasit alespoň tři lidé a nikdo nebude proti, nabídneme dotyčné osobě commit access. Ano, nemusíme mít důkazy o tom, že je daná osoba schopna vyřešit složité problémy ve všech oblastech kódové základny, to nevadí: prokázala nám, že je schopna přinejmenším správně odhadnout vlastní schopnosti. Technické dovednosti se lze naučit (a vyučovat), ale úsudku, tomu se naučit nelze. Proto je to tak důležitá věc, kterou musí každá osoba prokázat dříve, než jí udělíte commit access.

---

[28] Upozorňuji, že commit access znamená něco trošku jiného v decentralizovaných systémech správy verzí, kde může kdokoli nastavit úložiště propojené na projekt, a sám si pak udělit commit access k tomuto úložišti. Nicméně pojem *commit access* nadále platí: „commit access“ je zkratkovitý název „práva provádět změny v kódu, které se přenesou do dalšího vydání softwaru.“ V centralizovaných systémech správy verzí to znamená mít přímý commit access; v decentralizovaných systémech pak mít implicitně své změny přetaženy do hlavní distribuce. V obou případech je myšlenka stejná; mechanismus, kterým se tyto činnosti provádějí, není zas tak důležitý.

Pokud návrh na nového committera vyvolá diskusi, netýká se zpravidla jeho technických schopností, ale spíše jeho chování na mailing listu nebo na IRC. Někdy se stává, že někdo sice prokáže své technické dovednosti a schopnost pracovat v rámci formálních směrnic projektu, ale na veřejných fórech se soustavně projevuje jako agresivní a spolupráce neschopná osoba. To je vážný problém; pokud se někdo s časem nijak nemění, neformuje se, a to ani v reakci na narážky, výhrady či tipy, pak jej určitě jako committera nepřijmeme, bez ohledu na to, jaké jsou jeho technické dovednosti. Ve skupině dobrovolníků jsou sociální dovednosti, neboli schopnost „hrát si spořádaně na pískovišti“, stejně důležité jako čistě technické schopnosti. Jelikož se vše odehrává pod správou verzí, trest za přijetí nevhodného committera nejsou ani tak případné problémy, které by mohl způsobit v kódu (revize by tyto problémy rychle odhalila), jako spíše případná nutnost této osobě někdy v budoucnu odebrat commit access—, což je vždy nepříjemná a někdy i konfrontační záležitost.

Mnoho projektů trvá na tom, aby potenciální committer prokázal určitý stupeň technické zdatnosti a vytrvalosti předložením určitého počtu složitějších záplat—, tj. tyto projekty nechtějí jen vědět, že dotyčná osoba nebude škodit či ubližovat, chtějí mít také jistotu, že bude přinášet užitek celé kódové základně. To je sice v pořádku, ale dávejte si pozor, aby se z tak z příslušnosti ke committerům nestávalo spíše členství v exkluzivním klubu. Otázka, kterou by všichni měli mít neustále na mysli, zní: „Co kódu nejvíce prospěje?“, nikoli „Snížíme přijetím této konkrétní osoby sociální statut náležející všem committerům?“ Smysl commit access netkví v posilování vlastního sebevědomí, slouží k tomu, aby byly do kódu co možná nejplynuleji zadávány správné změny. Pokud máte 100 committerů, z nichž 10 pravidelně provádí velké změny a dalších 90 jen několikrát za rok opraví překlepy a malé chyby, je to pořád lepší, než mít k dispozici jen těch prvních 10.

Odebrání commit access

O odebírání commit access je nutno nejprve uvést následující: především se snažte do této situace vůbec nedostat. V závislosti na tom, komu a proč commit access odebíráte, mohou být diskuse kolem tohoto tématu velice svárlivé. A i kdyby nebyly, budou určitě časově náročnou překážkou v další produktivní práci.

Nicméně pokud musíte k tomuto kroku přistoupit, diskuse by měla probíhat soukromě mezi stejnými lidmi, kteří by byli s to hlasovat pro to, aby dotyčné osobě byl *udělen* jakýkoli typ commit access, který momentálně mají. Této diskuse by se však neměla účastnit dotyčná osoba. To je samozřejmě v rozporu s obvykle uznávaným pravidlem naprosté otevřenosti, ale v tomto případě je to nezbytné. Zaprvé, nikdo by nemohl mluvit zcela svobodně. Zadruhé, pokud návrh spadne pod stůl, asi byste nebyli rádi, kdyby dotyčná osoba věděla, že se projednávalo odebrání jejího commit access, tím by se totiž otevíraly další otázky („Kdo byl na mé straně? Kdo stál proti mně?“), které by vedly k frakcionářství nejhrubšího zrna. Za určitých výjimečných okolností si skupina může přát, aby se dotyčná osoba dozvěděla o tom, že se zvažuje nebo se zvažovalo odebrání jejího commit access; jako jistá výstraha, tento otevřený přístup by však měla odsouhlasit celá skupina. Nikdo by z vlastní iniciativy nikdy neměl odtajnit informace z diskuse a hlasování, které ostatní považovali za tajné.

Po odebrání commit access je tato skutečnost již věcí veřejnou (viz část **Nedělejte s ničím tajnosti** dále v této kapitole), zvolte proto maximálně taktní způsob, kterým tuto zprávu ohlásíte okolnímu světu.

Částečný commit access

Některé z projektů mají commit access odstupňovaný. Mají například přispěvatele, jejichž commit access jim umožňuje volný pohyb v dokumentaci, ale nemohou nijak upravovat vlastní kód. Obvyklé oblasti pro uplatňování částečného commit access jsou dokumentace, překlady, propojení kódu s jinými programovacími jazyky, specifikační soubory pro balíčky (např. RedHat RPM atd.) a další oblasti, kde případná chyba nepovede k problémům v základním jádře projektu.

Jelikož commit access se netýká pouze potvrzování změn, ale také účasti v hlasováních (viz část **Kdo hlasuje?** v kapitole **4. Společenská a politická infrastruktura**), přirozeně se nabízí otázka: o čem tedy mohou částeční committeři hlasovat? Na tuto otázku neexistuje jedna správná odpověď; závisí na typech oblastí s částečným commit access, které se ve vašem projektu nacházejí. V projektu Subversion jsme tyto věci uspořádali co nejjednodušeji: částečný committer může hlasovat o záležitostech, které jsou omezeny čistě na jeho vlastní oblast, o žádných jiných. Důležité upozornění: máme zde zaveden mechanismus pro podávání tzv. poradních hlasů (committer v podstatě na hlasovací lístek napíše „+0“ nebo „+1 (nezávazné)“ namísto pouhého „+1“). Není důvod umlčovat lidi pouze proto, že jejich hlas není formálně závazný.

Plnoprávní committeři mohou hlasovat o čemkoli, stejně jako mají commit access kdekoli; navíc o přibírání nových committerů mohou hlasovat pouze plnoprávní committeři. V praxi je však možnost přibírat nové částečné committery obvykle delegována: kterýkoli plnoprávný committer může tzv. „sponzorovat“ (vzít pod svá křídla) nového částečného committera a stávající částečný committer v určité oblasti si často může v podstatě zvolit nové committery pro tuto oblast (to je zvláště vhodné, pokud je nutno zajistit hladký průběh překladatelské činnosti).

Váš projekt bude možná vyžadovat jiné uspořádání podle povahy práce, ale tyto obecné zásady platí pro všechny projekty. Každý committer by měl mít právo hlasovat o záležitostech, které spadají do rámce jeho commit access, nikoli o záležitostech, které do tohoto rámce nespadají; hlasování o procedurálních otázkách by mělo být standardně umožněno všem plnoprávným committerům, pokud nebude z nějakého důvodu (na základě rozhodnutí plnoprávných committerů) nutno tento elektorát rozšířit.

Co se týče přidělování částečného commit access: ve většině případů je nejlepší, když systém správy verzí nevynucuje oblast částečného commit accesu, ačkoliv je toho schopen. Důvody viz část **Autorizace** v kapitole **3. Technická infrastruktura**.

Nečinní committeři

Některé projekty automaticky odebírají commit access těm, kteří po určitou dobu (například jeden rok) neprovedou žádnou změnu. Myslím si, že tento postup je zpravidla neúčinný, ba dokonce kontraproduktivní, a to hned ze dvou důvodů.

Zaprvé, může někoho svádět k provádění sice přijatelných, ale zato zbytečných změn jen proto, aby si uchovali commit access. Zadruhé, je vlastně bezúčelný. Pokud je při udělování commit access hlavním kritériem dobrý úsudek, proč bychom si pak měli myslet, že se něčí dobrý úsudek zhoršil jen proto, že byl nějakou dobu mimo projekt? I kdyby na několik let úplně zmizel, nedíval se na kód ani nesledoval diskuse o vývoji, ve chvíli, kdy se znovu objeví, *brzy* zjistí, jak moc z projektu „vypadl“ a přizpůsobí tomu své jednání. Prve jste jeho úsudku důvěřovali, proč mu tedy nedůvěřovat nadále? Když neztrácejí platnost maturitní vysvědčení, proč by měl ztrácet platnost commit access?

Někdy může sám committer požádat o propuštění z funkce nebo o výslovné označení za nečinného v seznamu committerů (více informací o tomto seznamu viz část **Nedělejte s ničím tajnosti** níže). V těchto případech by měl projekt samozřejmě souhlasit s přáním dané osoby.

Nedělejte s ničím tajnosti

Ačkoliv jsou diskuse kolem přibírání nových committerů důvěrné, vlastní pravidla a postupy nemusí být tajné. Nejlepší by bylo tato pravidla a postupy publikovat, aby si ostatní uvědomili, že committeři nejsou žádný tajný spolek či tribunál, uzavřený všem běžným smrtelníkům, ale že se do jejich skupiny může dostat prostě každý, kdo zasílá dobré záplaty a ví, jak se správně v komunitě chovat. V projektu Subversion jsme tyto informace umístili přímo do dokumentu se směrnicemi pro vývojáře, protože ti, kdo se budou pravděpodobně nejvíce zajímat o to, jak se commit access uděluje, jsou lidé, co mají v plánu přispívat do projektu kódem.

Kromě těchto postupů také zveřejněte aktuální *seznam* committerů. Tradiční místo pro tyto dokumenty je soubor nazvaný `MAINTAINERS` nebo `COMMITTERS` v nejvyšší úrovni stromu zdrojového kódu projektu. Měl by nejprve uvádět kompletní seznam committerů, po něm různé oblasti s částečným commit access a pak členy každé oblasti. Každá osoba v seznamu by měla být uvedena svým jménem a emailovou adresou; pokud si to dotyčná osoba přeje, tato adresa by mohla být kódována, aby se zabránilo spamu (viz část **Skrývání adres v archivech** v kapitole **3. Technická infrastruktura**).

Vzhledem k tomu, že rozdíl mezi plnoprávným a částečným commit access je zřejmý a dostatečně definovaný, seznam by měl na toto členění také dbát. Kromě toho by seznam neměl udávat neformální rozdíly, které v každém projektu přirozeně vznikají, jako například, kdo je zvláště vlivný a jak. Jedná se o veřejný záznam, nikoli o seznam zasloužilých členů. Committery uvádějte buď v abecedním pořádku, nebo v pořadí, v jakém do projektu přistoupili.

Uvádění tvůrců a spoluautorů (Credit)

Credit je základní platidlo ve světě svobodného softwaru. Ať už lidé říkají o svých motivech k účasti na projektu cokoli, neznám žádného vývojáře, který by rád dělal celou práci anonymně nebo pod cizím jménem. K tomu mají všichni hmatatelné důvody: osobní reputace v projektu zhruba ovlivňuje to, jaký vliv kdo má; účast na open source projektu však může mít i nepřímou finanční hodnotu, někteří zaměstnavatelé se po ní dnes ohlížejí v životopisech zájemců o práci. Kromě toho existují i nehmotné důvody, ty jsou snad ještě pádnější: lidé prostě chtějí být oceňováni a instinktivně vyhledávají známky toho, že je jejich práce ostatními uznávána. Příslib uvedení jejich jména mezi tvůrci softwaru je jednou z nejlepších motivací, které projekt nabízí. Když jsou vděčně přijímány i drobné příspěvky, lidé se k projektu rádi vracejí a přispívají víc a víc.

Jedna z nejdůležitějších vlastností kolektivně vyvíjeného softwaru (viz kapitola **3. Technická infrastruktura**) je, že pečlivě zaznamenává a eviduje, kdo, co a kdy udělal. Tyto stávající mechanismy používejte v největší možné míře, abyste zajistili, že je spoluautorství zaznamenáno přesně, konkrétně uvádějte povahu příspěvku. Ve zprávě protokolu nestačí jen napsat: „Děkujeme J. Novákovi <jnovak@example.com>“, když můžete místo toho napsat: „Děkujeme J. Novákovi <jnovak@example.com> za bug report a reprodukční návod“.

V projektu Subversion máme zaveden neformální, ale důsledný předpis pro uvádění ohlašovatele chyby buď v políčku „issue“, je-li používáno, nebo případně ve zprávě protokolu příslušného commitu, který chybu opravil. Rychlý přehled záznamů o potvrzení provedených změn (commit log) až k číslu 14525 ukazuje, že cca 10 % těchto potvrzení uvádí něčí jméno a e-mailovou adresu, obvykle osoby, která nahlásila nebo analyzovala chybu, jež byla tímto potvrzením opravena. Upozorňuji, že tato osoba je někdo jiný než vývojář, který v praxi toto potvrzení změny provedl a jehož jméno je již zaznamenáno automaticky systémem správy verzí. Z více než 80 plnoprávných a částečných committerů, které dnes Subversion má, bylo 55 nejprve uvedeno v záznamu o potvrzení změn (většinou několikrát), než se z nich stali committeři. To samozřejmě nedokazuje, že je-li někdo uveden jako spoluautor, bude se i nadále podílet na projektu, ale vytváří to přinejmenším atmosféru, v níž lidé vědí, že jejich příspěvky budou dozajista oceňovány.

Je důležité rozlišovat mezi běžným uznáním a zvláštním poděkováním. Při diskusích o určité části kódu nebo o něčích jiných příspěvcích je správné vyjádřit uznání jejich práci. Například můžete zmínit, že „Danielovy poslední změny v delta kódu znamenají, že odteď již můžeme implementovat funkční vlastnost X“, to zároveň ostatním pomáhá určit, o kterých změnách hovoříte, a uznat Danielovu práci. Na druhou stranu samostatný děkovný vzkaz Danielovi za změny v delta kódu nemá žádný bezprostřední praktický význam. Neposkytuje žádnou informaci, neboť systém správy verzí a ostatní mechanismy již zaznamenaly fakt, že Daniel tyto změny provedl. Děkovat všem za vše by bylo rušivé a zcela zbytečné, vyjádření díků je účinné především tím, jak moc vyniká oproti běžnému pozadí různých oblíbených a ohraných komentářů. Tím samozřejmě nechci říci, že byste neměli lidem vůbec děkovat. Jenom to musíte dělat způsobem, které nepovede k inflaci vděku. Následující zásady vám určitě pomohou:

- Čím krátkodobější je dané fórum, tím svobodněji se zde můžete cítit při vyjadřování díků. Například, poděkovat někomu mezi řečí během IRC konverzace je v pořádku, stejně jako podobné poděkování uvedené někde stranou v e-mailu věnovaném převážně jiným otázkám. Ale určitě neposílejte e-mail jen za účelem poděkování, pokud se ovšem nejedná o opravdu neobyčejný počin. Taktéž nenarušujte webové stránky projektu vyjádřením vděčnosti. Jakmile s tím jednou začnete, už nebudete vědět kdy a kde přestat. A *nikdy* nevkládejte díky do komentářů v kódu; odvádělo by to od hlavního cíle komentářů, tj. napomoci čtenáři v pochopení kódu.
- Čím méně je někdo zapojen do projektu, tím vhodnější je mu poděkovat za něco, co udělal. Může to vypadat nerozumně, ale plně to souhlasí s přístupem, že se díky mají vyjádřit, kdykoli někdo provede více, než jste původně čekali. Proto by neustálé děkování pravidelným přispěvatelům za práci, kterou dělají běžně, mohlo být vykládáno tak, že od nich očekáváte méně než oni sami od sebe. A vy přeci chcete dosáhnout úplně opačného efektu!
- Existují ojedinělé výjimky z tohoto pravidla. Je přijatelné poděkovat někomu za plnění jeho očekávané role v projektu, pokud tato role v sobě skrývá krátkodobé občasné intenzivní úsilí. Klasickým příkladem je release manager, jehož práce přechází na vyšší obrátky vždy v době kolem vydání, ale jinak setrvává v nečinnosti (tj. v nečinnosti jako release manager, v každém případě—může být aktivním vývojářem, ale to už je jiná záležitost).
- Podobně jako při kritizování a uvádění jmen spoluautorů i vyjádření vděčnosti by mělo být konkrétní. Neděkujte lidem jen za to, že jsou skvělí, byť by třeba byli. Poděkujte jim za něco mimořádného, co udělali, a jako třešničku na dortu jim ještě přesně řekněte, proč je to tak mimořádné.

Obecně vždy dochází k napětí mezi tím, jak zajistit, aby byly řádně uznány jednotlivé příspěvky účastníků projektu a přitom zachovat projekt jako skupinové úsilí a nikoli jako přehlídku individuální slávy. Stačí, když si budete tohoto napětí neustále vědomi a budete podporovat skupinové úsilí, věci se vám pak nevymknou z kontroly.

Odnože (Forks)

V části **Schopnost vytvářet odnože** v kapitole **4. Společenská a politická infrastruktura** jsme viděli, jak schopnost vytvářet odnože ovlivňuje způsob řízení projektu. Ale co se děje, když se odnož skutečně objeví? Jak ji zvládat a jaké dopady od ní může očekávat? Nebo naopak, kdy je zapotřebí odnož *iniciovat*?

Odpovědi na tyto otázky závisí na typu konkrétní odnože. Některé odnože jsou způsobeny pokojnými, ale nesmiřitelnými neshodami v otázce na další směřování projektu; většina odnoží je zřejmě způsobena jak technickými neshodami, tak osobními konflikty. Samozřejmě, že není vždy možné tyto důvody přesně rozlišit, i technické argumenty mohou zahrnovat ryze osobní výtky. Co mají všechny odnože společné, je, že se jedna skupina vývojářů (někdy dokonce jeden jediný vývojář) rozhodne, že náklady na spolupráci s některými či všemi ostatními už převažují nad výhodami.

Jakmile se projekt rozštěpí, nelze najít definitivní odpověď na otázku, která odnož je „ta pravá“ nebo „původní“. Bude se běžně hovořit o odnoži F (fork) pocházející z projektu P, jakoby P pokračovalo dál beze změn, zatímco F se odklonilo do zcela nového teritoria, což však vypovídá spíše o osobním názoru pozorovatele celého procesu. V podstatě se jedná o to, jak celou věc vnímáte: když bude dostatečně velké procento pozorovatelů souhlasit, výrok se stane pravdivým. V tomto případě jednoduše od počátku neexistuje žádná objektivní pravda, kterou bychom jen nedokázali zprvu dokonale vnímat. Jsou to spíše vjemy, které *jsou* onou objektivní pravdou, protože projekt—nebo odnož—jsou jednoduše entity, které existují pouze v lidských myslích.

