Katedrála a tržiště

Eric Raymond

(Přeložil Miloslav Nič)

Podrobně rozebírám úspěšný otevřený projekt, fetchmail. Tento projekt cíleně testoval některé překvapivé teorie o softwarovém inženýrství. Tyto teorie, inspirované historií Linuxu, jsou zasazeny do kontextu dvou různých vývojových stylů, modelu „katedrály“, kterou používá většina komerčního světa a Linuxového modelu „tržiště“. Ukazuji, že tyto modely vycházejí z protikladných představ o vývoji software. Na základě zkušeností získaných s Linuxem poté dokazuji, že „Pokud máte dostatek očí, všechny chyby jsou průhledné“, nabízím analogie s jinými sebeopravujícími systémy a článek končím diskuzí o tom, jaké implikace tato fakta mají pro budoucnost software.

Sazbu LaTeXem provedl Petr Krčmář

Obsah

1 Katedrála a tržiště

Kdo by si pomyslel ještě před pěti lety, že prvotřídní operační systém může vzniknout jakoby kouzlem z dobrovolné práce několika tisíc programátorů roztroušených po celém světě propojených pouze chapadly internetu.

Já určitě ne. Počátkem roku 1993, kdy se Linux poprvé objevil na mé radarové obrazovce, jsem se již 10 let zabýval Unixem a otevřeným vývojem. V polovině osmdesátých let jsem patřil k prvým přispěvatelům GNU, přes internet jsem zpřístupnil řadu otevřeného software, vyvinul jsem nebo se podílel na vývoji několika programů (nethack, VC a GUD módy Emacsu, xlife, atd.), které se dodnes používají. Domníval jsem se, že vím jak na to.

Linux postavil na hlavu mnohé z toho, co jsem znal. Léta jsem šířil Unixovou víru v drobné nástroje, rychlé psaní prototypů a evoluční programování. Také jsem ale věřil v existenci kritické meze složitosti, po jejímž překročení je již vyžadován více centralizovaný přístup. Věřil jsem, že nejdůležitější software (operační systémy a opravdu velké nástroje jako Emacs) musí být stavěny stejně jako katedrály, pečlivě opracovávány jednotlivými čaroději nebo malými skupinami kouzelníků pracujících v příjemném osamění a publikujícími prvé výsledky své práce až ve vhodný čas.

Vývoj ve stylu Linuse Torvaldse – publikuj brzy a často, přenes na ostatní vše co můžeš, buď otevřený až téměř k bodu promiskuity, to bylo velké překvapení. Linuxové společenství více připomíná velké hlučné tržiště různých metod a postupů než tichou, oddanou práci na stavbě katedrály. Linuxové archivy, které přijímají příspěvky od **kohokoliv**, jsou toho důkazem. Zdálo by se, že z něčeho takového se může koherentní a stabilní systém vynořit pouze sérií zázraků.

Fakt, že tento styl tržiště fungoval a fungoval dobře, to bylo velké překvapení. Když jsem se naučil v tomto světě orientovat, pracoval jsem usilovně nejen na jednotlivých projektech, ale také jsem se pokoušel porozumět, jak je možné, že se svět Linuxu nejen že nerozletí na kusy v naprostém zmatku, ale naopak neustále sílí rychlostí, kterou si umí stavitelé katedrál stěží představit.

V polovině roku 1996 se mi zdálo, že jsem systému porozuměl. Náhoda mi přihrála do cesty výborný způsob, jak své teorie otestovat formou otevřeného projektu, který jsem mohl cíleně rozběhnout ve stylu tržiště. Využil jsem šance a projekt skončil velkým úspěchem.

Ve zbytku článku vám budu vyprávět příběh tohoto projektu a na jeho základě navrhnu několik algoritmů pro efektivní práci v otevřeném projektu. Ne vše jsem se naučil až ve světě Linuxu, ale uvidíme, jak linuxový svět klade na některé věci obzvláštní důraz. Pokud se nemýlím, pomůže vám přesně pochopit, co činí z Linuxového společenství takovou studnici dobrého software a pomůže i vám zvýšit produktivitu.

2 Pošta musí projít

Od roku 1993 jsem měl na starosti technické zabezpečení malého poskytovatele bezplatného přístupu k internetu (Chester County InterLink - CCIL, West Chester, Pennsylvania). Patřím k zakladatelům CCIL a napsal jsem pro něj unikátní mnohauživatelský buletinový software, který si můžete prohlédnout přes `telnet` na adrese `locke.ccil.org`. V dnešní době podporuje téměř 3000 uživatelů na 30 linkách. Tato práce mi umožnila 24hodinový přístup na internet přes 56K linku CCIL, pro moji práce byl takový přístup nutností.

Díky tomu jsem si zvykl na okamžitý přístup k e-mailu. Z různých složitých důvodů bylo obtížné zprovoznit SLIP mezi mým domácím počítačem (`snark.thyrsus.com`) a CCIL. Když se mi to konečně podařilo, pravidelné logování a kontrola pošty přes telnet mě začalo obtěžovat. Potřeboval jsem nějak přesunout poštu na můj domácí počítač snark tak, abych byl upozorněn při jejím přijetí. Poštu jsem pak chtěl dále zpracovávat s pomocí různých programů na mém osobním počítači.

Jednoduché přesměrování pošty programem sendmail nepřipadalo v úvahu, protože můj počítač není vždy připojen k síti a nemá stálou IP adresu. Potřeboval jsem program, který by mne připojil přes SLIP a poté přenesl poštu. Věděl jsem, že takové programy existují a že většina z nich využívá jednoduchý aplikační protokol, nazývaný POP (poštovní protokol). A skutečně, náš BSD/OS operační systém v sobě obsahoval i POP3 server.

Potřeboval jsem tedy POP3 klienta. Rozhlédl jsem se po síti a jeden jsem našel. Tedy ve skutečnosti jsem jich nalezl několik. Po nějaký čas jsem používal pop-perl, ale tomu chyběla funkce, kterou jsem považoval za podstatnou, nebyl totiž schopen pozměnit adresy přicházející pošty tak, abych mohl na obdrženou poštu přímo odpovídat.

Problém spočíval v tomto: řekněme, že někdo se jménem „joe“ mi poslal e-mail. Pokud jsem si jej stáhl na snark a poté se na něj pokusil odpovědět, můj poštovní klient se snažil poslat odpověď neexistujícímu joeovi na snarku. Ruční editace adres mne ale brzy začala unavovat.

Toto byla věc, kterou mohl počítač dělat za mne. Žádný z existujících klientů ale nevěděl jak. A to nám přináší prvé ponaučení:

1. Každý dobrý program začíná tím, že řeší potíže samotného programátora.

Možná, že by to mělo být samozřejmé (existuje staré přísloví: „Nutnost je matka pokroku“), ale až příliš často vývojáři tráví svůj čas prací na programech, za které jsou placeni, ale které ani nemilují ani nepotřebují. To však neplatí ve světě Linuxu a snad právě proto je průměrná kvalita software vyvinutého v Linuxovém společenství tak vysoká.

Takže, vrhl jsem se okamžitě do horečného programování a začal psát zcela nového POP3 klienta, který by mohl soutěžit s existujícími programy? V žádném

případě! Pečlivě jsem prostudoval známé POP programy a hledal takový, který nejvíce odpovídal mým představám.

2. Protože dobří programátoři vědí co psát. Velcí vědí, co přepsat (a znovu použít)

Ačkoli o sobě netvrdím, že jsem velký programátor, snažím se ty velké napodobit. Důležitým rysem velkých programátorů je konstruktivní lenost. Vědí, že člověk není oceňován za úsilí, ale za výsledky, a že je téměř vždy snazší začít od dobrých částečných řešení než od úplného počátku.

Linus Torvalds také nezačal psát Linux od prvé řádky. Místo toho použil kód a nápady Minixu, malého operačního systému napodobujícího Unix, který byl určen pro počítače řady 386. Dnes již veškerý původní kód Minixu buďto zmizel zcela nebo byl od základu přepsán, ale na počátku vytvářel lešení podpírající batole, které se nakonec stalo Linuxem.

Veden stejnými motivy jsem prohlížel existující POP programy a hledal takové, které byly vhodné jako základ pro další vývoj.

Tradice sdílení zdrojových souborů Unixového světa byla vždy nakloněna recyklování starších kódů (proto také GNU zvolil Unix jako svůj základní operační systém). Linux dovedl tuto tradici téměř k jejímu technologickému limitu. V jeho světě jsou přístupné terabajty volných zdrojů. Jestliže zde hledáte téměř vyhovující program, máte větší šanci na úspěch než kdekoliv jinde.