Pokud osoby, které vznik odnože iniciují, mají dojem, že spouštějí zcela novou větev hlavního projektu, pak je otázka vnímání snadno a okamžitě vyřešena. Všichni, vývojáři i uživatelé, se budou k odnoži chovat jako k novému projektu, s novým jménem (možná vycházejícím ze jména původního, leč snadno odlišitelným), samostatnou webovou stránkou a vlastní filosofií či cíli. Věci se však začnou komplikovat, když mají obě strany dojem, že jsou legitimními strážci původní projektu, a mají tak právo nadále používat i jeho původní název. Pokud existuje organizace s ochrannými právy na značku/název nebo právní mocí nad danou oblastí či webovými stránkami, celý spor se vyřeší z pozice právní autority: tato organizace rozhodne, kdo (která skupina) je původním projektem a kdo (která skupina) je odnož, protože tato organizace drží v této PR (public relations) válce všechny karty v ruce. Tyto záležitosti pochopitelně málokdy zajdou takto daleko: každý už totiž ví, co je silová dynamika, každý se vyhne boji, jehož výsledek je předem jasný, a rovnou přeskočí na konec.

Naštěstí ve většině případů se téměř nepochybuje o tom, která strana je projektem a která odnoží, protože odnož je v podstatě odhlasováním důvěry. Pokud nadpoloviční většina vývojářů podpoří některý ze směrů, který odnož navrhne, není zpravidla zapotřebí odnož vytvářet—projekt se může jednoduše tímto směrem ubírat sám od sebe, není-li řízen diktátorsky nějakým zvláště neústupným tyranem. Naopak, pokud tento směr podpoří méně než polovina vývojářů, odnož je evidentní menšinovou vzpourou a slušnost i selský rozum kážou, aby tato odnož sama sebe považovala za odnož a nikoli za hlavní proud.

Zvládání odnoží

Pokud ve vašem projektu někdo hrozí vytvořením odnože, zůstaňte klidní a pamatujte na své dlouhodobé cíle. Samotná *existence* odnože projekt nijak nepoškozuje; je to spíše jen ztráta vývojářů a uživatelů. Váš skutečný cíl tedy není potlačení odnože, ale minimalizace těchto škodlivých účinků. Můžete zuřit, můžete mít pocit, že odnož jedná neoprávněně a samozvaně, ale pokud tyto emoce vyjádříte veřejně, nedosáhnete ničeho, jen si proti sobě popudíte dosud nerozhodnuté vývojáře. Naopak, nenuťte lidi, aby si volili jediné řešení a zkuste s odnoží spolupracovat, jak jen to bude možné. Pro začátek neodnímejte commit access ve vašem projektu někomu jen proto, že se rozhodl pracovat na odnoži. Práce na odnoži neznamená, že dotyčná osoba rázem ztratila způsobilost pracovat i na původním projektu; committeři před touto odlukou by měli zůstat committery i po ní. Kromě toho byste měli vyjádřit své přání uchovat co největší kompatibilitu s odnoží a vyjádřete naději, že vývojáři budou mezi oběma projekty přenášet změny, kdykoli to bude vhodné. Pokud máte administrativní přístup k projektovým serverům, veřejně

nabídněte všem odštěpeným (forkers) infrastrukturní pomoc v době náběhu. Nabídněte jim například kompletní hloubkově historickou kopii úložiště správy verzí, pokud si ji nemohou obstarat jinak, aby nemuseli začínat bez historických dat (to nemusí být nutné v závislosti na konkrétním systému správy verzí. Zeptejte se jich, zda by nepotřebovali něco dalšího, a pokud můžete, poskytněte jim to. Všemožně se snažte ukázat, že jim nechcete stát v cestě a že si skutečně přejete, aby o úspěchu či neúspěchu odnože rozhodovaly jen jejich schopnosti a nic jiného.

Důvod, proč se takto chovat—a proč to dělat veřejně—, netkví v tom, že byste chtěli odnoži skutečně pomoci, ale abyste přesvědčili vývojáře, že vaše strana je sázka na jistotu, snažte se působit nepomstychtivě. Ve válce má někdy smysl (strategický smysl, ne-li lidský smysl) přinutit lidi, aby volili mezi stranami, ale ve svobodném softwaru to skoro nikdy smysl nemá. Ve skutečnosti někteří vývojáři po vytvoření odnože nepokrytě pracují na obou projektech a snaží se uchovávat jejich vzájemnou kompatibilitu. Tito vývojáři pomáhají udržovat komunikaci po vytvoření odnože. Umožňují vašemu projektu využívat zajímavé nové funkční vlastnosti odnože (ano, odnož skutečně může vlastnit věci, které chcete i vy) a rovněž zvyšují pravděpodobnost pozdějšího opětného splynutí obou projektů.

Někdy je odnož natolik úspěšná, že ačkoliv byla na začátku i vlastními iniciátory vnímána jen jako odnož, stane se z ní verze, kterou všichni preferují, až nakonec díky široké poptávce nahradí původní projekt. Slavným příkladem je odnož GCC/EGCS. *GNU Compiler Collection* (GCC, dříve *GNU C Compiler*) je nejpopulárnější open source kompilátor původního kódu a také jeden z nejlépe přenositelných kompilátorů na světě. Kvůli neshodám mezi oficiálními správci GCC a Cygnus Software,[29] jedna z nejaktivnějších skupin vývojářů GCC, Cygnus, vytvořila odnož GCC, nazvanou EGCS. Odnož byla záměrně nesoupeřivá: vývojáři EGCS se nikdy nesnažili vykreslovat svou verzi GCC jako novou oficiální verzi. Naopak, soustředili se na maximální vylepšení EGCS, zapracovávali záplaty mnohem rychleji než oficiální správci GCC. EGCS nabývalo na popularitě, až se několik významných distributorů operačních systémů rozhodlo dodávat EGCS jako svůj standardní kompilátor namísto GCC. V tuto chvíli si správci GCC přiznali, že lpět na názvu GCC, zatímco všichni přešli na odnož EGCS, by všechny zúčastněné jen zatěžovalo zbytečnou změnou názvu, aniž by jakkoli zabránilo přecházení na odnož EGCS. Proto GCC přejala kódovou základnu EGCS a od té doby už je zase jen jedno GCC, ale výrazně zdokonalené díky odnoži.

Tento příklad ukazuje, proč nemůžete vždy považovat odnož za veskrze špatnou věc. Odnož může být v určitou chvíli bolestivá a nevítaná záležitost, ale nikdy nemůžete vědět, zda náhodou neuspěje. Proto ji musíte s celým projektovým týmem pečlivě sledovat a být připraveni nejen na případné přejímání funkčních vlastností a kódu, ale v nejkrajnějším případě i na připojení k odnoži, pokud získá na svou stranu obecné povědomí o projektu. Samozřejmě, že v mnoha případech budete schopni předpovědět úspěšnost odnože, podle toho, kdo se k ní připojuje. Pokud odnož spustí největší stěžovatel v projektu a přidá se k němu hrstka otrávených vývojářů, kteří se stejně nikdy nechovali konstruktivně, vytvořením odnože za vás v podstatě vyřešili celý problém, a pravděpodobně se nemusíte bát, že by odnož

[29] Dnes součást RedHat (**http://www.redhat.com/**).

původnímu projektu vzala vítr z plachet. Když se však začnou odnož podporovat vlivní a uznávaní vývojáři, měli byste se sami sebe ptát, proč. Možná byl projekt příliš restriktivní a nejlepším řešením by tedy bylo převzít do hlavní linie projektu některé či všechny kroky navrhované odnoží—, tj. vyhnout se problému s odnoží tím, že se jí sami stanete.

Iniciování odnože

Následující rada předpokládá, že vytvoříte odnož jako východisko z nouze. Než odnož skutečně vytvoříte, musíte vyčerpat všechny ostatní možnosti. Vytvoření odnože s sebou téměř vždy nese ztrátu vývojářů s pouze nejasným příslibem získání nových vývojářů někdy v budoucnu. Odnož rovněž znamená zahájení soutěže o pozornost uživatelů: každý, kdo si chce stáhnout daný software, se sám sebe ptá: „Hm, chci tenhle, nebo spíš tenhle?“ Ať je váš projekt tím či oním softwarem, celá situace je zmatečná, protože klade otázku, která nebyla dříve vůbec položena. Někteří tvrdí, že odnože jsou zdravé pro softwarový ekosystém jako celek, uvádějí klasický přírodní výběr: přežije ten nejsilnější, což v důsledku znamená, že všichni budou mít lepší software. To může být z hlediska ekosystému pravda, ale z pohledu jednotlivého projektu to pravda není. Většina odnoží neuspěje a většině projektů se nelíbí, že jsou rozštěpovány.

Logickým důsledkem je, že byste neměli používat hrozbu vzniku odnože jako extremní diskusní postup—„Buď budete dělat věci po mém, nebo vytvořím odnož!“— každý totiž ví, že odnož, která nezaujme a nepřetáhne vývojáře z původního projektu, nemá příliš šancí na dlouhý život. Všichni pozorovatelé—, nejen vývojáři, ale také uživatelé a dodavatelé operačních systémů—, sami posoudí, ke které straně se přiklonit. Měli byste proto dávat najevo svou silnou neochotu vytvářet odnože, protože, když odnož nakonec skutečně vytvoříte, můžete věrohodně tvrdit, že vám již nic jiného nezbývalo.

Při posuzování úspěšnosti vaší odnože nesmíte vypustit ze zřetele *žádný* faktor. Například pokud má mnoho vývojářů v projektu stejného zaměstnavatele, pak i pokud jsou otrávení a soukromě by podpořili odnož, pravděpodobně své pocity a přesvědčení nevysloví nahlas, pokud vědí, že jejich zaměstnavatel bude proti. Mnoho programátorů svobodného softwaru se často domnívá, že držení bezplatné licence na kód znamená, že žádná firma nemůže dominovat vývoji. Je skutečně pravda, že licence je, v konečném smyslu, garancí svobody—, pokud chtějí ostatní opravdu rozštěpit projekt a mají na to dostatek zdrojů, mohou tak učinit. Ale v praxi jsou vývojářské týmy některých projektů z větší části financovány jediným subjektem a nemá smysl předstírat, že podpora tohoto subjektu je jim lhostejná. Pokud je tento subjekt proti odnoži, jeho vývojáři se pravděpodobně na odnoži nebudou podílet, byť po tom soukromě touží.

Pokud stále trváte na tom, že musíte vytvořit odnož, nejprve si soukromě sešikujte všechny podporovatele, teprve poté vlídně oznamte vytvoření odnože. I když jste na aktuální správce naštvaní nebo jste z nich zklamáni, nezmiňujte to ve svých zprávách. Zcela věcně oznamte, co vás vedlo k rozhodnutí vytvořit odnož a že projektu, od nějž se odštěpujete, nepřejete nic zlého. Za předpokladu, že nově vzniklý projekt pokládáte za odnož (na rozdíl od případu, kdy se v nouzové situaci snažíte zachraňo-

vat původní projekt), zdůrazněte, že vytváříte odnož od kódu nikoli od názvu a zvolte si název, který nebude v konfliktu s názvem projektu. Můžete použít název, který obsahuje nebo odkazuje na původní jméno projektu, pokud tento nový název nezavdává příčinu k záměně identity. Samozřejmě, že je zcela v pořádku, když na webových stránkách odnože jasně vysvětlíte, že vychází z původního programu a že dokonce doufáte, že jej nahradí. Jenom neznepříjemňujte uživatelům život a nenuťte je rozmotávat spory o identitu obou projektů.

Nakonec můžete svůj projekt pořádně spustit, vykročte pravou nohou a všem committerům z původního projektu udělte automaticky commit access k odnoži, a to i těm, kteří otevřeně nesouhlasili s potřebou vytvářet odnož. I kdyby tento commit access nikdy nevyužili, váš signál je jasně čitelný: mezi námi sice existují neshody, ale nejsme nepřátelé, vítáme příspěvky do kódu z jakéhokoli kompetentního zdroje.

9. Licence, autorská práva a patenty

9. Licence, autorská práva a patenty — 257

Terminologie — 257
Aspekty licence — 260
GPL a kompatibilita licence — 262

Volba licence — 263
MIT / X Window System License — **263**
GNU General Public License — **264**
Je GPL svobodná licence, či nikoli? — **265**
A co licence BSD? — **266**

Převod a vlastnictví autorských práv — 267
Nedělat nic — **267**
Licenční smlouvu s přispěvatelem (CLA) — **268**
Převod autorských práv — **268**

Systém dvojitých licencí — 269
Patenty — 270
Další zdroje — 274

9. Licence, autorská práva a patenty

Licence, kterou si zvolíte, pravděpodobně nebude mít významnější dopad na užívání vašeho projektu, pokud bude tato licence open source. Uživatelé si zpravidla vybírají software na základě jeho kvality a funkčních vlastností, nikoli podle podrobností obsažených v příslušné licenci. Přesto byste měli alespoň v obrysech rozumět celé problematice licencování svobodného softwaru, jednak abyste zajistili, že bude projektová licence odpovídat projektovým cílům, jednak abyste mohli diskutovat s ostatními o rozhodnutích týkajících se licence. Upozorňuji vás však předem, že nejsem právník, a žádnou z informací obsažených v této kapitole tak nelze chápat jako oficiální právní radu. Pro skutečnou právní radu si musíte najmout právníka nebo jím sám být.

Terminologie

V jakékoli diskusi o open source licencování se jako první problém projeví skutečnost, že pro stejnou věc máme pravděpodobně mnoho různých slov: *svobodný software, open source, FOSS, F/OSS* a *FLOSS*. Začněme tedy drobným uspořádáním terminologie.

svobodný software (free software)
Software, který lze volně sdílet a upravovat, včetně zásahů do zdrojového kódu. Termín vytvořil Richard Stallman, který jej kodifikoval v licenci GNU General Public License (všeobecná veřejná licence GNU, dále jen „GPL“) a založil nadaci Free Software Foundation (**http://www.fsf.org/**), která tento pojem propagovala.

Ačkoliv termín „svobodný software“ pokrývá téměř stejnou oblast softwaru jako „open source“, FSF (mimo jiných) preferuje termín „svobodný software“, protože se jím zdůrazňuje myšlenka svobody a koncepce volně šiřitelného softwaru jako společenského (nikoli nutně technického) hnutí. FSF přiznává, že termín je dvojznačný—anglický termín „free“ totiž může znamenat „bezplatný, beznákladový“ i „svobodný“, jako v anglickém slově „freedom“ (svoboda)—ale má přesto dojem, že je to termín nejlepší a že ostatní alternativy v angličtině jsou taktéž dvojznačné. (V této knize je slovo „free“ používáno ve smyslu „freedom“ (svobodný, resp. svoboda, nikoli „bezplatný, beznákladový“.)

open source software
Jiný název pro svobodný software. Tento jiný název však odráží důležitý filosofický rozdíl: termín „open source“ vytvořila organizace Open Source Initiative (**http://www.opensource.org/**) jako vědomou alternativu k termínu „free software“ (svobodný software), aby byl tento software stravitelnější pro velké společnosti, prezentovala jej spíše jako vývojovou metodiku než politické hnutí. Také tím možná chtěli překonat další stigma, že totiž co je „bezplatné“, musí být horší kvality.

Zatímco každá licence, která je svobodná, je také open source a naopak (s několika nevýznamnými výjimkami), lidé si rádi zvolí jeden termín a toho se pak drží. Obecně platí, že ti, kteří preferují termín „free software“ (svobodný software), zaujímají k této záležitosti spíše filosofický či morální postoj, zatímco ti, kteří preferují termín „open source“, buď celou záležitost nevnímají jako otázku svobody, nebo nechtějí dávat najevo, že ji jako otázku svobody vnímají. Podrobnou historii tohoto rozkolu naleznete v části **„Free“ versus „open source“** v kapitole **1. Úvod**.

Free Software Foundation nabízí skvělý— zcela neobjektivní, zato vycizelovaný a jasný—výklad těchto dvou termínů na adrese **http://www.fsf.org/licensing/essays/free-software-for-freedom.html.** Výklad Open Source Initiative je (nebo byl v roce 2002) rozepsán na dvou stránkách: **http://web.archive.org/web/20021204155022/, http://www.opensource.org/advocacy/case_for_hackers.php#marketing** a **http://web.archive.org/web/20021204155022/, http://www.opensource.org/advocacy/free-notfree.php.**

FOSS, F/OSS, FLOSS

Tam, kde jsou dvě entity, budou brzy tři; a přesně toto se děje s termíny pro svobodný software. Akademický svět, pravděpodobně v touze po přesnosti a všeobsažnosti, byť na úkor elegance, se dohodl na užívání zkratky FOSS, případně F/OSS, tj. „Free / Open Source Software“. Další prudce se rozvíjející alternativou je FLOSS, což znamená „Free / Libre Open Source Software“ (výraz *libre* je mnoha jazykům blízký, a navíc není poznamenán dvojznačností anglického termínu "free"; bližší informace viz **http://en.wikipedia.org/wiki/FLOSS**).

Všechny tyto termíny označují v podstatě stejnou věc: software, který může upravovat a dále předávat kdokoli, někdy —nikoli vždy—s tím požadavkem, aby odvozená díla byla volně šiřitelná za stejných podmínek.

DFSG-compliant

Odpovídající zásadám Debian Free Software Guidelines (**http://www.debian.org/social_contract#guidelines**). Jedná se o velmi rozšířený a používaný test, zda je určitá licence skutečně open source (free, *libre* atd.). Posláním projektu Debian je zachovat zcela svobodný operační systém, tj. takový, po jehož instalaci nebude dotyčný uživatel nikdy na pochybách, zda má právo celý tento systém upravovat a šířit dál. Zásady Debian Free Software Guidelines jsou požadavky, které musí licence softwarového balíčku splňovat, aby mohla být zahrnuta do Debianu. Jelikož Debian Project dost dlouho dumal nad tím, jak takový test vytvořit, zásady, s nimiž přišli, se osvědčily jako velice pevné (viz **http://en.wikipedia.org/wiki/DFSG**), a pokud je mi známo, nebyla vůči nim vznesena žádná vážnější námitka ze strany Free Software Foundation ani Open Source Initiative. Pokud víte, že daná licence je DFSG-compliant, můžete si být jisti, že garantuje všechny důležité svobody (jako například možnost vytvářet odnože, dokonce navzdory přání původního autora), které jsou nezbytné k zachování dynamiky open source projektu. Všechny licence, o nichž pojednává tato kapitola, jsou DFSG-compliant.