Obdobně vše fungovalo i v mém případě. Spolu s tím, co jsem nalezl již dříve, můj druhý průzkum odkryl celkem 9 kandidátů: fetchpop, PopTart, getmail, gwpop, pimp, pop-perl, popc, popmail a upop. Nejdříve jsem se rozhodl pro fetchpop od Seung-Hong Oh. Vložil jsem do něho moji funkci na přepisování adres a učinil několik dalších změn, které autor později zařadil do verze 1.9.

Několik týdnů později jsem ale narazil na zdroj programu popclient, který napsal Carl Harris, a zjistil jsem, že stojím před problémem. Ačkoliv fetchpop obsahoval několik dobrých původních nápadů (například mód démon), dokázal pouze spravovat POP3 a byl programován poněkud amatérsky (Seung-Hong byl schopný, ale nezkušený programátor, a obě tyto charakteristiky se na kódu projevily). Carlův kód byl lepší, na profesionální úrovni a stabilní, ale jeho programu chybělo několik důležitých funkcí, které je celkem obtížné naprogramovat, včetně těch, které jsem programoval já sám.

Zůstat při starém nebo volit změnu? Pokud bych se rozhodl pro změnu, musel bych obětovat všechen dosud napsaný kód, získal bych ale lepší vývojovou základnu. Praktickým argumentem, který doporučoval změnu, byla podpora řady protokolů. POP3 je nejběžněji používaný protokol, ale není jediný. Fetchpop a další podobné programy neuměly POP2, RPOP nebo APOP a já již začínal přemýšlet o přidání IMAPu (nejnovějšího a nejvšestrannějšího poštovního protokolu).

Měl jsem ale i další, teoretický důvod, proč přemýšlet o změně. Bylo to něco, co jsem se naučil dlouho před Linuxem.

Počítejte s tím, že alespoň jednou budete muset vše přepsat, stejně vás to nemine (Fred Brooks, „The Mythical Man-Month“, Kapitola 11)

Nebo, řečeno jinými slovy, tak dlouho problému doopravdy neporozumíte, pokud se ho nepokusíte nějak vyřešit. Podruhé už budete možná vědět, jak na to. Takže pokud chcete najít správné řešení, buďte připraveni začít **alespoň jednou** znovu.

Dobře (řekl jsem si), úpravy fetchpopu byly mým prvním pokusem a změnil jsem programovou základnu.

Poté, co jsem 25. června 1996 poslal opravy popclienta Carl Harrisovi, zjistil jsem, že on sám již ve své podstatě ztratil o projekt zájem. Kód už byl trochu zaprášený a bylo v něm několik drobných, snadno opravitelných chyb. Já potřeboval učinit řadu změn a tak jsme se rychle dohodli na tom, že projekt převezmu.

Aniž jsem si to sám uvědomil, projekt nabral na tempu. Nepracoval jsem již pouze na drobných změnách existujícího POP klienta. Převzal jsem vývoj celého programu a v mé hlavě se rojily nápady, od kterých jsem očekával, že pravděpodobně povedou k velkým změnám.

Ve společenstvích programátorů, ve kterých je podporováno sdílení kódu, je takovýto vývoj projektu přirozený. Jednal jsem podle následujícího pravidla:

4. Pokud máte správný přístup, zajímavé problémy si vás najdou samy.

Ale přístup Carla Harrise byl ještě důležitější. Věděl že:

5. Když ztratíte zájem o program, vaší poslední povinností je předat jej schopnému nástupci.

Aniž jsme o tom museli diskutovat, Carl i já jsme věděli, že máme společný cíl, nalézt to nejlepší možné řešení. Jedinou otázkou pro oba zůstalo, jak prokázat, že jsem vhodný následovník. Jakmile se mi to podařilo, zachoval se úctyhodně, a přenechal mi své místo. Doufám, že budu jednat stejně, až přijde můj čas.

3 Je důležité mít uživatele

A tak jsem zdědil popclienta. Rovněž, a to bylo stejně důležité, jsem zdědil jeho uživatele. Mít uživatele, to je bezvadná věc, a nejen proto, že můžete prokázat, že děláte něco užitečného. Pokud si jich řádně hledíte, mohou se z nich stát vaši spolupracovníci.