OSI-approved
Schválen organizací Open Source Initiative. Další široce používaný test, zjišťující, zda určitá licence umožňuje všechny nezbytné svobody. OSI definice open source softwaru je založena na zásadách Debian Free Software Guidelines a licence, která splňuje jednu z těchto definic, téměř vždy splňuje i definici druhou. Během let se samozřejmě vyskytlo několik výjimek, jednalo se však pouze o ojedinělé licence, bez významu pro tuto publikaci. Na rozdíl od projektu Debian eviduje organizace OSI seznam všech licencí, které kdy schválila, viz **http://www.opensource.org/licenses/**, to znamená, že být „OSI-approved" je zcela jednoznačné tvrzení: příslušná licence buď je, či není na tomto seznamu.

Free Software Foundation rovněž eviduje seznam licencí na adrese **http://www.fsf.org/licensing/licenses/license-list.html.** FSF třídí licence nejen podle toho, zda jsou svobodné, ale také zda jsou kompatibilní s GNU General Public Licence. Kompatibilita s GPL je důležité téma, o němž budeme pojednávat v sekci GPL a kompatibilita licence dále v této kapitole.

proprietární, closed-source
Opozitum k termínu „svobodný" (free) nebo „open source". Označuje software distribuovaný v rámci podmínek tradiční placené licence, kde uživatelé platí za každou kopii, nebo v rámci jiných podmínek, které jsou dostatečně restriktivní, aby zabránily dynamice open source softwaru v běžném chodu. I software distribuovaný bezplatně může být proprietární, pokud jeho licence neumožňuje volné rozšiřování a úpravy.

V obecné rovině jsou termíny „proprietární" a „closed-source" synonyma. Nicméně termín „closed-source" navíc znamená, že zdrojový kód nelze ani sledovat. Jelikož zdrojový kód nelze u většiny proprietárních softwarů vidět, je toto rozlišení zpravidla bez jakéhokoli významu. Příležitostně však někdo vydá proprietární software pod licencí, která jiným umožňuje zobrazit si zdrojový kód. Někdy se bohužel takovémuto softwaru mylně říká „open source" nebo „nearly open source" (téměř open source) apod., což je ovšem zavádějící. *Viditelnost* zdrojového kódu není zásadním problémem; důležité je, co s tímto zdrojovým kódem můžete dělat. Proto je rozdíl mezi proprietárním a close-source softwarem v podstatě bezpředmětný a oba termíny je možno považovat za synonyma.

Někdy se pro termín „proprietární" používá jako synonymum pro výraz *komerční*, v přísném slova smyslu se však nejedná o stejnou věc. Svobodný software může být komerční. Koneckonců svobodný software lze prodávat, pokud není kupujícímu zapovězeno šířit jeho kopie. Svobodný software lze komerčně využívat i jinými způsoby, například prodejem podpory, služeb a certifikátů. Dnes existují firmy s kapitálem mnoha milionů dolarů, které jsou vystavěny na svobodném softwaru, tento software totiž není apriori antikomerční nebo antifiremní. Na druhou stranu je ze své podstaty antiproprietární, a to je klíčový rozdíl oproti tradičním licenčním modelům založeným na jednotlivých kopiích.

volné dílo (public domain)
Bez držitele autorských práv, tzn. neexistuje nikdo, kdo by měl právo omezovat kopírování příslušného díla. Být součástí volného díla neznamená, že příslušné dílo nemá žádného autora. Všechno má svého autora; i když se autor či autoři díla rozhodnou postoupit své dílo do veřejného majetku, nic se tím nemění na skutečnosti, že dílo napsali (či jinak vytvořili).

Pokud je určité dílo součástí veřejného majetku, materiál pocházející z tohoto díla lze zakomponovat do díla chráněného autorským zákonem a *kopie* tohoto materiálu je pak kryta stejnými autorskými právy jako celé dílo. Tím však není nijak dotčena dostupnost původního díla, které nadále zůstává volným dílem. Publikování určitého díla jako volné dílo je tak, technicky vzato, jedním ze způsobů, jak toto dílo učinit „svobodným" dle zásad většiny organizací pro certifikaci svobodného softwaru. Existují však dobré důvody, proč používat licenci namísto pouhého postoupení do volného díla: i u svobodného softwaru mohou být určitá omezení užitečná, a to nejen pro držitele autorských práv, ale i pro příjemce dotyčného díla, jak bude objasněno v následující kapitole.

copyleft
Licence využívající zákona o autorském právu tak, aby byl dosažený výsledek opačný než u tradičního „copyrightu". Podle toho, koho se zeptáte, označuje termín „copyleft" buď licence, které připouštějí svobody, jež zde probíráme, nebo – v užším slova smyslu – tyto svobody nejen připouštějí, ale dokonce je *vymáhají*, požadují totiž, aby tyto svobody doprovázely příslušné dílo. Free Software Foundation užívá výhradně druhou definici; jinde je to různé: mnoho lidí tento termín používá stejně jako FSF, jiní —včetně těch, kdo píší do mainstreamových médií—používají spíše první definici. Není jisté, zda jsou si všichni uživatelé tohoto termínu vědomi, že je nutno rozlišovat.

Zářný příklad užší a přísnější definice je GNU General Public Licence, která požaduje, aby každé odvozené dílo bylo licencováno dle GPL, bližší informace viz část **GPL a kompatibilita licence** níže v této kapitole.

Aspekty licence

Ačkoliv je dnes k dispozici mnoho různých licencí svobodného softwaru, v důležitých otázkách všechny tyto licence tvrdí to samé: každý může upravovat kód, každý jej může šířit v původní i upravené formě a držitelé autorských práv a autoři neposkytují žádnou záruku (odmítnutí odpovědnosti je zvláště důležité vzhledem k tomu, že uživatelé mohou používat upravené verze, aniž by to věděli). Rozdíly mezi licencemi se ztenčily na několik opakujících se aspektů:

kompatibilita s proprietárními licencemi
Některé svobodné licence umožňují, aby byl dotyčný kód použit v proprietárních programech. Tím nejsou dotčeny licenční podmínky proprietárního programu: je nadále proprietární, může však obsahovat kód z neproprietárního zdroje. Apache License, X Consortium License,

BSD-style license a MIT-style license – jsou všechno příklady licencí kompatibilních s proprietárním softwarem.

kompatibilita s ostatními svobodnými licencemi
Většina svobodných licencí je kompatibilní s ostatními, to znamená, že kód krytý jednou licencí lze kombinovat s kódem z jiné licence a výslednou kombinaci pak distribuovat v rámci jedné z těchto licencí, aniž by tím byly porušeny podmínky licence druhé. Nejvýznamnější výjimku v tomto ohledu představuje GNU General Public Licence, která vyžaduje, aby kterékoli dílo používající kód GPL bylo samo distribuováno pod licencí GPL a bez dalších omezení mimo požadavky GPL. GPL je kompatibilní s některými svobodnými licencemi, s některými však není. Tato problematika je podrobněji probírána části **GPL a kompatibilita licence** dále v této kapitole.

povinnost uvádění autorů
Některé svobodné licence požadují, aby jakékoli využití dotyčného kódu bylo doplněno oznámením, jehož umístění a způsob zobrazení jsou zpravidla stanoveny a které uvádí autory nebo držitele autorských práv ke kódu. Tyto licence jsou často nadále kompatibilní s proprietárními: nutně nevyžadují, aby odvozené dílo bylo svobodné, požadují jasné uvedení autorů svobodného kódu.

ochrana obchodní známky
Varianta povinnosti uvádět autory. Licence chránící obchodní známky stanovují, že název původního softwaru (nebo držitelů autorských práv k tomuto softwaru či jejich institucí atd.) *nesmějí* být používány v odvozených dílech bez předchozího písemného souhlasu. Ačkoliv povinnost uvádění autorů trvá na uvádění určitého jména/názvu a ochrana obchodní známky trvá na tom, aby nebyla používána, oba přístupy jsou vyjádřením stejného přání: aby reputace původního kódu byla chráněna a přenášena, nikoli oslabena asociacemi.

ochrana „umělecké integrity“
Některé licence (např. Artistic License, používaná pro nejpopulárnější implementaci programovacího jazyka Perl a licence Donalda Knutha TeX) vyžadují, aby úpravy a další šíření byly prováděny způsobem, který jasně rozlišuje mezi neupravenou původní verzí kódu a všemi případnými úpravami. Umožňují v zásadě stejné svobody jako jiné svobodné licence, ale kladou určité požadavky, které usnadňují ověřování integrity původního kódu. Tyto licence se mimo specifické programy, pro něž byly vytvořeny, nesetkaly s velkým úspěchem, v této kapitole jim nebude věnována pozornost; uvádím je zde pouze pro úplnost.

Většina těchto ustanovení se vzájemně nevylučuje, některé licence jich obsahují několik. Všechna v podstatě kladou na příjemce určité požadavky výměnou za jeho právo užívat a/nebo dále šířit kód. Například, některé projekty chtějí, aby jejich názvy a reputace byly přenášeny společně s kódem. Tento požadavek si zaslouží uložení dodatečných povinností, které jsou stanoveny v klauzuli o uvádění autorů či ochranné známce; podle náročnosti tohoto požadavku si pak uživatelé raději zvolí balíček s méně přísnou licencí.

GPL a kompatibilita licence

Rozhodně nejmarkantnější rozdíl v licencích je mezi licencemi, které nejsou kompatibilní s proprietárními, a těmi, které s proprietárními kompatibilní jsou, tj. mezi GNU General Public License (GNU GLP) a všemi ostatními. Jelikož prvotním cílem autorů GPL je propagace svobodného softwaru, záměrně vytvořili licenci, která znemožňuje „rozpuštění" kódu krytého GPL v proprietárních programech. Konkrétně, mezi požadavky GPL (celý seznam viz **http://www.fsf.org/licensing/licenses/gpl.html**) jsou zvláště důležité tyto dva:

1. Veškerá odvozená díla—tj. díla obsahující nezanedbatelnou část kódu krytého GPL—musí být distribuována podle podmínek GPL.
2. Na další šíření původního či odvozeného díla nesmí být uvalena žádná dodatečná omezení. (Budeme-li přesně citovat: „Nesmíte zavést žádná další omezení na výkon práv zaručených či stvrzených touto licencí.")

Pomocí těchto podmínek se GPL daří šířit svobodu. Je-li program autorsky chráněn licencí GPL, podmínky jeho dalšího šíření jsou *„infekční"*—přenášejí se na vše, do čeho je daný kód začleněn, čímž je účinně zajištěno, že kód pod licencí GPL nelze používat v closed-source programech. Nicméně stejná ustanovení rovněž způsobují nekompatibilitu GPL s některými jinými svobodnými licencemi. Obvykle k tomu dochází tak, že jiná licence vznáší určitý požadavek—například doložka o uvádění jmen autorů, která požaduje určitý způsob uvádění jmen původních autorů—který je nekompatibilní s duchem ustanovení GPL „Nesmíte zavést žádná další omezení..." Z pohledu Free Software Foundation jsou tyto důsledky druhého řádu žádoucí nebo alespoň nejsou tragické. GPL totiž nejen uchovává svobodu vašeho softwaru, ale účinným způsobem vytváří z vašeho softwaru nástroj, kterým *ostatní* software tlačí k tomu, aby i on vymáhal svobodu.

Řešení otázky, zda se jedná o správný způsob propagace svobodného softwaru, je jednou z nejdéle trvajících svatých válek na internetu (viz část **Vyvarujte se svatých válek** v kapitole **6. Komunikace**), zde ji však nebudeme déle zkoumat. Pro naše účely je důležité zjištění, že kompatibilita GPL je závažným faktorem při výběru licence. GPL je daleko nejpopulárnější open source licencí; viz **http://freshmeat.net/stats/#license**, pohybuje se kolem 68 %, druhá nejoblíbenější licence má pouhých 6 %. Pokud chcete, aby váš kód mohl být volně směšován s kódem podléhajícím GPL—a GPL kódů existuje skutečně mnoho—musíte si vybrat licenci kompatibilní s GPL. Většina open source licencí kompatibilních s GPL je také kompatibilních s proprietárními licencemi: tzn. kód pod takovouto licencí lze použít v programech podléhajících GPL i v proprietárních programech. Samozřejmě, že *výsledky* tohoto směšování už nebudou vzájemně kompatibilní, neboť jeden bude spadat pod licenci GPL a druhý pod closed-source licenci. Tyto záležitosti se však týkají pouze odvozených děl, nikoli kódu, který šíříte na úplném počátku.

Free Software Foundation naštěstí vede seznam licencí, které jsou kompatibilní s GPL a které s GPL kompatibilní nejsou, viz **http://www.gnu.org/licenses/license-list.html**. Všechny licence, o nichž pojednává tato kapitola, jsou na tomto seznamu uvedeny, buď na jedné, či na druhé straně.

Volba licence

Při volbě licence, kterou budete aplikovat na váš projekt, použijte již existující licenci namísto vytváření licence nové, je-li to možné. Stávající licence jsou lepší ze dvou důvodů:

- Obeznámenost. Pokud použijete jednu ze tří či čtyř nejpopulárnějších licencí, lidé nebudou cítit nepříjemnou povinnost pročítat složitou právnickou hantýrku jen proto, aby mohli používat váš kód; danou licenci totiž už budou dlouhou dobu znát.
- Kvalita. Pokud nemáte k dispozici tým právníků, sotva můžete vytvořit licenci, která by byla z právnického hlediska celistvá. Zde uváděné licence jsou výsledkem mnohaletého přemýšlení a zkušeností; nemá-li váš projekt skutečně neobvyklé potřeby, pravděpodobně nevytvoříte žádnou lepší licenci.

Způsob, jak některou z těchto licencí aplikovat na váš projekt, naleznete v části **Jak licenci aplikovat** v kapitole **2. Zahájení projektu.**

MIT / X Window System License

Pokud chcete, aby byl váš kód přístupný co největšímu počtu vývojářů a odvozených děl a nevadí vám, že bude využíván v proprietárních programech, vyberte si licenci MIT / X Window System (jedná se o licenci, pod níž Massachusetts Institute of Technology vydalo původní kód X Window System) Základní poselství této licence hlásá: „Tento kód můžete používat, jak je vám libo.“ Je kompatibilní s GNU GPL, krátká, jednoduchá a velice srozumitelná:

```
Copyright (c) <year> <copyright holders>

Tímto je osobě, která obdrží kopii tohoto softwaru a související
dokumentační soubory (dále jen „Software“), udělováno bezplatné povolení
obchodovat se Softwarem bez jakýchkoli omezení, mimo jiné jej používat,
upravovat, slučovat, publikovat, šířit, sublicencovat a/nebo prodávat kopie
Softwaru a totéž povolovat osobám, jimž je Software dále poskytován,
při splnění následujících podmínek:

Výše uvedené označení autorského práva a toto povolení musí být uvedeno
ve všech kopiích či podstatných částech softwaru.
```

```
SOFTWARE SE NABÍZÍ TAK, JAK JE, BEZ JAKÉKOLI ZÁRUKY, AŤ JIŽ
VÝSLOVNÉ ČI PŘEDPOKLÁDANÉ, VČETNĚ ZÁRUK OBCHODOVATELNOSTI, ZPŮSOBILOSTI
A POUŽITELNOSTI PRO URČITÝ ÚČEL A PATENTOVÉ ČISTOTY. ZA ŽÁDNÝCH OKOLNOSTÍ
NEBUDOU AUTOŘI ČI DRŽITELÉ AUTORSKÝCH PRÁV ODPOVĚDNÍ ZA NÁROKY, ŠKODY NEBO
JINÉ ZÁVAZKY PLYNOUCÍ ZE ŽALOBY NA ZÁKLADĚ SMLOUVY, ÚMYSLNÉHO PORUŠENÍ PRÁVA
ČI JINÉ POVINNOSTI V SOUVISLOSTI SE SOFTWAREM NEBO POUŽÍVÁNÍM ČI JINÝMI
TRANSAKCEMI SE SOFTWAREM.
```

(Převzato z **http://www.opensource.org/licenses/mit-license.php.**)

GNU General Public License

Pokud preferujete, aby váš projektový kód nebyl používán v proprietárních programech nebo vám je přinejmenším jedno, zda v nich bude používán, vyberte si licenci GNU General Public License (**http://www.fsf.org/licensing/licenses/gpl.html**). GPL je v současnosti pravděpodobně nejpoužívanější licencí pro svobodný software na světě; toto okamžité „poznávací znamení“ je samo o sobě jednou z největších výhod GPL.

Při psaní knihovny kódů, která je především určena k využití jako součást jiných programů, pečlivě zvažte, zda jsou omezení stanovená GPL v souladu s cíli vašeho projektu. V některých případech—například, pokud chcete vyhodit ze sedla konkurenční proprietární knihovnu, která provádí stejnou operaci—by bylo strategičtější opatřit váš kód licencí, která umožní jeho začlenění do proprietárních programů, ačkoliv byste si to jinak sotva přáli. Free Software Foundation dokonce pro tyto případy vytvořila alternativu ke GPL: *GNU Library GPL*, později přejmenovanou na *GNU Lesser GPL* (většinou se používá zkratka LGPL). LGPL má mírnější omezení než GPL a lze ji snadněji kombinovat s nesvobodným kódem. Je však také trochu složitější a na první přečtení méně srozumitelná, čili pokud nebudete používat licenci GPL, doporučuji vám použít licenci typu MIT/X.

GNU Affero GPL: Verze GNU GLP pro kód na straně serveru

V roce 2007 vydala organizace Free Software Foundation variantu GPL nazvanou GNU Affero GPL (**http://www.fsf.org/licensing/licenses/agpl.html**).[30] Jejím účelem je vynutit si ustanovení pro sdílení podobná GPL u rostoucího počtu společností, které poskytují „hosted services“—tj. software, který běží na jejich serverech, s nimiž jsou uživatelé v interakci pouze přes síť, a který tak není nikdy přímo distribuován uživatelům jako spustitelný program či zdrojový kód. Mnoho těchto společností používalo software krytý licencí GPL, často upravený, nemusely však sdílet své změny s okolním světem, protože žádný kód nedistribuovaly.

Řešení GNU AGPLv3 bylo přidat do regulérní licence GPL klauzuli „Interakce přes dálkové sítě“ uvádějící: „... pokud upravíte Program, vámi upravená verze musí všem uživatelům, kteří se systémem interagují pomocí počítačové sítě, jasně nabízet... možnost získat odpovídající zdrojový kód vaší verze...

bezplatně, prostřednictvím některého ze standardních či běžných způsobů usnadňujících kopírování softwaru.“ Tím se prováděcí oprávnění GPL rozšířila do zcela nového světa poskytovatelů aplikačních služeb. Free Software Foundation doporučuje, aby GNU AGPLv3 byla používána pro veškerý software, který bude běžně spouštěn přes síť.

Upozorňuji, že AGPLv3 není přímo kompatibilní s GPLv2 (ačkoliv je samozřejmě kompatibilní s GPLv3). Nicméně většina softwaru chráněného licencí GPLv2 tak či tak zahrnuje ustanovení „nebo jakékoli pozdější verze“, takže jej můžete jen přesunout na GPLv3, pokud a když jej budete potřebovat sloučit s kódem krytým AGPLv3. Pokud však potřebujete kombinovat s programy, které jsou chráněny výhradně v rámci GPLv2 (tj. bez ustanovení „nebo jakékoli pozdější verze“), pak nebude AGPLv3 fungovat.

Ačkoliv je historie AGPLv3 trochu komplikovanější, vlastní licence je jednoduchá: jedná se pouze o GPLv3 s jednou zvláštní doložkou (ustanovením) o síťové interakci. V tomto ohledu chci upozornit na skvělý článek o AGPLv3 na Wikipedii: **http://en.wikipedia.org/wiki/Affero_General_Public_License**

Je GPL svobodná licence, či nikoli?

Jeden z důsledků volby GPL je možnost—sice malá, nikoli však nekonečně malá—že se vy či váš projekt stanete účastníkem sporu o to, zda je GPL skutečně „svobodná“ licence, neboť uvaluje určitá omezení na to, co můžete s kódem dělat—jmenovitě: omezení, že kód nelze šířit pod žádnou jinou licencí. Pro někoho existence tohoto omezení znamená, že GPL je „méně svobodná“ než jiné tolerantnější licence, jako např. MIT/X. Obvykle tento argument směřuje k tvrzení, že „svobodnější“ musí být lepší než „méně svobodný“ (koneckonců, kdo by měl něco proti svobodě?), z toho pak jasně vyplývá, že tyto tolerantnější licence jsou lepší než GPL.

Tato debata je dalším příkladem moderní svaté války (viz část **Vyvarujte se svatých válek** v kapitole **6. Komunikace**). Neúčastněte se jí, přinejmenším ne na projektových fórech. Nesnažte se dokazovat, že je GPL méně svobodná, stejně svobodná nebo svobodnější než jiné licence. Raději zdůrazněte specifické důvody, proč si váš projekt vybral GPL. Pokud jste si danou licenci vybrali pro její popularitu, řekněte to. Pokud jste si určitou svobodnou licenci zvolili také kvůli tomu, že je možno ji uplatňovat i u odvozeného díla, řekněte to taky; v každém případě se nenechávejte zatáhnout do diskuse, zda tato licence činí váš kód méně či více svobodným. Svoboda je přeci komplikovaná otázka a nemá příliš smysl se o ní bavit, pokud bude terminologie sloužit jen jako zástěrka.

---

[30] Historie této licence a jejího názvu je poněkud komplikovanější. První verzi této licence původně vydala společnost Affero, Inc, která ji založila na GNU GPL verzi 2. Běžně se jí říkalo AGPL. Později se Free Software Foundation rozhodla tuto myšlenku přijmout, ale zatím vydali verzi 3 své GNU GPL, takže na ní založili i svou novou „afferoizovanou“ licenci a nazvali ji „GNU AGPL“. Stará licence Affero je dnes víceméně odmítána. Pokud chcete použít ustanovení podobná stylu licence Affero, měli byste zvolit verzi GNU. Pro jednoznačnost uvádím, že byste ji měli správně označovat „AGPLv3L“, „GNU AGPL“ nebo nejpřesněji „GNU AGPLv3“.

Jelikož však čtete knihu a nikoli vlákno mailing listu, musím osobně připustit, že jsem nikdy nepochopil argument, že „licence GPL není svobodná“. Jediné omezení, které licence GPL vznáší, je, že zabraňuje lidem, aby zaváděli jakákoli *další* omezení. Říci, že v důsledku vede tato licence k umenšení svobody, mi připadá, jako by někdo tvrdil, že zákaz otroctví rovněž redukuje svobodu, protože se tak některým lidem zabraňuje vlastnit otroky.

(No a pokud se přeci jen do debaty necháte zatáhnout, tak rozhodně nepřilévejte olej do ohně uváděním pobuřujících analogií).

A co licence BSD?

Velké množství open source softwaru je distribuováno pod licencí BSD (označována také jako *BSD-style licence*). Původní licence BSD byla používána pro Berkeley Software Distribution, v níž University of California vydala důležité části implementace UNIX. Tato licence (přesné znění je uvedeno v kapitole 2.2.2 v **http://www.xfree86.org/3.3.6/COPYRIGHT2.html#6**) byla svým duchem podobná licenci MIT/X, s výjimkou jedné klauzule:

> *Všechny reklamní materiály, které zmiňují funkční vlastnosti nebo používání tohoto softwaru, musí uvádět následující oznámení: Tento produkt obsahuje software vyvinutý University of California, Lawrence Berkeley Laboratory.*

Přítomnost této klauzule nejen učinila původní licenci BSD nekompatibilní s GPL, ale také vytvořila nebezpečný precedens: zatímco jiné organizace umístily podobné klauzule o propagaci do *svého* svobodného softwaru—přičemž namísto „the University of California, Lawrence Berkeley Laboratory“ dosadily svá vlastní jména—redistributoři softwaru museli čelit stále náročnějším požadavkům na to, co všechno musí zobrazovat. Naštěstí si celá řada projektů, které tuto licenci používaly, tento problém uvědomila a příslušnou reklamní klauzuli prostě vypustila. V roce 1999 tak dokonce učinila i University of California.

Výsledkem těchto rozhodnutí pak byla revidovaná verze licence BSD, která je prostě původní licencí BSD s vypouštěnou reklamní klauzulí. Nicméně, díky této příhodě je slovní spojení „licence BSD“ poněkud dvojznačné: odkazuje na původní, nebo na revidovanou verzi? Proto preferuji licenci MIT/X, která je v zásadě ekvivalentní a nepřipouští dvojí výklad. Přesto však existuje důvod, proč preferovat revidovanou verzi licence BSD před licencí MIT/X, BSD totiž obsahuje následující ustanovení:

> *Název <ORGANIZACE> ani jména přispěvatelů nelze používat k podpoře či propagaci produktů odvozených od tohoto softwaru bez předchozího výslovného písemného souhlasu.*

Bez této klauzule totiž není jisté, zda může příjemce softwaru používat jméno/název poskytovatele licence, tato klauzule však všechny případné pochybnosti eliminuje. Pro organizace, které se pečlivě věnují správě svých ochranných obchodních známek, tak může být revidovaná licence BSD o něco lepší než MIT/X. Obecně však z licence s liberálními autorskými právy nevyplývá, že by příjemci

produktu měli právo používat či rozmělňovat vaše obchodní známky — zákon o autorském právu a známkové právo jsou dvě různé věci.

Pokud chcete používat revidovanou licenci BSD, její šablona je k dispozici na **http://www.opensource.org/licenses/bsd-license.php.**

Převod a vlastnictví autorských práv

Existují tři způsoby, jak nakládat s vlastnictvím autorských práv na svobodný kód a na dokumentaci, do níž přispívalo mnoho lidí. První způsob je naprosté ignorování problematiky autorských práv (tento způsob však rozhodně nedoporučuji). Druhý způsob je získat od všech osob, které na projektu spolupracovaly, tzv. *licenční smlouvu s přispěvatelem* (contributor license agreement) (*dále též „CLA“*), tato smlouva bude projektu výslovně zaručovat právo na používání příspěvků příslušné osoby. U většiny projektů tento způsob řešení zcela postačuje, navíc v některých právních řádech lze tuto smlouvu CLA zaslat i e-mailem. Třetí způsob je zajistit si převod autorských práv od přispěvatelů, projekt (tj. určitá právnická osoba, zpravidla nezisková organizace) tak bude držitelem autorských práv na vše. Po právní stránce je tento způsob nejdokonalejší; pro přispěvatele je však také nejnamáhavější; uplatňuje jej pouze několik málo projektů.[31]

Všimněte si, že i v rámci centralizovaného vlastnictví autorských práv zůstává kód[32] svobodný, protože open source licence neumožňují držiteli autorských práv zpětně si přivlastňovat všechny kopie kódu. Takže i kdyby projekt jako právnická osoba najednou obrátil a začal distribuovat veškerý kód pod restriktivní licencí, nemělo by to veřejné komunitě způsobit žádné problémy. Ostatní vývojáři by jednoduše vytvořili odnož založenou na poslední svobodné kopii kódu a pokračovali by dál, jakoby se nic nestalo. Jelikož o této možnosti dobře vědí, většina přispěvatelů bez problémů spolupracuje, pokud je požádáte, aby podepsali CLA nebo převod autorských práv.

Nedělat nic

Většina projektů si od svých přispěvatelů žádné CLA ani převody autorských práv nikdy nevyžádá. Naopak, přijmou kód, pokud je dostatečně jasné, že přispěvatel chce, aby byl začleněn do projektu. Za normálních okolností je to v pořádku. Ale kdykoli se může někdo rozhodnout, že bude žalovat porušení autorských práv, přičemž se bude prohlašovat za skutečného majitele dotyčného kódu a tvrdit, že nikdy nesouhlasil s tím, aby projekt šířil kód pod open source licencí.

---

[31] Vlastní převod autorských práv je podroben příslušným národním zákonům, licence určené pro Spojené státy tak mohou v jiných zemích narazit na problémy (např. v Německu, kde – jak známo – není převod autorských práv možný).

[32] Od této chvíle budu termín „kód“ používat jak pro kód, tak pro dokumentaci.

Podobnou věc provedla projektu Linux například společnost SCO Group, bližší informace viz **http://en.wikipedia.org/wiki/SCO-Linux_controversies.** Pokud k něčemu takovému dojde, projekt nebude mít v ruce žádnou dokumentaci, která by prokazovala, že mu přispěvatel formálně udělil práva na používání kódu, což může ztížit možnosti právní obhajoby.

Licenční smlouvy s přispěvatelem (CLA)

CLA nabízí asi nejlepší kompromis mezi bezpečností a snadným způsobem vyjednávání. CLA je standardně zpracována jako elektronický formulář, který vývojář vyplní a zašle projektu. Mnohé právní řády uznávají elektronické předložení této smlouvy za postačující. Bezpečnostní digitální podpis může, ale nemusí být vyžadován; s právníkem se poraďte, která metoda by pro váš projekt byla nejlepší.

Většina projektů používá dvě nepatrně odlišné smlouvy CLA, jednu pro fyzické osoby a druhou pro přispěvatele z řad firem/právnických osob. V obou typech smluv je však základní sdělení úplně stejné: přispěvatel uděluje projektu *„... stálou, celosvětovou, nevýlučnou, bezplatnou, nezpoplatněnou, neodvolatelnou licenci na autorská práva pro reprodukci, přípravu, odvozená díla, veřejnou prezentaci, veřejné provozování, sublicencování a distribuci příspěvků a takovýchto odvozených děl.“* I zde platí, že byste měli využít služeb právníka pro schválení všech CLA, pokud se vám však podaří dostat do smlouvy všechna výše uvedená přídavná jména, máte to víceméně ošetřeno.

Pokud žádáte od přispěvatelů CLA, zdůrazněte, že *nežádáte* o skutečný převod autorských práv. Mnoho CLA začíná upozorněním čtenáře, že:

> *Tento dokument je pouze licenční smlouvou; neslouží k převodu vlastnictví autorských práv a neupravuje vaše práva k používání vašich vlastních příspěvků pro jiné účely.*

Uveďme několik příkladů:

- CLA s jednotlivými přispěvateli:
  http://apache.org/licenses/icla.txt
  http://code.google.com/legal/individual-cla-v1.0.html
- CLA s firemními přispěvateli:
  http://apache.org/licenses/cla-corporate.txt
  http://code.google.com/legal/corporate-cla-v1.0.html

Převod autorských práv

Převod autorských práv znamená, že přispěvatel postupuje projektu vlastnictví autorských práv na své vlastní příspěvky. V ideálním případě se tento převod sepisuje na papír a odesílá faxem či klasickou poštou na adresu projektu.

Některé projekty trvají na řádném převodu práv, protože v případech, kdy je nutno vymáhat podmínky open source licence soudně, je vhodné, aby autorská práva na celou základnu kódu vlastnila jediná právnická osoba. Pokud neexistuje jeden subjekt, který by byl oprávněn tak učinit, mohou spolupracovat všichni přispěvatelé, ale někteří nemusí mít čas nebo nemusí být ani k zastižení, když bude zapotřebí tento problém řešit.

Různé organizace aplikují různý stupeň přísnosti při shromažďování převodů práv. Některým stačí neformální prohlášení přispěvatele na veřejném mailing listu—něco ve smyslu „Tímto postupuji projektu svá autorská práva k tomuto kódu, aby byl licencován za stejných podmínek jako zbývající části kódu.“ Nejméně jeden právník, s nímž jsem o této problematice hovořil, řekl, že toto prohlášení zcela postačuje, pravděpodobně proto, že je proneseno v kontextu, kde je postupování/převádění autorských práv běžné a očekávané a protože představuje *dobrý úmysl* ze strany projektu zjistit pravý záměr vývojáře. Free Software Foundation naopak přešla k opačnému extrému: žádá přispěvatele, aby fyzicky podepsali a poštou na papíře zaslali formální prohlášení o převodu autorských práv, někdy i pro jediný příspěvek, jindy pro všechny aktuální i budoucí příspěvky. Pokud je vývojář zaměstnán, FSF požaduje, aby toto prohlášení podepsal i zaměstnavatel.

Paranoia FSF je v tomto případě pochopitelná. Pokud někdo poruší podmínky GPL začleněním některého z jejich softwaru do proprietárního programu, FSF bude muset proti tomuto porušení bojovat u soudu, pro tyto případy chtějí, aby jejich autorská práva byla pokud možno neprůstřelná. Jelikož je FSF držitelem autorských práv na mnoho populárních softwarů, vidí tuto možnost jako reálnou hrozbu. O tom, zda má být i vaše organizace takto pedantská, rozhodujete pouze vy, po poradě s právníky. Obecně lze říci, že pokud váš projekt nepotřebuje z nějakého specifického důvodu plný převod autorských práv, stačí, když aplikujete licenční smlouvy CLA; jsou pro všechny zainteresované strany mnohem jednodušší.

Systém dvojitých licencí

Některé projekty se snaží financovat samy sebe pomocí systému dvojitých licencí, kde proprietární odvozená díla mohou platit držiteli autorských práv za právo užívat kód, ale kód zůstává nadále volně k užívání open source projekty. Tento systém funguje samozřejmě lépe u knihoven kódů než u samostatných aplikací. Přesné podmínky se případ od případu liší. Často je pro svobodnou stranu použita licence GNU GPL, jelikož ta již ostatním uživatelům brání začlenit dotyčný kód do jejich proprietárního produktu bez souhlasu držitele autorských práv, někdy je to však obyčejná licence, která má stejný účinek. Příkladem první licence je MySQL, popsaná na adrese **http://www.mysql.com/company/legal/licensing/**; příkladem druhého typu je licenční strategie Sleepycat Software, popsaná na adrese **http://www.oracle.com/technology/software/products/berkeley-db/htdocs/licensing.html.**

Možná se zeptáte: jak může držitel autorských práv nabízet proprietární licence za povinný poplatek, když podmínky GNU GPL stanovují, že kód musí být vždy k dispozici za méně restriktivních podmínek? Odpověď zní, že podmínky GPL jsou tím, co držitel autorských práv uvaluje na všechny ostatní;

sám držitel se tak může rozhodnout tyto podmínky na sebe *neaplikovat*. Podívejte se na celý problém takto: zkuste si představit, že držitel autorských práv má neomezený počet kopií softwaru, uložených v kbelíku. Pokaždé, když odtud vyjme jednu kopii a chce ji odeslat do světa, může se rozhodnout, jakou licenci jí přidělí: GPL, proprietární nebo nějakou jinou. Jeho právo provádět tento výběr není vázáno na GPL nebo jinou open source licenci; je to jednoduše pravomoc, kterou mu garantuje zákon o autorských právech.

Dvojité licencování je atraktivní v tom, že poskytuje způsob, jak může projekt svobodného softwaru získat spolehlivý zdroj příjmů. Bohužel však také může kolidovat s běžnou dynamikou open source projektů. Problém je v tom, že každý dobrovolník, který přispěje do kódu, přispívá do dvou různých entit: do svobodné verze kódu a do jeho proprietární verze. Zatímco přispěvatel bude rád přispívat do svobodné verze, v open source projektech je to koneckonců norma, přispívat do cizího poloprivátního zdroje příjmů mu bude zřejmě připadat směšné. Směšnost celé situace se vyostřuje skutečností, že v systému dvojitého licencování musí držitel autorských práv skutečně shromáždit formální podepsané převody autorských práv od všech přispěvatelů, aby sám sebe chránil před nevrlým přispěvatelem, který by si později nárokoval procenta z licenčních poplatků za proprietární software. Vlastní proces shromažďování těchto převodů práv konfrontuje přispěvatele se skutečností, že dělají práci, která někomu jinému vydělává peníze.

Ne všem dobrovolníkům bude tato skutečnost vadit; koneckonců jejich příspěvky jdou také do open source verze, což může být skutečný středobod jejich zájmů. Nicméně dvojité licencování je stav, kdy si držitel autorských práv sám připisuje zvláštní práva, která ostatní účastníci projektu nemají; je tedy jasné, že tento stav v určitou chvíli vyvolá napětí, přinejmenším u některých dobrovolníků.

V praxi to vypadá tak, že společnosti založené na softwaru s dvojitou licencí nemají skutečně rovnostářské vývojové komunity. Z externích zdrojů mají opravy chyb menšího měřítka a čisticí záplaty, ale většinu těžké práce musí stejně dělat z vnitřních zdrojů. Například, Zack Urlocker, zástupce ředitele pro marketing ve společnosti MySQL, mi řekl, že společnost musela stejně nakonec najmout nejaktivnější dobrovolníky. Tedy, ačkoliv je vlastní produkt open source licencován pod GPL, jeho vývoj je víceméně řízen společností, přičemž neustále hrozí (silně nepravděpodobná) možnost, že někdo skutečně nespokojený se způsobem, jakým společnost nakládá se softwarem, může vytvořit odnož. Nevím, do jaké míry tato hrozba předběžně formuje zásady a politiku společnosti, v každém případě to však nevypadá, že by MySQL byla přijímána s nějakými problémy, ani ve světě open source, ani mimo něj.

Patenty

Softwarové patenty jsou aktuálně v oblasti svobodného softwaru jakýmsi hromosvodem, neboť představují jedinou skutečnou hrozbu, vůči níž se komunita svobodného softwaru nedokáže sama bránit. Problémy s autorskými právy a obchodními známkami se vždycky dají nějak obejít. Pokud část vašeho kódu vypadá, že by mohla porušit něčí autorská práva, můžete ji jednoduše přepsat. Pokud se zjistí, že má někdo na název vašeho projektu ochrannou známku, v nejhorším případě můžete celý

projekt prostě přejmenovat. Ačkoli je takovéto přejmenování z krátkodobého hlediska velice nepříjemné, v dlouhodobém horizontu na něm zas tolik nezáleží, vlastní kód bude stejně dělat to, co dělal dřív.