Další silnou stránkou tradice Unixu, kterou Linux dotlačil až do extrému, je to, že mnoho uživatelů je zároveň programátory. Protože je zdrojový kód dostupný, mohou se i oni stát **efektivními** hackery. To může být neocenitelné pro zkrácení vývojového cyklu. Pokud se jim dostane alespoň malého povzbuzení, vaši uživatelé naleznou problémy, navrhnou řešení a pomohou vylepšit kód mnohem rychleji, než vy sám bez jejich pomoci.

6. Pokud jednáte s uživateli jako se spolupracovníky, je to ta nejsnazší cesta k rychlému vylepšení kódu a efektivnímu odstraňování chyb.

Sílu tohoto efektu je velmi snadné podcenit. Ve skutečnosti asi všichni z nás pohybujících se v otevřeném prostředí drasticky podceňovali, jak snadno je udržován systém v chodu při zvyšujícím se množství uživatelů a při jeho narůstající složitosti, dokud nám Linus Torvalds neukázal, jak na to.

Ve skutečnosti se domnívám, že Linusův nejchytřejší a nejdůležitější přínos nebylo vytvoření jádra Linuxu, ale jeho objev modelu vývoje. Když jsem tuto větu pronesl jednou v jeho přítomnosti, usmál se a tiše zopakoval něco, co často říká: „Já jsem ve skutečnosti velmi líný a rád jsem chválen za věci, které udělá někdo jiný“. Líný jako liška. Nebo, jak by mohl říct Robert Heinlein, příliš líný na to, abych mohl selhat.

Když se ohlédneme do minulosti, nalezneme zde precedent využívající metody Linuxu a to vývoj Lisp knihoven GNU Emacs a archívů Lisp kódů. Na rozdíl od katedrálního stylu práce na Emacs-C a většině ostatních FSF programů, vývoj Lisp zdrojů byl velmi flexibilní a veden uživateli. Nápady a prototypy programů byly často několikrát přepsány než dosáhly stabilní konečné podoby. A volné spojení spolupracovníků přes internet á la Linux bylo časté.

Ovšem, i můj vlastní nejúspěšnější příspěvek před fetchmailem byl s velkou pravděpodobností Emacs VC mód, spolupráce se třemi dalšími lidmi ve stylu Linuxu, z nichž dodnes osobně znám pouze Richarda Stallmana FSF, autora Emacsu a zakladatele FSF. Jednalo se o front end pro SCCS, RCS a později CVS, pro které Emacs nabízel jednoduchou kontrolu verzí. Vyvinul se z malého, jednoduchého sccs.el módu, který napsal někdo jiný. A vývoj VC byl úspěšný, protože, na rozdíl od samotného Emacsu, Lisp kód mohl velmi rychle procházet cykly publikuj/otestuj/vylepši.

4 Publikuj brzy, publikuj často

Časné a časté publikování vykonané práce patří ke kritickým částem vývojového modelu Linuxu. Většina vývojářů, včetně mne, dříve věřila, že je to špatný postup pro všechny větší projekty, protože prvé verze mají téměř určitě řadu chyb a vy nechcete pokoušet trpělivost svých uživatelů.

Tato víra posilovala obecnou důvěru ve styl stavění katedrál. Pokud bylo základní podmínkou, aby uživatelé viděli co nejméně chyb, pak je nejlepší publikovat maximálně jednou za půl roku a dřít do úmoru na odstraňování chyb v mezidobí. Emacs C byl vyvíjen tímto způsobem. Lisp knihovny ale vznikaly rozdílně. Jednalo se o aktivní archivy mimo kontrolu FSF, ve kterých jste mohli nalézt nové a experimentální verze i v obdobích, kdy samotný Emacs nebyl publikován.

Nejdůležitější z těchto archívů, Ohio state elisp archív, vykazoval ducha a mnoho rysů dnešních velkých linuxových archívů. Ale jen málo z nás tehdy opravdu důsledně přemýšlelo o tom, co děláme, nebo o tom, co samotná jejich existence naznačovala o problémech modelu stavitelů katedrál, který FSF využíval. V roce 1992 jsem se vážně pokoušel zahrnout velkou část Ohio archívu do oficiální Emacs Lisp knihovny, ale narazil jsem na řadu politických problémů a tento pokus byl z velké části neúspěšný.

Ale během roku, jak se Linux stával známým, bylo jasné, že probíhá něco jiného a mnohem zdravějšího. Linusova otevřená vývojová politika byla přesným protikladem výstavby katedrál. Sunsite a tsx-11 archívy překypovaly, byla rozšiřována řada distribucí. A to vše doprovázela předtím nevídaná frekvence publikací nových verzí jádra systému.