Ale patent je oficiální zákaz použití určitého nápadu. Je úplně jedno, kdo kód napsal nebo jaký programovací jazyk byl použit. Jakmile někdo obviní projekt svobodného softwaru z porušení patentové ochrany, projekt musí buď zastavit implementaci určité funkční vlastnosti, nebo čelit nákladnému a časově náročnému soudnímu procesu. Jelikož jsou nejčastějšími iniciátory těchto soudních sporů bohaté společnosti—tj. někdo, kdo má prostředky a dispozice získat patenty v první řadě—většina projektů svobodného softwaru si nemůže druhou možnost (soudní proces) dovolit, a musí tak předem kapitulovat, ač se může domnívat, že patent by byl u soudu pravděpodobně nevymahatelný. Aby se takovéto situaci vyhnuli, začínají projekty svobodného softwaru kódovat defenzivně, tj. předem vylučují patentované algoritmy, i když jsou tím nejlepším, nebo dokonce jediným dostupným řešením daného programovacího problému.[33]

Podle průzkumů a nepřímých či neoficiálních důkazů je nejen drtivá většina programátorů open source, ale i většina *všech* programátorů přesvědčena, že softwarové patenty by měly být zcela zrušeny.[34] Programátoři open source softwaru jsou zpravidla v této otázce neochvějní, a mohou tak odmítnout práci na projektech, které jsou příliš silně spojovány se shromažďováním a vymáháním softwarových patentů. Pokud vaše organizace shromažďuje softwarové patenty, učiňte veřejné a neodvolatelné prohlášení, že tato patentová práva nebudou nikdy uplatňována vůči open source projektům a že je hodláte použít pouze pro vlastní obhajobu v případě, že některá jiná strana dá vůči vaší organizaci podnět k soudnímu řízení ve věci porušení patentových práv. Není to jen obecně správný krok, je to dokonce velice správná práce s veřejností v oblasti open source.[35]

Shromažďování patentů pro obranné účely je dnes bohužel velice racionální krok. Současný patentový systém, přinejmenším ten americký, svou povahou připomíná spíše závody ve zbrojení: pokud vaši konkurenti získali velké množství patentů, nejlepší obranou pro vás je získat jich taky hodně. Budete-li totiž někdy vystaveni hrozbě soudního sporu ve věci porušení patentových práv, můžete reagovat podobnou hrozbou—v takovém případě si obvykle obě znesvářené strany spolu sednou a vypracují dohodu o vzájemném poskytnutí licencí, aby žádná z nich nemusela nic platit... tedy samozřejmě kromě svých právníků.

---

[33] Společnosti Sun Microsystems a IBM však v této otázce učinily přinejmenším vstřícné gesto a uvolnily velké množství softwarových patentů—1600, respektive 500—pro použití v open source komunitách. Nejsem právník, a nemohu tedy fundovaně ocenit skutečnou užitnou hodnotu tohoto gesta, ale i kdyby se jednalo o důležité patenty a dle podmínek tohoto daru by skutečně byly volně k použití v open source projektech, byla by to pořád jen kapka v moři.

[34] Jeden z takovýchto průzkumů je k dispozici zde **http://groups.csail.mit.edu/mac/projects/lpf/Whatsnew/survey.html.**

[35] Například společnost RedHat vydala oficiální příslib, že open source projekty budou před jejími patentovými právy bezpečně kryty **http://www.redhat.com/legal/patent_policy.html.**

Škoda, kterou softwarové patenty svobodnému softwaru způsobují, je však zákeřnější než pouhé přímé hrozby vůči vývoji kódu. Softwarové patenty vyvolávají mezi firmwarovými programátory atmosféru tajnůstkaření, kteří se oprávněně bojí, že zveřejněním detailů o svých rozhraních poskytnou technickou pomoc konkurenci, která se je bude snažit utlouct soudními spory o porušení patentových práv. Toto nebezpečí není pouze teoretické; již dlouhou dobu se s ním můžete setkat například v oblasti grafických karet. Mnoho výrobců grafických karet se zdráhá publikovat podrobnější programovací specifikace, které jsou zapotřebí pro vytvoření vysokovýkonnostních open source ovladačů pro jejich grafické karty, tím brání svobodným operačním systémům v podpoře těchto karet, která by mohla plně využít jejich potenciál. Jistě se zeptáte: proč by to výrobci dělali? Byl by nesmysl, aby podnikali kroky *proti* softwarové podpoře; vždyť kompatibilita s větším množstvím operačních systémů přeci znamená větší tržby z prodeje karet. Zjistilo se však, že za dveřmi pracoven programátorů probíhá vzájemné porušování patentových práv mezi jednotlivými výrobci, někdy vědomé, jindy neúmyslné. Patenty jsou tak nepředvídatelné a jejich působnost může být tak široká, že si žádný výrobce karet nemůže být nikdy jistý, že je stoprocentně chráněn před jejich porušením, a to i přes důkladnou patentovou rešerši . Proto se výrobci neodvažují publikovat plné specifikace svých rozhraní, jejich konkurenti by si tak totiž snadno mohli udělat obrázek o tom, zda jsou porušena některá patentová práva. (Povaha této záležitosti je takového druhu, že o její existenci samozřejmě nikde nenajdete písemné potvrzení z primárního zdroje; osobně jsem se o tom dozvěděl v soukromém rozhovoru.)

Některé licence na svobodný software obsahují speciální klauzule, které bojují nebo se přinejmenším snaží odradit od uplatňování softwarových patentů. Například GNU GPL obsahuje text následujícího znění:

```
  7. Pokud jsou vám, v důsledku soudního rozhodnutí nebo obvinění z porušení
patentových práv nebo z jiných důvodů (nikoli jen v záležitostech týkajících
se patentů), uloženy takové podmínky (ať již rozhodnutím soudu, smlouvou
nebo jinak), které se vylučují s podmínkami této Licence, nejste tím
osvobozen od podmínek této Licence. Pokud nemůžete šířit chráněné dílo tak,
abyste vyhověl zároveň svým závazkům vyplývajícím z této Licence a jiným
platným závazkům, nesmíte jej šířit vůbec. Pokud by například patentové
osvědčení nepovolovalo bezplatnou redistribuci programu všemi, kdo vaším
přičiněním získají přímo nebo nepřímo jeho kopie, pak by jediný možný
způsob, jak vyhovět zároveň patentovému osvědčení i této licenci, spočíval
v ukončení distribuce programu.

[...]

Tato kapitola vás nechce navádět k porušování jakýchkoli patentů nebo jiných
nároků spojených s majetkovými právy, ani nechce napadat platnost
takovýchto nároků či žalob; má jediný cíl: chránit integritu systému pro
šíření svobodného softwaru, který je zaveden díky veřejným licencím. Mnoho
lidí dodalo rozsáhlé příspěvky celé řadě softwaru distribuovaného
```

```
prostřednictvím tohoto systému, přičemž se spoléhali na soustavnou aplikaci
tohoto systému; je čistě na autorovi/dárci, aby se sám rozhodl, zda chce
distribuovat software prostřednictvím jiného systému a příjemce licence
nemůže tuto volbu nijak požadovat.
```

I licence Apache License, verze 2.0 (**http://www.apache.org/licenses/LICENSE-2.0**), obsahuje protipatentové požadavky. Zaprvé požaduje, aby každý, kdo pod touto licencí distribuuje kód, implicitně zahrnul bezplatnou patentovou licenci pro všechny patenty, které vlastní a které by se mohly kódu týkat. Zadruhé, což je nanejvýš důmyslné, trestá každého, kdo podá žalobu na porušení patentových práv souvisejících s dílem krytým touto licencí, a automaticky dotyčnému odebírá implicitní patentovou licenci ve chvíli, kdy je takováto žaloba podána:

```
3. Udělení patentové licence. Podle podmínek této licence vám každý
přispěvatel tímto uděluje trvalou, celosvětovou, nevýlučnou, bezplatnou,
nezpoplatněnou, neodvolatelnou (s výjimkami uvedenými v této kapitole)
patentovou licenci k provádění, zadávání, používání, nabízení k prodeji,
prodeji, dovozu nebo jinému přenosu díla, pokud se tato licence týká pouze
patentových nároků – licencovatelných tímto přispěvatelem – které jsou nutně
porušeny jeho příspěvkem (příspěvky), ať již samostatně nebo v kombinaci
tohoto příspěvku (příspěvků) s dílem, k němuž byl tento příspěvek
(příspěvky) předložen(y). Pokud zahájíte patentní spor vůči jakémukoli
subjektu (včetně křížového nároku či protižaloby v soudním řízení), přičemž
budete tvrdit, že dané dílo či příspěvek začleněný do tohoto díla
představuje přímé či nepřímé porušení patentových práv, budou vám k datu
podání žádosti o tento spor odebrány všechny patentové licence, jež jsou vám
v rámci této licence na dílo uděleny.
```

Ačkoliv je z právního i politického hlediska užitečné zakomponovat tímto způsobem patentovou ochranu do licencí na svobodný software, v důsledku nebudou tato opatření dostatečně účinná na to, aby rozptýlila obavy, které v komunitě svobodného softwaru vyvolala hrozba patentových sporů.

Tyto obavy by mohla účinně rozptýlit pouze změna v podstatě či výkladu mezinárodního patentového práva. Chcete-li se o tomto problému dozvědět více, včetně toho, jak se proti němu bojuje, vizte adresu **http://www.nosoftwarepatents.com/**. Článek ve Wikipedii **http://en.wikipedia.org/wiki/Software_patent** také poskytuje mnoho užitečných informací o softwarových patentech. Argumenty proti softwarovým patentům jsem rovněž shrnul v článku na svém blogu, viz **http://www.rants.org/2007/05/01/how-to-tell-that-software-patents-are-a-bad-idea/.**

Další zdroje

Tato kapitola je pouhým úvodem do problematiky licencí svobodného softwaru. I když doufám, že obsahuje dostatek informací, abyste mohli sami začít u svého vlastní open source projektu, jakýkoli hlubší výzkum v otázkách týkajících se licencí rychle vyčerpá vše, co tato publikace může nabídnout. Níže uvádím seznam dalších zdrojů informací o licencování open source:

- *Understanding Open Source and Free Software Licensing* (Průvodce udělováním licencí k open source a svobodnému softwaru), autor Andrew M. St. Laurent. O'Reilly Media, 1. Vydání, srpen 2004, ISBN: 0-596-00581-4.

  Vyčerpávající publikace o udělování licencí k open source softwaru, pojatá v celé složitosti problému, včetně mnoha témat, která byla v této kapitole opomenuta. Podrobněji viz **http://www.oreilly.com/catalog/osfreesoft/**.

- *Make Your Open Source Software GPL-Compatible Or Else* (Udělejte svůj open source software kompatibilní s GPL nebo jinak) od Davida A. Wheelera , na adrese **http://www.dwheeler.com/essays/gpl-compatible.html.**

  Tento detailní a skvěle napsaný článek vysvětluje, proč je důležité používat licence kompatibilní s GPL, ač vlastní GPL nepoužíváte. Článek se rovněž dotýká mnoha dalších otázek ohledně licencí a nabízí mnoho vynikajících odkazů.

- **http://creativecommons.org/**
  Creative Commons je organizací, která prosazuje celou řadu flexibilnějších a liberálnějších autorských práv než běžná autorsko-právní praxe. Nabízejí licence nejen na software, ale také na text, výtvarná díla a hudbu, všechny jsou dostupné prostřednictvím uživatelsky příjemného prostředí pro výběr licencí; některé z těchto licencí jsou copyleft, jiné nejsou, ale jsou i tak svobodné, další jsou tradiční „copyrighty“ s některými méně striktními omezeními. Webová stránka Creative Commons podává ke všem těmto licencím maximálně jasná vysvětlení. Kdybych měl jmenovat jednu internetovou stránku, která představuje širší filosofické aspekty hnutí svobodného softwaru, byla by to právě tato.

A. Svobodné systémy pro správu verzí

A. Svobodné systémy pro správu verzí

Zde uvádím všechny open source systémy pro správu verzí, jež mi byly známy v polovině roku 2007. Pravidelně jsem využíval pouze Subversion. S většinou těchto systémů jsem měl nepatrné či žádné zkušenosti, s výjimkou Subversion a CVS; níže uvedené informace jsou převzaty z jejich webových stránek. Viz také **http://en.wikipedia.org/wiki/List_of_revision_control_software**.

CVS — http://www.nongnu.org/cvs/
CVS existuje už delší dobu a mnoho vývojářů jej důvěrně zná. Svého času byl tento systém skutečně převratný: jednalo se o vůbec první open source verzi řídícího systému s přístupem skrz WAN síť pro vývojáře (alespoň co vím) a také o první systém, který nabízel anonymní read-only checkout, což novým vývojářům umožňovalo snadnější zapojení do projektů. CVS umožňujě verzovat pouze soubory, nikoli adresáře; nabízí větvení, označování (tagging) a dobrou výkonnost na straně klienta, bohužel příliš úspěšně nezvládá větší nebo binární soubory. Navíc nepodporuje atomické potvrzování změn. [DISCLAIMER: *Aktivně jsem se podílel na vývoji CVS po dobu přibližně pěti let, pak jsem napomáhal při spuštění projektu Subversion, který měl CVS nahradit.*]

Subversion — http://subversion.tigris.org/
Projekt Subversion byl především vytvořen jako nástupce CVS—tj. s přibližně stejným přístupem ke správě verzí jako CVS, ale s vyloučením všech problémů a opomenutí ve funkčních vlastnostech, které velice často obtěžují uživatele CVS. Jedním z cílů projektu Subversion zaměřeným na lidi, kteří si již zvykli na CVS, je nalézt způsob relativně hladkého přechodu na Subversion. Nemám zde dostatek prostoru, abych zabíhal do přílišných detailů o funkčních vlastnostech softwaru Subversion; bližší informace naleznete na jeho webové stránce. [DISCLAIMER: *Podílím se na vývoji softwaru Subversion, je to jediný ze zde uvedených systémů, který pravidelně využívám.*]

SVK — http://svk.elixus.org/
Ačkoliv je postaven na Subversion, SVK pravděpodobně spíše připomíná některý z decentralizovaných systémů uvedených níže. SVK podporuje distribuovaný vývoj, lokální potvrzování změn (commit), sofistikované slučování změn a schopnost zrcadlit stromy ze systémů třetích stran. Další detaily naleznete na jejich webové stránce.

Mercurial — http://www.selenic.com/mercurial/
Mercurial je distribuovaný systém pro správu verzí, který mimo jiné nabízí „kompletní odkazový rejstřík souborů a souborů změn; síťovou propustnost a CPU efektivní HTTP a SSH sync protokoly; libovolné slučování vývojových větví; integrované samostatné webové rozhraní; přenositelnost na UNIX, MacOS X a Windows“ a mnoho dalšího (tento seznam byl volně parafrázován z webové stránky Mercurial).

GIT — http://git.or.cz/
GIT je projekt spuštěný Linus Torvaldsem pro správu zdrojového stromu (jádra) Linux kernel. GIT byl zprvu spíše úzce zaměřen na potřeby vývoje jádra, ale postupně se rozvinul a v současnosti jej používají jiné projekty než Linux kernel. Jeho domovská stránka o tomto systému říká, že „...je určen k rychlé a efektivní manipulaci s velmi rozsáhlými projekty; využívá se hlavně pro různé open source projekty, zejména pro Linux kernel. GIT patří do kategorie nástrojů pro správu distribuovaného zdrojového kódu, podobá se např. GNU Arch nebo Monotone (případně BitKeeper v oblasti proprietárního softwaru). Každý pracovní adresář GIT je plnoprávným úložištěm s kompletní možností sledování revizí, nezávisle na síťovém přístupu nebo centrálním serveru.“

Bazaar — http://bazaar-vcs.org/
Bazaar (zkratka „bzr“) je distribuovaný systém pro správu verzí, který se soustředí na jednoduché používání a flexibilní model dat. Jedná se o oficiální projekt GNU a prvotní systém pro správu verzí určený hostitelské stránce svobodného softwaru Launchpad.net. Bazaar provádí plně distribuovanou správu verzí: veškerá práce probíhá ve větvích a každý vývojář má standardně k dispozici kompletní kopii historie větve. Větve lze vzájemně slučovat decentralizovaným způsobem, ale Bazaar lze také nakonfigurovat tak, aby fungoval centralizovaně. Bazaar začínal jako fork v rámci GNU Arch, byl ale celý od základu přepsán a dnes již s GNU Arch nemá žádnou přímou souvislost.

Darcs — http://darcs.net
„David's Advanced Revision Control System je dalším nástupcem CVS. Je napsán v programovacím jazyku Haskell a používá se s Linuxem, MacOS X, FreeBSD, OpenBSD a Microsoft Windows. Darcs obsahuje cgi script, který lze používat k zobrazení obsahu vašeho úložiště.“

Arch — http://www.gnu.org/software/gnu-arch/
GNU Arch podporuje jak distribuovaný, tak centralizovaný vývoj. Vývojáři potvrzují své změny do „archívu“, který může být lokální, a tyto změny lze přesouvat do jiných archívů, podle toho, zda se do nich dle správců těchto archívů hodí či nikoli. Tato metodologie jasně naznačuje, že Arch má propracovanější podporu slučování než CVS. Arch rovněž umožňuje snadno vytvářet větve archívů, k nimž příslušný vývojář nemá commit access. Tento systém je zde popsán pouze ve stručnosti; další informace viz na webových stránkách Arch.

monotone — http://www.venge.net/monotone/
„monotone je svobodný distribuovaný systém pro správu verzí. Poskytuje jednoduché, jednosouborové úložiště transakčních verzí s plně odděleným provozem a efektivním peer-to-peer synchronizačním protokolem. Rozumí slučování s historickou citlivostí, odlehčeným větvím, integrované revizi kódu a testování třetí stranou. Využívá kryptografické pojmenovávání verzí a RSA certifikáty na straně klienta. Má dobrou internacionalizovanou podporu, není nijak externě závislý, běží pod Linuxem, Solarisem, OSX i ve Windows, s licencí v rámci GNU GPL.

Codeville — http://codeville.org/
„K čemu další systém pro správu verzí? Všechny ostatní systémy pro správu verzí vyžadují, abyste uchovávali pečlivý záznam o vztazích mezi jednotlivými větvemi, a vyhnuli se tak opakovanému slučování stejných konfliktů. Codeville je mnohem více anarchický. Umožňuje vám aktualizovat nebo potvrzovat změny do jakéhokoli úložiště, a to kdykoli bez jakýchkoli zbytečných opakovaných slučování."

„Codevill funguje tak, že vytvoří identifikátor pro každou změnu, která se provede, a zapamatuje si seznam všech změn, které byly aplikovány na každý jednotlivý soubor, a poslední změnu, kterou se upravil každý řádek každého souboru. Pokud se objeví jakýkoli konflikt, program zkontroluje, zda jedna ze dvou stran již byla aplikována na druhou, a pokud ano, automaticky nechá druhou stranu vyhrát. Pokud existuje skutečný konflikt verzí, který nelze automaticky sloučit, Codevill se chová téměř stejně jako CVS."

Vesta — http://www.vestasys.org/
„Vesta je přenosný systém SCM (Software Configuration Management), tj. systém pro správu konfigurace softwaru, určený pro podporu vývoje softwarových systémů téměř všech velikostí, od skutečně malých (méně než 10.000 zdrojových řádků) až ke skutečně rozsáhlým systémům (10.000.000 zdrojových řádků)."

„Vesta je vyspělý systém. Je výsledkem více než desetiletého výzkumu a vývoje prováděného společností Compaq/Digital Systems Research Center a je produktivně využíván mikroprocesorovou skupinou Alpha společnosti Compaq již déle než dva a půl roku. Skupinu Alpha tvoří více než 150 aktivních vývojářů na dvou pracovištích vzdálených od sebe tisíce mil, na východním a západním pobřeží Spojených států. Skupina využívá systém Vesta ke správě sestav se 130 MB zdrojových dat, z nichž každá produkuje 1,5 GB odvozených dat. Sestavy vytvářené pracovištěm na východním pobřeží produkují denně v průměru 10-15 GB odvozených dat, která jsou všechna pod správou systému Vesta. I když byl systém Vesta vytvořen za účelem vývoje softwaru, skupina Alpha prokázala jeho flexibilitu při použití během vývoje hardwaru, kontrolou souborů s jazykem pro popisování hardwaru do zařízení pro kontrolu zdrojového kódu Vesta a sestavovacích simulátorů a dalších odvozených objektů pomocí sestavovacího programu Vesta. Členové někdejší skupiny Alpha, nyní součásti Intelu, dále pokračují ve využívání systému Vesta v novém mikroprocesorovém projektu."

Aegis — http://aegis.sourceforge.net/
Aegis je transakční systém pro správu konfigurace softwaru. Nabízí framework, v němž může tým vývojářů nezávisle pracovat na mnoha změnách programu, přičemž Aegis koordinuje zpětné integrování těchto změn do hlavního zdroje programu s co možná nejméně přerušeními.

CVSNT — http://cvsnt.org/
„CVSNT je pokročilý multi-platformní systém pro správu verzí. Je kompatibilní s průmyslovým standardním protokolem CVS, podporuje mnoho dalších funkčních vlastností. (...) CVSNT je open source,

svobodný software pod licencí GNU General Public License.“ Seznam jeho funkčních vlastností obsahuje položky jako například: ověřování přes všechny standardní CVS protokoly a specifický Windows protokol SSPI a Active Directory; podpora zabezpečeného přesunu přes server nebo kódované SSPI; přenositelný (běží v prostředí Windows nebo Unix); NT verze je plně integrována se systémem Win32; díky MergePoint zpracování už není zapotřebí další tagování pro slučování; nadále v aktivním vývoji.

META-CVS — http://common-lisp.net/project/meta-cvs/
„Meta-CVS je systém pro správu verzí založený na CVS. Uchovává si sice většinu funkčních vlastností CVS, včetně veškeré síťové podpory, je však mnohem schopnější než CVS a snadněji se používá. Seznam funkčních vlastností, uvedený na webové stránce META-CVS, zahrnuje: možnost tvoření verzí struktury adresářů, vylepšená manipulace s typy souborů, jednodušší a uživatelsky příjemnější větvení a slučování, podpora symbolických odkazů, seznamy vlastností připojené k datům verzí, vylepšený import dat třetí strany a jednoduchý upgrade ze základního CVS.“

OpenCM — http://www.opencm.org/
„OpenCM je koncipován jako bezpečná náhrada CVS s vysokou integritou. Seznam klíčových funkcí a vlastností naleznete na stránce s funkčními vlastnostmi (features page). Nenabízí sice tolik funkčních vlastností jako CVS, podporuje však několik užitečných prvků, které CVS postrádá. V krátkosti lze říci, že OpenCM poskytuje prvotřídní podporou při přejmenovávání a konfiguraci, kryptografické ověřování a řízení přístupu a prvotřídní větvení.“

PRCS — http://prcs.sourceforge.net/
„PRCS, neboli Project Revision Control System (systém pro řízení revizí projektu) je frontend souboru nástrojů, které (jako CVS) nabízejí způsob, jak lze se skupinami souborů a adresářů manipulovat jako s jedinou entitou, při zachování koherentních verzí celé skupiny. (...) Je používán k podobným účelům jako SCCS, RCS a CVS, ale (přinejmenším podle slov jeho autorů) je mnohem jednodušší než kterýkoli z těchto systémů.“

ArX — http://www.nongnu.org/arx/
„ArX je distribuovaný systém pro správu verzí nabízející větvení a slučování, kryptografické ověřování integrity dat a možnost snadného publikování archívů na HTTP serveru.“

SourceJammer — http://www.sourcejammer.org/
„SourceJammer je systém pro řízení a verzování zrojového kódu, je napsán v jazyku Java. Tvoří jej komponenta na straně serveru, která uchovává historii souborů a verzí a obstarává přihlašování, odhlašování atd. a další příkazy; naopak komponenta na straně klienta dává serveru dotazy a spravuje soubory na klientově straně souborového systému.“

FastCST — http://www.zedshaw.com/projects/fastcst/index.html
„‘Moderní‘ systém využívající sady změn u revizí souborů a distribuovaných operací, nikoli centralizovaného řízení. Pokud máte e-mailový účet, můžete používat FastCST. Pro rozsáhlejší distribuci potřebujete pouze FTP server a/nebo HTTP server, případně můžete využít zabudovaný příkaz „serve“, kterým přímo obsloužíte vaše objekty. Všechny sady změn jsou univerzálně jedinečné a mají velké množství metadat, takže před vlastním vyzkoušením změn můžete zamítnout vše, co nechcete. Slučování se provádí porovnáním slučované sady změn vůči aktuálnímu obsahu adresářů, nikoli pokusy o sloučení s jinou sadou změn.“

Superversion — http://www.superversion.org/
„Superversion je multi-uživatelský distribuovaný systém pro správu verzí, založený na sadách změn. Je zamýšlen jako průmyslově výkonná open-source alternativa k běžně prodávaným komerčním řešením, nabízí stejně snadné (dokonce snadnější) používání a podobnou výkonnost. Intuitivní a efektivní použitelnost byly od počátku jednou z hlavních priorit při vývoji Superversion.“

B. Volně přístupné bug trackery (záznamníky chyb)

B. Volně přístupné bug trackery (záznamníky chyb)

Bez ohledu na to, jaký bug tracker daný projekt používá, někteří vývojáři si na něj neustále rádi stěžují. Mezi všemi ostatními standardními vývojářskými nástroji je vůči bug trackerům vznášeno vždy nejvíce stížností. Myslím si, že je to především kvůli tomu, že bug trackery jsou natolik vizuální a interaktivní nástroje, že si velice snadno dokážeme představit různá vylepšení, která bychom na nich provedli (kdybychom na to měli čas), a tato potenciální vylepšení rádi vytrubujeme do světa. Berte tyhle nezbytné stížnosti s rezervou—, mnohé z následujících bug trackerů jsou velice kvalitní.

V rámci tohoto seznamu se termín „issue“ (problém) používá k označení položek zaznamenávaných trackery. Pamatujte však na to, že každý systém může používat svou vlastní terminologii, v níž může mít termín „issue“ ekvivalenty jako „artifact“, „bug“ nebo nějaký jiný.

Bugzilla — http://www.bugzilla.org/
Bugzilla je velice populární, aktivně udržovaný tracker a své uživatele podle všeho nadmíru uspokojuje. V práci používám už čtyři roky upravenou verzi Bugzilly a jsem s ní spokojen. Nelze ji příliš uzpůsobit uživatelským požadavkům, ale - ač je to trochu zvláštní - může to být její přednost: Instalace Bugzilly vypadají hodně podobně bez ohledu na místo, kde se používají, to znamená, že mnoho vývojářů je již navyklých na její rozhraní, a budou se tak cítit jako doma, ať jsou, kde jsou.

GNATS — http://www.gnu.org/software/gnats/
GNU GNATS je jeden z nejstarších open source bug trackerů; je široce užíván. Jeho největší předností je diverzita rozhraní (lze jej používat nejen prostřednictvím internetového prohlížeče, ale také e-mailem nebo nástroji příkazového řádku) a ukládání issues v prostém textu. Skutečnost, že jsou všechna issue data ukládána v textových souborech na disk, usnadňuje tvorbu uživatelských nástrojů pro zachycování a analýzu dat (například při vytváření statistických zpráv). GNATS je také schopen různými způsoby automaticky zachycovat e-maily a přiřazovat je k příslušným issues podle schémat (šablon) v záhlaví emailů, což velice usnadňuje konverzaci mezi zapisujícím uživatelem a vývojářem.

RequestTracker (RT) — http://www.bestpractical.com/rt/
Webová stránka RT uvádí následující: „RT je podnikový ticketový systém, který skupině uživatelů umožňuje inteligentní a účinnou správu úkolů, issue a dotazů či žádostí předložených komunitou uživatelů,“ což je v kostce vše podstatné. RT má šikovně propracované internetové rozhraní a podle všeho i dosti širokou instalovanou bázi. Rozhraní je po vizuální stránce poněkud složité, ale tento problém odpadne, jakmile si na toto rozhraní zvyknete. RT je licencován v rámci GNU GPL (jejich webové stránky to však z nějakého důvodu neobjasňují).

Trac — http://trac.edgewall.com/
Trac je něco víc než jen bug tracker: ve skutečnosti to je integrovaná wiki a bug tracking systém. Využívá wiki odkazy k propojení issue, souborů, sad změn ve správě verzí a prosté wiki stránky. Lze jej velice snadno nastavit a integrovat se Subversion (viz příloha **A. Svobodné systémy pro správu verzí**).

Roundup — http://roundup.sourceforge.net/
Roundup se velice snadno instaluje (vyžaduje Python 2.1 nebo novější verzi) a používá. Je opatřen webovým, e-mailovým a command-line rozhraním. Šablony issue dat a webové rozhraní lze uživatelsky upravit, podobně jako část logiky přechodu mezi stavy.

Mantis — http://www.mantisbt.org/
Mantis je online bug tracking systém, napsaný v PHP, využívající databázi MySQL jako úložiště. Má funkční vlastnosti, které byste předpokládali. Osobně toto webové rozhraní považuji za jasné, intuitivní a přehledné.

Flyspray — http://www.flyspray.org/
Flyspray je online bug tracking systém, napsaný v PHP. Jeho webové stránky Flyspray popisují jako „nekomplikovaný“ tracker, mezi jeho funkční vlastnosti patří: současná podpora více databází (aktuálně MySQL a PGSQL); současná podpora více projektů; „sledování“ úkolů s oznamováním změn (e-mailem nebo přes Jabber); souhrnná historie úkolů; tematizace CSS; soubory v přílohách; pokročilé vyhledávací funkce (přesto snadno použitelné); RSS/Atom feedy; wiki a plaintext vstupy; hlasování; grafy závislosti.

Scarab — http://scarab.tigris.org/
Scarab je koncipován jako bug tracker s vysokým uživatelským přizpůsobením, opatřený množstvím funkčních vlastností, nabízející víceméně souhrn všech funkčních vlastností ostatních bug trackerů; vkládání dat, výzvy, zprávy, oznámení zainteresovaným stranám, kolektivní shromažďování komentářů a zaznamenávání závislosti.

Scarab lze uživatelsky přizpůsobit prostřednictvím správcovských webových stránek. V jediné instalaci Scarab můžete mít aktivních několik „modulů“ (projektů). V rámci daného modulu můžete vytvářet nové typy issue (defekty, doplňky, úkoly, žádosti o podporu atd.) a přidávat libovolné příznaky, kterými sladíte tracker se specifickými požadavky vašeho projektu.

Koncem roku 2004 se blížilo vydání Scarab verze 1.0.

Debian Bug Tracking System (DBTS) — http://www.chiark.greenend.org.uk/~ian/debbugs/
Debian Bug Tracking System je neobvyklý tím, že veškeré vstupy a manipulace s issues jsou prováděny e-mailem: každý issue dostává svou zvláštní e-mailovou adresu. DBTS je dobře škálovatelný: **http://bugs.debian.org/** obsahuje například 277.741 issues.

Jelikož interakce probíhají prostřednictvím regulérních e-mailových klientů, tj. v prostředí, které je známé a snadno přístupné většině lidí, DBTS je vhodný pro manipulaci s velkým množstvím příchozích zpráv, které potřebují rychlou klasifikaci a reakci. Samozřejmě má tento systém i nevýhody. Vývojáři musí investovat určitý čas, aby se seznámili se systémem e-mailových příkazů, a uživatelé musí své bug reporty psát bez webového formuláře, který by jim pomáhal při výběru informací, jež se mají zapisovat. Systém nabízí uživatelům nástroje, jimiž lze zdokonalit odesílané bug reporty, jako například program reportbug s příkazovým řádkem nebo balíček debbugs-el pro Emacs. Většina uživatelů však tyto nástroje nevyužije; jednoduše budou psát e-maily ručně a mohou či nemusí dodržovat pokyny pro bug reporting vydávané vaším projektem.

DBTS má webové rozhraní určené pouze ke čtení, nahlížení a dotazování se na issue.

Trackery s Trouble Tickety
Jsou zaměřeny spíše na tracking ticketů u helpdesků nežli na bug tracking u softwaru. Pravděpodobně vám bude užitečnější normální bug tracker, tyto záznamové systémy uvádím pouze pro úplnost a proto, že mohou existovat neobvyklé projekty, pro něž může být systém trouble ticketů vhodnější než tradiční bug tracker.

WebCall — http://myrapid.com/webcall/

Bluetail Ticket Tracker (BTT) — http://btt.sourceforge.net/
BTT stojí někde mezi standardním trouble ticket trackerem a bug trackerem. Nabízí funkce pro nastavení soukromí, které jsou ve světě open source bug trackerů poněkud neobvyklé: uživatelé systému jsou rozdělení do kategorií Staff (zaměstnanci), Friend (přítel), Customer (zákazník) nebo Anonymous (anonymní uživatel) a v závislosti na dané kategorii je jím zpřístupněno více či méně dat. Nabízí určitou e-mailovou integraci, rozhraní s příkazovým řádkem a mechanismy pro převod e-mailů do ticketů. Dále je vybaven funkčními vlastnostmi pro uchovávání informací, které nejsou asociovány s žádným specifickým ticketem, jako například interní dokumentace nebo FAQ.

B. Volně přístupné bug trackery (záznamníky chyb)

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

Neměl byste se starat; je to úplně jedno a svůj čas můžete věnovat mnohem užitečnější činnosti.

Slavný „přístřeškový“ vzkaz Poul-Henning Kampa (jehož výňatek si lze přečíst v kapitole **6. Komunikace**) je výmluvným pojednáním o negativních tendencích, které ovládají skupinové diskuse.

Je zde přetištěn s jeho laskavým svolením. Původní URL je **http://www.freebsd.org/cgi/getmsg.cgi?fetch=506636+517178+/usr/local/www/db/text/1999/freebsd-hackers/19991003.freebsd-hackers**; pro snadnější odkazování ostatních lze rovněž použít **bikeshed.com**.

```
Věc: Přístřešek pro kola (jakékoli barvy) na zelenější trávě...
From: Poul-Henning Kamp <phk@freebsd.org>
Date: Sat, 02 Oct 1999 16:14:10 +0200
Message-ID: <18238.938873650@critter.freebsd.dk>
Sender: phk@critter.freebsd.dk
Bcc: Blind Distribution List: ;
MIME-Version: 1.0

[skrytá kopie committerům, hackerům]

Můj poslední příspěvek byl přijat dostatečně vřele, a tak mě nic neodradilo
od toho, abych odeslal další; dnes mám čas i náladu toto učinit.

Trochu jsem váhal nad správným způsobem distribuce tohoto příspěvku, tentokrát jej
rozesílám jako skrytou kopii committerům a hackerům, nic lepšího mě asi
nenapadne. Sám nejsem členem skupiny hackerů, ale o tom bude řeč později.

Věc, která mě tentokrát vyprovokovala k psaní, bylo vlákno nazvané „sleep(1)
should do fractional seconds“ (sleep (1) by měl podporovat zlomky sekund),
které nás obtěžuje už mnoho dní, možná i několik týdnů, nechce se mi ani
zjišťovat, jak vlastně dlouho.

Těm, kterým toto konkrétní vlákno uniklo: Gratuluji.

Tento plamenný boj rozpoutal návrh, upravit sleep(1) DTRT (Do Right Thing,
Udělal správnou věc), pokud mu bude zadán neceločíselný argument.
Nic víc už k tomuto tématu neřeknu, protože se jedná o tak zanedbatelnou
položku, že by si to člověk podle délky vlákna nikdy nepomyslel; položku,
které je věnováno více pozornosti než jiným *problémům*, kvůli nimž tady jsme.
```

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

Celá sága „sleep(1)“ je asi nejkřiklavějším příkladem diskuse o barvě přístřešku na kola, kterou jsme kdy ve FreeBSD vedli. Návrh byl promyšlen dobře, získali bychom kompatibilitu s OpenBSD a NetBSD, a přitom byli nadále plně kompatibilní s jakýmkoli kódem, který kdy kdo napsal.

Přesto bylo od oné chvíle vzneseno a spuštěno tolik připomínek, námitek, návrhů a změn, že by si jeden myslel, že taková změna musí zacelit všechny díry v ementálu nebo změnit chuť koly nebo něco podobně světoborného.

Někteří z vás se mě ptali: „Co je to zač, ten případ s přístřeškem na kola?“

Je to dlouhá historka, tedy spíše stará historka, protože dlouhá vlastně moc není. Počátkem 60. let 20. století napsal C. Northcote Parkinson knihu s názvem „Parkinsonův zákon“, která pojednávala o dynamice řízení lidí.

Určitě je k dostání na Amazonu nebo v knihovně vašeho tatínka, je to cenná kniha a čas s ní rozhodně nepromrháte, pokud máte rádi Dilberta, bude se vám určitě líbit i Parkinson.

Někdo mi nedávno řekl, že když si ji přečetl, zjistil, že dnes už platí jen z 50 %. To je po čertech slušný výsledek, řekl bych, mnoho současných knih o managementu má „úspěšnost střelby“ podstatně nižší a této knize je už více než 35 let.

V onom konkrétním příkladu s přístřeškem na kola je druhým hlavním činitelem jaderná elektrárna, což docela dobře ilustruje dobu, kdy byla kniha napsána.

Parkinson na tomto příkladu ukazuje, že je poměrně snadné přijít za představenstvem firmy a získat od nich schválení pro výstavbu jaderné elektrárny v hodnotě miliónů nebo i miliard dolarů, ale pokud navrhnete postavit přístřešek na kola, skončíte v nekonečných debatách.