Linus jednal se svými uživateli jako se spolupracovníky tím nejefektivnějším způsobem

7. Publikuj brzy. Publikuj často. A naslouchej svým zákazníkům.

Linusova inovace nespočívala ani tak v tom, že něco podobného dělal (to už mělo dlouhou tradici ve světě Unixu), ale v tom, že celý proces převedl do rozměrů a složitosti samotného Linuxu. V této rané epoše (okolo roku 1991), byly běžné publikace nového jádra i **několikrát denně**. Protože si kultivoval svoji základnu spolupracovníků a využíval internet pro spolupráci více než kdokoliv jiný, tak vše pracovalo.

Ale **jak** to všechno mohlo pracovat? A bylo to něco, co jsem já sám mohl kopírovat nebo vše spočívalo na nenapodobitelné genialitě Linuse Torvaldse?

Já si to nemyslel. Je pravda, že Linus je zatraceně dobrý programátor, vždyť kolik z nás by dokázalo vytvářet celé jádro operačního systému v produkční kvalitě? Ale Linux nepředstavoval žádný ohromný koncepční skok vpřed. Linus není, tedy alespoň zatím, inovační génius designu, takový, jako je třeba Richard Stallman nebo James Gosling (NeWS, Java). Linus je spíše inženýrský génius, má šestý smysl pro obcházení chyb a slepých uliček, má opravdový cit pro na-

lezení nejméně namáhavé cesty z bodu A do bodu B. Ovšem, celkový design Linuxu v sobě nese tuto kvalitu a odráží Linusův ve své podstatě konzervativní a zjednodušující přístup.

Takže, pokud rychlé publikování a využívání internetu nebylo náhodné, ale představovalo integrální součást Linusova inženýrského génia pro nalezení nejsnazší cesty, co se snažil maximalizovat? Co dostával z celého soustrojí?

Pokud je otázka položena takto, sama nabízí odpověď. Linus udržoval své uživatele/spoluautory neustále stimulované a odměňované. Stimulované tím, že mají neustále před sebou nějakou uspokojující práci, odměňované tím, že vidí neustálá (dokonce **denní**) vylepšení své práce.

Linus se pokoušel maximalizovat počet hodin, které jsou věnovány na odstraňování chyb a na rozvoj, ačkoliv hrozilo možné riziko nestability kódu a vyhoření uživatelské báze, pokud by se nějaká vážná chyba ukázala neodstranitelnou. Linus se choval tak, jako by věřil v něco jako:

8. Pokud máte dostatečně velkou základnu spolupracovníků a testerů, téměř každý problém bude rychle charakterizován a jeho řešení bude pro někoho jednoduché.

Nebo, méně formálně „Pokud máte dostatek očí, všechny chyby jsou průhledné.“ Já to nazývám Linusovým zákonem.

Má původní formulace tohoto jevu zněla: každý problém bude pro někoho řešitelný. Linus ovšem upozornil, že člověk, který problému porozuměl a vyřešil ho, nemusí být a většinou nebývá ten, kdo jej první charakterizoval. Linus říká: „Někdo nalezne problém a někdo jiný mu rozumí. A já dokonce tvrdím, že nalézt problém je náročnější“. Důležité ale je, že nalezení i vyřešení většinou proběhne rychle.

V tom je, myslím, základní rozdíl mezi stylem stavitelů katedrál a stylem tržiště. Z pohledu stavitelů katedrál, programové chyby a vývojové problémy jsou náročné a komplikované jevy. Trvá měsíce pečlivého zkoumání několika zasvěcených, než získáte jistotu, že jste vše vyhladili. Proto dlouhé intervaly mezi publikacemi a nevyhnutelná zklamání, že dlouho očekávaný program není zdaleka dokonalý.

Z pohledu tržiště naopak předpokládáte, že chyby jsou obvykle snadno řešitelné, nebo alespoň, že se takovými rychle stanou, když se na ně zaměří tisíce dychtivých spolupracovníků, nedočkavě očekávajících další verzi. Proto publikujete často, abyste získali více oprav a jako prospěšný vedlejší efekt ztrácíte méně, pokud občas dojde k nějakému problému.