Parkinsonovo vysvětlení je, že jaderná elektrárna je tak obrovská, tak drahá a tak složitá, že ji lidé nedokážou plně pochopit; místo toho, aby se o to pokusili, budou předpokládat, že předtím, než celá věc dospěla takhle daleko, už všechny důležité detaily zkontroloval někdo jiný. Richard P. Feynman ve svých knihách uvádí několik zajímavých a trefných příkladů týkajících se Los Alamos.

Jenže s přístřeškem na kola je to jiné. Něco takového sbijete dohromady za víkend a ještě se stihnete večer podívat na fotbal. Takže ať už jste připraveni sebelépe, ať už je váš návrh seberozumnější, vždy se najde někdo,

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

kdo se chopí příležitosti a předvede, že dělá svou práci, že věci věnuje pozornost, že u toho je.

V Dánsku tomu říkáme „nechat na něčem svůj otisk prstu“. Je to věc osobní cti a prestiže; jde o to, mít na co ukázat a říct: „Vidíte? To jsem dělal já.“ Toto chování je sice velmi typické pro politiky, ale pokud se naskytne vhodná příležitost, dopouští se jej kdekdo. Je to něco jako otisk stop v ještě mokrém betonu.

Klaním se původnímu navrhovateli, který si stál za svým a neochvějně čelil kobercovému bombardování ze strany rozzuřeného davu, změna je dnes v našem stromu. Osobně bych se po několika zprávách na tomto vlákně otočil na pětníku a zmizel.

Tím se dostávám k tomu, proč nejsem členem skupiny hackerů:

Odhlásil jsem se z ní před několika lety, nezvládal jsem totiž každodenní kvanta e-mailů. Od té doby jsem se ze stejného důvodu stáhl z několika dalších mailing listů.

A přesto mi neustále chodí spousta mailů. Mnoho z nich jsem pomocí filtrů přesměroval do /dev/null: Lidé jako [vypuštěno] se tak nikdy nedostanou na mou obrazovku, nebudu potvrzovat změny v jazycích, kterým stejně nerozumím, potvrzovat změny na portech a tak dále. Všechny tyhle a mnohé další věci teď mizí někam pryč, aniž bych o tom vůbec věděl.

Navzdory těmto skartovacím nástrojům, kterými je má e-mailová schránka vybavena, dostávám neustále příliš mnoho e-mailů.

Odtud se do celého tohoto obrázku dostala ta zelenější tráva:

Přál bych si, abychom na našich mailing listech zredukovali zbytečný šum a umožnili lidem sem tam postavit nějaký ten přístřešek pro kola; a je mi skutečně jedno, jakou barvou ho natřou.

Prvním přáním je, abychom e-maily používali zdvořile, citlivě a inteligentně.

Kdybych dokázal ve stručnosti a přesně definovat soubor kritérií, kdy bychom měli a kdy bychom neměli odpovídat na e-mail, s nimiž by navíc každý souhlasil a dodržoval je, byl bych šťastný; jsem však natolik rozumný, že se o to ani nepokusím.

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

Rád bych zde navrhl několik rozbalovacích oken, která by – k mému potěšení – používaly e-mailové programy ve chvíli, kdy se někdo rozhodne zaslat či odpovědět na e-mail prostřednictvím mailing listu, do něhož by chtěl připsat i mé jméno:

```
+-------------------------------------------------------------+
| Váš e-mail bude rozeslán stovkám tisíců lidí, kteří jej    |
| budou muset alespoň 10 sekund číst, než se vůbec rozhodnou,|
| zda je zajímavý či nikoli. Na čtení tohoto e-mailu se tak  |
| vynaloží nejméně dva člověko-týdny. Mnoho příjemců bude    |
| muset za download tohoto e-mailu platit.                   |
|                                                            |
| Jste skutečně přesvědčeni, že je váš e-mail natolik        |
| důležitý, abyste jím obtěžovali všechny zde uvedené lidi?  |
|                                                            |
|          [ANO]        [UPRAVIT]        [ZRUŠIT]           |
+-------------------------------------------------------------+

+-------------------------------------------------------------+
| Upozornění: Ještě jste si nepřečetli všechny e-maily       |
| v tomto vláknu. Někdo jiný už mohl říci k tématu úplně     |
| to samé, co chcete právě odpovědět vy. Přečtěte si prosím  |
| důkladně celé vlákno, než odpovíte na některý e-mail v něm |
| obsažený.                                                  |
|                                                            |
|                        [ZRUŠIT]                            |
+-------------------------------------------------------------+

+-------------------------------------------------------------+
| Upozornění: Váš e-mailový program vám ještě neukázal       |
| celý tento vzkaz. Z toho logicky vyplývá, že jste jej      |
| ještě nemohli celý přečíst a řádně mu porozumět.           |
|                                                            |
| Je nezdvořilé odpovídat na e-mail, dokud jste              |
| jej celý nepřečetli a nezamysleli se nad ním.              |
|                                                            |
| Časovač vychladnutí vám nyní zabrání odpovídat             |
| na jakýkoli email v tomto vlákně po dobu jedné hodiny.     |
|                                                            |
|                       [Zrušit]                             |
+-------------------------------------------------------------+
```

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

```
+-----------------------------------------------------------+
| Tento e-mail jste sepsali rychlostí přesahující N.NN znaků |
| za sekundu. Je celkem nemožné psát a přemýšlet při         |
| rychlosti přesahující A.AA znaků za sekundu, a vaše odpověď|
| je tak pravděpodobně nesouvislá, špatně promyšlená a/nebo  |
| emocionální.                                               |
|                                                            |
| Časovač vychladnutí vám nyní zabrání poslat                |
| jakýkoli email po dobu jedné hodiny.                       |
|                                                            |
|                       [Zrušit]                             |
+-----------------------------------------------------------+
```

Druhá část mého přání je poněkud emocionálnější. Kapacity, které jsme na zkrocení divokého požáru ve vláknu sleep(1) vyhradili, se přes své mnohaleté zkušenosti s tímto projektem ani neobtěžovaly tento drobný úkol splnit, tak proč jsou teď tak popuzeni, když se o to postará někdo jiný, navíc jejich služebně mladší?

To bych si přál vědět.

Vím však určitě, že věcná argumentace „reakční konzervatismus“ tohoto kalibru sotva zastaví. Možná, že jsou frustrováni faktem, že za poslední dobu nepřispěli k projektu ničím hmatatelným, nebo se může jednat o odpudivý případ „starých bručounů, kteří vědí nejlépe, jak se mají mladí chovat.“

Tak či tak, celá situace nepřináší projektu žádný užitek, ale nenapadá mě, jak ji ukončit. Jedině snad, že bychom měli přestat živit strašidla skrývající se v mailing listech: Ignorujte je, neodpovídejte jim, zapomeňte na ně.

Doufám, že ve FreeBSD získáme silnější a širší základnu přispěvatelů a že společně zabráníme starým bručounům a [vypuštěno] celého světa, aby tyto přispěvatele sežvýkali, vyplivli a zastrašili ještě dříve, než se nohama dotknou země.

Těm, kteří vyčkávali, protože je od přispění odradila strašidla: Nezbývá mi, než se omluvit a vyzvat vás, abyste se pokusili do projektu zapojit, takhle jsem si prostředí projektu určitě nepředstavoval.

Poul-Henning

C. Měl bych se starat o to, jakou barvu má přístřešek pro kola?

D. Vzorové pokyny pro hlášení chyb (bugů)

D. Vzorové pokyny pro hlášení chyb (bugů)

Následuje nepatrně upravená kopie on-line instrukcí projektu Subversion, určených novým uživatelům, které je seznamují s tím, jak správně hlásit chybu. Viz část **Ke všem uživatelům se chovejte jako k potenciálním dobrovolníkům** v kapitole **8. Řízení dobrovolníků**, kde je vysvětleno, proč je důležité, aby každý projekt měl takovéto instrukce. Originál dokumentu naleznete zde **http://svn.collab.net/repos/svn/trunk/www/bugs.html.**

```
                    Hlášení chyb v projektu Subversion

Tento dokument Vás informuje o tom, jak a kde hlásit chyby. (Neuvádí se
zde kompletní seznam nevyřešených chyb - ten naleznete zde.)

Kam nahlásit chybu
------------------

    * Pokud je chyba přímo v Subversion, zašlete e-mail na adresu
      users@subversion.tigris.org. Jakmile bude takto nahlášená chyba
      potvrzena jako chyba skutečná, někdo, například vy, ji může zadat
      do issue trackeru (záznamníku problémů). (Pokud jste si skutečně
      jistí, že se jedná o chybu, neváhejte a zašlete ji přímo do našeho
      vývojářského mailing listu, dev@subversion.tigris.org. Pokud si však jisti
      nejste, bude lépe ji nejprve zaslat na users@; někdo vám tam bude
      schopen říci, zda je chování, které hlásíte, standardní či nikoli.)

    * Pokud je tato chyba v knihovně APR, nahlaste ji prosím na oba tyto
      mailing listy: dev@apr.apache.org, dev@subversion.tigris.org.

    * Pokud je chyba v knihovně Neon HTTP, nahlaste ji prosím na:
      neon@webdav.org, dev@subversion.tigris.org.

    * Pokud je chyba v Apache HTTPD 2.0, nahlaste ji prosím na oba tyto
      mailing listy: dev@httpd.apache.org, dev@subversion.tigris.org.
      Mailing list vývojářů Apache httpd je velice exponovaný,
      váš vzkaz s bug reportem tak může být přehlédnut. Bug report můžete
      také zaevidovat na adrese: http://httpd.apache.org/bug_report.html.

    * Pokud máte blechu v kožichu, hlaďte ji, ať je potichu.
```

```
Jak nahlásit chybu
------------------

Nejprve se ujistěte, zda se opravdu jedná o chybu. Pokud se Subversion chová
jinak, než byste čekali, podívejte se do dokumentace a archívů mailing
listu, abyste se ujistili, zda má skutečně fungovat tak, jak očekáváte.
Samozřejmě, pokud je chyba průkazná a očividná, když vám například
Subversion zničí data a způsobí doutnání monitoru, můžete svému úsudku
věřit. Pokud si však nejste jisti, zeptejte se nejprve uživatelů na mailing
listu users@subversion.tigris.org nebo se zeptejte na IRC
(irc.freenode.net), kanál #svn.

Když máte potvrzeno, že se jedná o chybu, je nejdůležitější podat její
jednoduchý popis a návod na reprodukci. Pokud například chyba, tak jak jste ji
objevili, figuruje v pěti souborech z deseti commitů, pokuste se ji navodit
v jednom souboru na jeden commit. Čím jednoduší je návod, tím
úspěšnější bude i vývojář při reprodukování chyby a její opravě.

Když píšete návod na reprodukci, nepište romány o tom, co jste udělali, aby
k chybě došlo. Naopak, podejte doslovný přepis přesného sledu příkazů, které
jste spustili, a jejich výstupů. Za tímto účelem použijte příkazy zkopírovat
a vložit. Pokud chyba zasahuje do souborů, rozhodně uveďte jejich názvy
i obsah, pokud si myslíte, že by to mohlo být relevantní. Nejlepší je
zabalit svůj reprodukční návod jako skript, to nám pomůže asi nejvíce.

Krátká kontrola příčetnosti: máte spuštěnu nejnovější verzi
Subversion, ano? :-) Možná už byla chyba opravena; porovnejte svůj
reprodukční návod s poslední verzí vývojového stromu Subversion.

Kromě reprodukčního návodu potřebujeme také kompletní popis prostředí,
v němž chybu reprodukujete. To znamená:

    * Váš operační systém
    * Číslo release a/nebo revize Subversion
    * Kompilační a konfigurační volby, jimiž jste Subversion sestavili
    * Veškeré osobní úpravy, které jste na Subversion provedli
    * Verze Berkeley DB, s níž Subversion spouštíte, pokud nějakou máte
    * Všechny ostatní informace, které by mohly být relevantní. Raději
      zašlete příliš mnoho než příliš málo informací.

Teprve po splnění těchto úkolů jste připraveni sepsat zprávu. Nejprve jasně
popište, o jakou chybu se jedná. Tedy popište, jaké chování jste od
```

Subversion očekávali a jak se ve skutečnosti chová. Vám osobně se může chyba jevit jako zcela zjevná, někomu jinému však nikoli; nebudeme si raději dávat hádanky. Poté popište prostředí a návod na reprodukci. Pokud chcete uvést svou teorii o příčině chyby, nebo dokonce záplatu k její opravě, výborně - vizte http://subversion.apache.org/docs/community-guide/#patches - zde jsou uvedeny pokyny k zasílání záplat.

Všechny tyto informace zašlete na adresu dev@subversion.tigris.org; případně, pokud jste tak již učinili a byli jste vyzváni k evidování celého problému, přejděte na Issue Tracker a řiďte se zde uvedenými pokyny.

Děkujeme. Víme, že evidování účinného bug reportu je pracovně náročné, ale dobře vypracovaný bug report může vývojářům ušetřit hodiny práce, a zvýšit tak pravděpodobnost opravy dané chyby.

E. Copyright

E. Copyright

Toto dílo podléhá licenci Creative Commons
Attribution-ShareAlike License. Kopii této licence naleznete na adrese
http://creativecommons.org/licenses/by-sa/3.0/, případně o ni můžete
zažádat písemně na adrese Creative Commons, 559 Nathan Abbott Way,
Stanford, California 94305, USA. Souhrnný obsah licence je uveden níže,
po něm následuje její úplné znění. Pokud chcete některou část nebo
celé toto dílo v souladu s různými podmínkami distribuovat, kontaktujte
prosím jeho autora, Karla Fogela <kfogel@red-bean.com>.

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-

Bezplatně můžete:

* Šířit - kopírovat, distribuovat a sdělovat dílo veřejnosti
* Upravovat - toto dílo

Podle následujících podmínek:

* Uveďte autora – Máte povinnost uvést údaje o autorovi a tomto díle způsobem, který stanovil autor nebo poskytovatel licence (ne však tak, aby vznikl dojem, že podporují vás nebo způsob, jakým dílo užíváte).

* Zachovejte licenci – Pokud toto dílo jakkoliv upravíte nebo použijete ve svém díle, máte povinnost výsledek své práce šířit pod stejnou nebo slučitelnou licencí.

* Při jakémkoli dalším používání nebo distribuci musíte ostatní seznámit s licenčními podmínkami, které se k tomuto dílu vztahují. Nejlepší bude poskytnout jim odkaz na tuto webovou stránku.

* Jakákoli z výše uvedených podmínek může být prominuta na základě povolení majitele autorských práv.

* Tato licence nikterak nezmenšuje či neomezuje morální práva autora.

-*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*- -*-
Právní kodex Creative Commons: Attribution-ShareAlike 3.0 Česko
(Uveďte autora-Zachovejte licenci 3.0 Česko)

CREATIVE COMMONS CORPORATION („CREATIVE COMMONS“) NENÍ ADVOKÁTNÍ KANCELÁŘ ANI NEPOSKYTUJE PRÁVNÍ SLUŽBY. POSKYTOVÁNÍM NÁSLEDUJÍCÍHO TEXTU LICENČNÍHO UJEDNÁNÍ NEVZNIKÁ MANDÁTNÍ NEBO OBDOBNÝ VZTAH. CREATIVE COMMONS POSKYTUJE TEXT UJEDNÁNÍ TAKOVÝ, JAKÝ JE. CREATIVE COMMONS NEPOSKYTUJE ŽÁDNÉ ZÁRUKY ZA OBSAH TOHOTO TEXTU A NEODPOVÍDÁ ZA ŠKODY, KTERÉ VZNIKNOU JEHO UŽITÍM, NAD MINIMÁLNÍ ROZSAH STANOVENÝ PLATNOU PRÁVNÍ ÚPRAVOU.

Licenční ujednání

DÍLO (JAK JE DEFINOVÁNO NÍŽE) JE POSKYTOVÁNO ZA PODMÍNEK TÉTO CREATIVE COMMONS PUBLIC LICENSE (DÁLE JEN „CCPL“, „LICENČNÍ UJEDNÁNÍ“ NEBO „UJEDNÁNÍ“). DÍLO JE CHRÁNĚNO PLATNÝMI PŘEDPISY UPRAVUJÍCÍMI PRÁVO AUTORSKÉ. JAKÉKOLI UŽITÍ DÍLA, KTERÉ NENÍ V SOULADU S TĚMITO PŘEDPISY NEBO S TÍMTO LICENČNÍM UJEDNÁNÍM, JE ZAKÁZÁNO.

UŽITÍM DÍLA V SOULADU S TÍMTO LICENČNÍM UJEDNÁNÍM SE NABYVATEL ZAVAZUJE DODRŽOVAT PODMÍNKY TOHOTO UJEDNÁNÍ A STÁVÁ SE JAKO NABYVATEL STRANOU LICENČNÍ SMLOUVY V ROZSAHU, V JAKÉM PODMÍNKY TOHOTO UJEDNÁNÍ NABYVATELE SMLUVNĚ ZAVAZUJÍ. POSKYTOVATEL POSKYTUJE NABYVATELI LICENCI K DÍLU JEN POKUD NABYVATEL BEZPODMÍNEČNĚ AKCEPTUJE PODMÍNKY TOHOTO UJEDNÁNÍ.

1. Definice

a. Pojem **„dílo“** označuje pro účely tohoto ujednání autorské dílo nebo jiný nehmotný statek chráněný autorským zákonem, pokud příslušný právní řád jeho ochranu též uznává. Autorskými díly mohou být mimo jiné díla literární, výtvarná, hudební, audiovizuální, vědecká, fotografie nebo počítačové programy. Jinými nehmotnými statky jsou zejména umělecké výkony výkonných umělců, zvukové nebo zvukově obrazové záznamy, televizní a rozhlasové vysílání. Za díla jsou považovány též databáze. Pojem **„Dílo“** popřípadě **„DÍLO“** označuje konkrétní dílo, ke kterému poskytovatel poskytuje nabyvateli licenci za podmínek uvedených v tomto ujednání.

b. Pojem **„souborné dílo“** označuje pro účely tohoto ujednání soubor nezávislých děl nebo jiných prvků, který je jako celek dílem, a do něhož je celé Dílo v nezměněné, tj. neupravené podobě zařazeno. Souborným dílem může být zejména časopis nebo jiné periodikum, sborník, encyklopedie, antologie, pásmo nebo výstava. Zařazení Díla do souborného díla se nepovažuje za jeho úpravu. Pokud je to výslovně uvedeno, považuje se za souborné dílo též soubor, který místo Díla obsahuje upravené Dílo.

c. Pojem **„upravené Dílo“** označuje pro účely tohoto ujednání výsledek jakékoliv úpravy Díla, kterou může být zejména zpracování Díla nebo zpracování Díla s jinými díly, doplnění Díla nebo jiné změny Díla. Upraveným Dílem může být mimo jiné jeho překlad, dramatizace, zhudebnění. Za upravené Dílo se považuje i jeho spojení s dalším dílem či prvky (např. užití hudebního díla jako doprovodu), ale ne jeho pouhé zařazení do souborného díla.

d. Pojem **„autor“** označuje pro účely tohoto ujednání osobu nebo osoby, které Dílo vytvořily.

e. Pojem **„poskytovatel“** označuje pro účely tohoto ujednání autora nebo jinou fyzickou nebo právnickou osobu, která je oprávněna poskytnout licenci k užití Díla za podmínek uvedených v tomto ujednání.

f. Pojem **„nabyvatel“** označuje pro účely tohoto ujednání fyzickou nebo právnickou osobu, která užívá Dílo v souladu s tímto ujednáním a která neporušila ve vztahu k Dílu podmínky tohoto ujednání, ledaže získala od poskytovatele výslovný souhlas vykonávat práva k Dílu na základě tohoto ujednání i přes předchozí porušení jeho podmínek.

g. Pojem **„rozmnožování“** označuje pro účely tohoto ujednání zhotovování rozmnoženin díla, a to jakýmikoli prostředky. Rozmnoženiny mohou být mimo jiné tiskové, fotografické, zvukové, obrazové, nebo zvukově-obrazové a mohou mít též elektronickou podobu, zahrnující vyjádření analogové i digitální. Rozmnožováním je též zhotovení rozmnoženiny nezbytné k zavedení, uložení, zobrazení, provoz a přenos počítačového programu a vytěžování obsahu databáze.

h. Pojem **„rozšiřování“** označuje pro účely tohoto ujednání zpřístupňování originálu díla nebo jeho rozmnoženiny v hmotné podobě prodejem nebo jiným převodem vlastnického práva. Za rozšiřování díla se považuje také jeho vystavování, pronájem a půjčování.

i. Pojem **„sdělování veřejnosti“** označuje pro účely tohoto ujednání zpřístupňování díla v nehmotné podobě. Sdělováním veřejnosti se mimo jiné rozumí veřejné provozování díla nebo jeho přenos, vysílání rozhlasem nebo televizí a zpřístupňování díla veřejnosti prostřednictvím počítačové nebo jiné sítě, a to způsobem, že kdokoli může mít k němu přístup na místě a v čase podle své volby. Sdělováním veřejnosti je též zužitkování obsahu databáze.

j. Pojem **„licenční prvky“** označuje pro účely tohoto ujednání charakteristické prvky této licence, které stanovil poskytovatel a které jsou vyjádřeny v jejím označení: „Uveďte autora“ a „Zachovejte licenci“.

k. Pojem **„licenční ujednání kompatibilní s Creative Commons“** označuje pro účely tohoto ujednání licenční ujednání uvedené na adrese **http://creativecommons.org/compatiblelicenses/**, které Creative Commons uznala jako ve své podstatě ekvivalentní s tímto ujednáním, neboť splňuje minimálně následující kritéria:

   i. obsahuje licenční podmínky, které mají stejný účel a stejné důsledky jako licenční prvky tohoto ujednání a
   ii. výslovně umožňuje poskytování upraveného díla za podmínek tohoto ujednání nebo licenčního ujednání Creative Commons Unported se shodnými licenčními prvky, popřípadě licenčního ujednání určeného pro právní řád jiného státu se shodnými licenčními prvky.

2. Výjimky a omezení ochrany práv k Dílu

Toto ujednání neomezuje, nezužuje ani jinak nelimituje volná užití Díla, užití Díla na základě zákonné licence, vyčerpání práv při prvním převodu vlastnictví k originálu nebo rozmnoženině Díla v hmotné podobě nebo jiná zákonná omezení práv k Dílu.

3. Poskytnutí licence

Za podmínek stanovených tímto ujednáním poskytuje poskytovatel nabyvateli bezúplatnou, množstevně a místně neomezenou, nevýhradní a časově neomezenou (na celou dobu trvání práv k Dílu) licenci k Dílu:

a. oprávnění rozmnožovat Dílo, zahrnovat Dílo do souborných děl a jako součást souborných děl Dílo dále rozmnožovat,

b. oprávnění upravovat Dílo a upravené Dílo rozmnožovat, zařazovat do souborných děl a jako součást souborných děl dále rozmnožovat, pokud je z upraveného Díla nebo jeho označení zřejmé, že Dílo bylo změněno nebo jinak upraveno,

c. oprávnění Dílo samostatně nebo jako součást souborného díla rozšiřovat a sdělovat veřejnosti,
d. oprávnění rozšiřovat a sdělovat veřejnosti upravené Dílo.

e. Práva na odměnu za užití Díla podle tohoto ujednání jsou upravena následovně:

   i. Tímto ujednáním nejsou dotčena práva na odměnu za užití Díla, která poskytovatel nemůže neuplatnit nebo se jich vzdát, zejména práva povinně kolektivně spravovaná.
   ii. Ve všech ostatních případech se poskytovatel zavazuje svá práva na odměnu za užití Díla podle tohoto ujednání neuplatnit nebo se jich tímto vzdává.

Nabyvatel je oprávněn Dílo užívat výše uvedenými způsoby na všech nosičích a ve všech formátech, není-li takové užití omezeno zákonem. Oprávnění nabyvatele se vztahuje i na provádění technických úprav nezbytných k tomu, aby Dílo bylo dovoleným způsobem užito na jiném nosiči nebo v jiném formátu. Veškerá práva k Dílu, která nejsou výslovně poskytnuta touto licencí, zůstávají vyhrazena. Nabyvatel není povinen poskytnutou licenci využít.

Je-li součástí poskytované licence i zvláštní právo pořizovatele k jím pořízené databázi, poskytovatel se takového práva k Dílu v celém rozsahu vzdává.

4. Omezení licence

a. Nabyvatel je oprávněn rozšiřovat Dílo nebo ho sdělovat veřejnosti pouze za podmínek stanovených tímto ujednáním. Nabyvatel je při tom vždy povinen k Dílu připojit text tohoto ujednání nebo odkaz na něj ve formátu Uniform Resource Identifier (dále jen „URI"). Nabyvatel není oprávněn omezovat užití Díla nad rámec stanovený tímto ujednáním. Nabyvatel není oprávněn poskytovat podlicenci k Dílu. Při rozšiřování Díla nebo jeho sdělování veřejnosti je nabyvatel povinen zachovat beze změny všechny odkazy na toto ujednání a případná ustanovení o odpovědnosti vztahující se k Dílu. Při rozšiřování Díla nebo jeho sdělování veřejnosti nesmí nabyvatel použít žádné technické prostředky ochrany, které by omezovaly oprávnění dalších osob v užití Díla v souladu s tímto ujednáním. Ustanovení tohoto čl. 4 písm. a) se vztahují na Dílo i tehdy, je-li zahrnuto do souborného díla. Licence poskytovaná podle tohoto ujednání se však nemusí vztahovat na souborné dílo jako celek nebo na jeho ostatní části. Pokud nabyvatel vytvoří souborné dílo je povinen na žádost poskytovatele odstranit ze souborného díla údaje uvedené v ustanovení čl. 4 písm. c), je-li to fakticky možné. Pokud nabyvatel vytvoří upravené Dílo je povinen na žádost poskytovatele odstranit z upraveného Díla údaje uvedené v ustanovení čl. 4 písm. c), je-li to fakticky možné.

b. Nabyvatel je oprávněn rozšiřovat nebo sdělovat veřejnosti upravené Dílo výhradně za podmínek:

i. tohoto licenčního ujednání,
ii. pozdější verze tohoto licenčního ujednání se shodnými licenčními prvky,
iii. verze 3.0 nebo vyšší licenčního ujednání Creative Commons určeného pro právní řád jiného státu, jež obsahuje shodné licenční prvky,
iv. verze 3.0 nebo vyšší licenčního ujednání Creative Commons Unported se shodnými licenčními prvky, nebo
v. licenčního ujednání kompatibilního s Creative Commons.

Pokud nabyvatel poskytuje k upravenému Dílu licenci uvedenou v bodě (v), je povinen dodržovat podmínky této licence.

Pokud nabyvatel poskytuje jednu z licencí uvedenou v bodě (i)–(iv) (dále jen „dovolená licenční ujednání"), je povinen dodržovat podmínky takové licence, zejména: Je při rozšiřování upraveného Díla a jeho sdělování veřejnosti vždy povinen k upravenému Dílu připojit text dovoleného licenčního ujednání nebo odkaz na něj ve formátu URI. Není oprávněn omezovat užití upraveného Díla nad rámec stanovený tímto ujednáním. Při rozšiřování upraveného Díla nebo jeho sdělování veřejnosti je nabyvatel povinen zachovat beze změny všechny odkazy na dovolené ujednání a případná ustanovení o odpovědnosti. Při rozšiřování upraveného Díla nebo jeho sdělování veřejnosti nesmí použít žádné technické prostředky ochrany, které by omezovaly oprávnění dalších osob v užití upraveného Díla v souladu s dovoleným licenčním ujednáním. Ustanovení tohoto čl. 4 písm. b) se vztahují na upravené Dílo i tehdy, je-li zahrnuto do souborného díla. Licence poskytovaná podle dovoleného licenčního ujednání se však nemusí vztahovat na takové souborné dílo jako celek nebo na jeho ostatní části.

c. Při rozšiřování Díla, upraveného Díla nebo souborného díla nebo při jejich sdělování veřejnosti je nabyvatel povinen, pokud nebyl v souladu s ustanovením čl. 4 písm. a) požádán o opak, připojit beze změny všechny copyrightové doložky a je povinen způsobem odpovídajícím danému nosiči a v přiměřené formě uvést následující údaje, pokud existují a jsou mu známy:

i. jméno případně pseudonym autora nebo jména či označení jiných osob, které autor nebo poskytovatel uvedl v copyrightové doložce k Dílu, v podmínkách užití Díla nebo které označil jiným přiměřeným způsobem (dále jen „uvedené osoby"), pozdější verze tohoto licenčního ujednání se shodnými licenčními prvky,
ii. název Díla,
iii. odkaz ve formátu URI, který poskytovatel k Dílu připojil, pokud odkazuje na copyrightovou doložku k Dílu nebo na licenční podmínky a
iv. pokud se jedná o upravené Dílo, též údaje o Díle a o způsobu, kterým bylo upraveno, v souladu s ustanovením čl. 3 písm. b).

Údaje uvedené v ustanovení tohoto čl. 4 písm. c) má nabyvatel povinnost uvést jakýmkoli přiměřeným způsobem. V případě upraveného Díla nebo souborného díla se považuje za přiměřené, aby údaje vztahující se k Dílu byly uvedeny spolu s obdobnými údaji o ostatních dílech, která byla zahrnuta do souborného díla nebo využita při vzniku upraveného Díla, a to způsobem nesnižujícím jejich hodnotu ve srovnání s obdobnými údaji o ostatních uvedených dílech a ostatních uvedených osobách. Údaje uvedené v ustanovení tohoto čl. 4 písm. c) je nabyvatel oprávněn použít pouze pro označení Díla v souvislosti s užitím Díla v souladu s tímto ujednáním. Bez předchozího písemného souhlasu uvedených osob není nabyvatel oprávněn uvádět údaje o Díle způsobem, který by přímo či nepřímo vyvolal dojem účasti nebo jiné formy podpory ze strany uvedených osob.

d. Omezení uvedená v čl. 4 písm. a) až c) se nevztahují na ty části Díla, na které se vztahuje definice díla uvedená v čl. 1 písm. a) pouze z důvodu ochrany zvláštních práv pořizovatele databáze.

e. Tímto ujednáním nejsou dotčena osobnostní práva autora, pokud příslušný právní řád jejich ochranu uznává. Zejména si nikdo nesmí osobovat autorství k Dílu a Dílo smí být užito jen způsobem nesnižujícím jeho hodnotu. Za zásah do osobnostních práv autora se nepovažuje jednání v souladu s podmínkami stanovenými tímto ujednáním.

5. Odpovědnost za vady

NEBYLA-LI MEZI POSKYTOVATELEM A NABYVATELEM UZAVŘENA PÍSEMNÁ DOHODA UPRAVUJÍCÍ ODPOVĚDNOST POSKYTOVATELE ZA VADY DÍLA, POSKYTUJE POSKYTOVATEL DÍLO TAKOVÉ, JAKÉ JE. POSKYTOVATEL NEPROHLAŠUJE, ŽE DÍLO MÁ URČITÉ VLASTNOSTI A NEPOSKYTUJE K DÍLU ZÁRUKY, COŽ NABYVATEL BERE NA VĚDOMÍ.

6. Odpovědnost za škodu

POSKYTOVATEL NEODPOVÍDÁ ZA ŽÁDNOU ŠKODU NAD MINIMÁLNÍ ROZSAH STANOVENÝ PLATNOU PRÁVNÍ ÚPRAVOU, COŽ NABYVATEL BERE NA VĚDOMÍ.

7. Ukončení licence

a. Nabyvateli zaniká licence k Dílu podle tohoto ujednání okamžikem, kdy nabyvatel poruší podmínky tohoto ujednání. Tím nejsou dotčeny licence k upravenému Dílu nebo soubornému dílu, které nabyvatel poskytl nebo poskytne v souladu s tímto ujednáním dalším osobám, pokud tyto osoby podmínky příslušné licence dodržují. Články 1, 2, 5, 6, 7 a 8 tohoto ujednání zůstávají v účinnosti i po zániku oprávnění k užití Díla podle tohoto odstavce.

b. Nedojde-li k zániku licence k Dílu podle odstavce a), je licence k Dílu časově neomezená (na celou dobu trvání práv k Dílu). Poskytovatel je oprávněn současně poskytovat k Dílu jiné licence nebo může přestat Dílo šířit, pokud tím nebude dotčena licence k Dílu poskytnutá nabyvateli ani další licence poskytnuté v souladu s tímto ujednáním, a pokud zůstane, vyjma případu uvedeného v předchozím odstavci, v plném rozsahu platná a účinná licence, kterou poskytovatel nabyvateli na základě tohoto ujednání poskytl.

8. Závěrečná ustanovení

a. Když nabyvatel rozšiřuje nebo sděluje veřejnosti Dílo nebo souborné dílo, poskytuje licenci k Dílu dalším osobám přímo poskytovatel, a to za stejných podmínek a ve stejném rozsahu, v jakém získal licenci k Dílu nabyvatel na základě tohoto ujednání.

b. Když nabyvatel rozšiřuje nebo sděluje veřejnosti upravené Dílo, poskytuje licenci k Dílu dalším osobám přímo poskytovatel, a to za stejných podmínek a ve stejném rozsahu, v jakém získal licenci k Dílu nabyvatel na základě tohoto ujednání.

c. Pokud se některé ustanovení tohoto ujednání stane neplatným nebo neúčinným, nemá tato neplatnost nebo neúčinnost vliv na platnost a účinnost zbývajících ustanovení.

d. Bez písemného souhlasu druhé smluvní strany není smluvní strana oprávněna vyloučit nebo změnit žádné ustanovení tohoto ujednání.

e. Smluvní strany prohlašují, že toto ujednání tvoří úplnou dohodu o podmínkách poskytnutí licence k Dílu. Toto ujednání lze měnit pouze písemnou dohodou smluvních stran.

f. Pokud se smluvní strany nedohodnou jinak, řídí se právní vztahy podle tohoto ujednání právem České republiky.

Upozornění Creative Commons

Creative Commons není stranou licenční smlouvy a neposkytuje k Dílu žádné záruky. Creative Commos v žádném případě neodpovídá nabyvateli nebo třetím osobám za škodu nebo jinou újmu, kterou utrpěli v souvislosti s tímto ujednáním. Předchozí dvě věty se nevztahují na případ, kdy Creative Commons sebe jako poskytovatele podle tohoto ujednání výslovně uvede.

Ochrannou známku „Creative Commons“ a jiné ochranné známky nebo loga Creative Commons je možné použít pouze pro potřeby označení, že Dílo je poskytováno veřejnosti pod licencí CCPL. Jakékoli jiné užití ochranné známky „Creative Commons“ nebo jiné ochranné známky nebo loga Creative Commons vyžaduje předchozí písemný souhlas Creative Commons. Jejich užití se řídí aktuálním zněním pokynů pro užívání ochranných známek Creative Commons, které je k dispozici na internetových stránkách Creative Commons nebo na vyžádání. Uvedená pravidla pro užití ochranných známek nejsou součástí této licence.

Creative Commons je možno kontaktovat na adrese **http://creativecommons.org/.**

Karl Fogel

TVORBA OPEN SOURCE SOFTWARU
Jak řídit úspěšný projekt svobodného softwaru

Přeloženo z anglického originálu knihy *Producing Open Source Software*.
1. vydání, 2005
Vydáno nakladatelstvím O'Reilly Media (*oreilly.com*), USA.

Vydavatel:
CZ.NIC, z. s. p. o.
Americká 23, 120 00 Praha 2
Edice CZ.NIC
www.nic.cz

1. vydání, Praha 2012
Kniha vyšla jako 5. publikace v Edici CZ.NIC.

© 2005–2010 Karl Fogel
V licenci CreativeCommons Attribution-ShareAlike (3.0).

ISBN 978-80-904248-5-2 (tištěná verze, PDF)
ISBN 978-80-88168-03-4 (ve formátu EPUB)
ISBN 978-80-88168-04-1 (ve formátu MOBI)

cz: nic | SPRÁVCE DOMÉNY CZ

Edice CZ.NIC

Většina open source programů začne jako nápad v hlavě jednoho člověka nebo malé skupinky lidí. Pokud má však nápad dostatečný potenciál, může se z něj časem stát úspěšný projekt, který používají miliony lidí a vyvíjí komunita sestávající z desítek vývojářů. Kniha kterou právě držíte v rukou dává návod, jak váš open source projekt nastartovat a vést tak, aby měl co nejlepší šanci stát se právě takovým. — Autor v ní velice realisticky a s praktickým přístupem podává ucelený přehled problematiky vedení vývoje svobodného softwaru. Popisuje jak jednotlivé dílčí části managementu projektu, jako je komunikace s uživateli či správa zdrojových kódů, tak jeho postupný vývoj od volby jména a licence až po řešení problémů spojených s vývojem úspěšného projektu s celosvětovou vývojářskou a uživatelskou komunitou.

O autorovi **Karl Fogel** je velice aktivní vývojář a advokát svobodného softwaru. Pravděpodobně nejznámější je jeho role ve vývoji populárního systému pro správu verzí **Subversion**, jehož byl po řadu let hlavním vývojářem. Podílel se také na tvorbě systému pro hostování projektů **Launchpad** či editoru **Emacs** a napsal řadu esejů a článků popularizujících svodobný přístup k vývoji softwaru. — Svoje zkušenosti z vývoje a vedení open source projektů pak využil v knize **Producing Open Source Software**, jejíž český překlad právě držíte v ruce.

O edici Edice CZ.NIC je jedním z osvětových projektů správce české domény nejvyšší úrovně. Cílem tohoto projektu je vydávat odborné, ale i populární publikace spojené s internetem a jeho technologiemi. Kromě tištěných verzí vychází v této edici současně i elektronická podoba knih. Ty je možné najít na stránkách knihy.nic.cz

ISBN 978-80-904248-5-2

9 788090 424852

knihy.nic.cz