A to je vše. To opravdu stačí. Pokud je „Linusův zákon“ nepravdivý, potom jakýkoliv systém tak komplexní, jako je jádro Linuxu, na kterém se spolupodílelo tolik autorů, by se musel v nějaký okamžik zhroutit pod tíhou nepředvídaných špatných interakcí a nenalezených, hluboko ukrytých chyb. Pokud je ale pravdivý, pak postačuje na vysvětlení, proč Linux obsahuje tak málo programových chyb.

A možná by to ani nemělo být takové překvapení. Sociologové již před lety objevili, že průměrný názor velkého množství stejně dobrých (nebo špatných) pozorovatelů je podstatně spolehlivější než názor jakéhokoliv náhodně zvoleného pozorovatele. To se nazývá Delphi efektem. Zdá se, že Linus prokázal, že tento jev se týká i tvorby operačního systému, že Delphi efekt dokáže zkrotit i tak komplexní záležitost, jako je konstrukce jádra OS.

Jsem vděčen Jeffovi Dutky <dutky@wam.umd.edu>, který upozornil na to, že Linusův zákon může být přeložen jako „debugování je paralelizovatelné“. Jeff si všiml, že ačkoliv odstraňování chyb vyžaduje, aby jednotliví spolupracovníci komunikovali s nějakým koordinátorem, nevyžaduje významnou spolupráci mezi jednotlivými spolupracovníky. Proto nepadá za oběť kvadratickému zvyšování komplexity a nárokům na administraci, která v jiných případech znesnadňuje zapojení více vývojářů.

Ve skutečnosti se teoretická ztráta účinnosti způsobená duplikací práce nezdá být problémem ve světě Linuxu. Jeden z výsledků časného a častého publikování je fakt, že případné duplikace jsou brzy odhaleny.

Brooks dokonce učinil následující pozorování: „Celková cena údržby běžně používaného programu je okolo 40 % nebo i více z ceny jeho vývoje.“ Je překvapující, že tato cena je značně ovlivněna počtem uživatelů. **Více uživatelů nalezne více chyb.**

Více uživatelů nalezne více chyb, neboť s počtem uživatelů přibývá způsobů, kterým jsou programy testovány. Tento efekt je zesílen, pokud se uživatelé zároveň podílejí na vývoji programu. Každý z nich vnímá problémy při detekci chyb z jiného úhlu pohledu. Delphi efekt funguje, jak se zdá, právě díky této rozmanitosti. Ve specifickém kontextu odhalování chyb tento styl rovněž snižuje množství duplikátních činností.

Takže s přibývajícím počtem testerů se sice nemusí zmenšit komplexita té nejobtížnější chyby z pohledu vývojáře, ale stoupá šance, že někomu jinému se podaří chybu odhalit.

Linus je také obezřetný. V případě, že se **vyskytnou** vážné problémy, Linuxové jádro je číslováno takovým způsobem, že potenciální uživatel může buďto používat poslední verzi, která je označena jako stabilní, a nebo sledovat vývoj a mít nové možnosti, ale to vše za cenu rizika chyb v programu. Tuto taktiku ještě neuplatňuje většina linuxových hackerů, ale asi by měla. Dostupnost obou alternativ přispívá k atraktivitě programu.

5 Když růže není růží

Poté, co jsem studoval Linusovo chování a zformuloval teorii, proč bylo úspěšné, vědomě jsem se rozhodl tuto teorii otestovat na mém vlastním (ačkoliv zdaleka ne tak komplexním) projektu.

Ale první věc, kterou jsem učinil, byla reorganizace a zjednodušení popclienta. Carlova implementace byla velmi rozumná, ale vykazovala nepotřebnou složitost, která je běžná u mnoha C programátorů. Považoval kód za jádro a strukturu dat jako podporu pro kód. Výsledkem bylo, že kód byl nádherný, ale struktura dat náhodná a dosti ošklivá (alespoň podle vysokého standardu tohoto zkušeného LISP hackera).

K přepisování jsem měl i další důvod než vylepšení kódu a datových struktur. Potřeboval jsem program zcela pochopit. Není to žádná legrace, opravovat chyby v programu, kterému nerozumíte.

Během prvého měsíce jsem se držel základního Carlova plánu. První vážnou změnou bylo přidání protokolu IMAP. To jsem učinil tak, že jsem program rozčlenil na obecnou část a specifické metody. Toto a předchozí změny ilustrují obecný princip, kterého by se měli programátoři přidržovat, zejména pak v jazycích, jako je C.

9. Promyšlené datové struktury a průměrný kód fungují mnohem lépe než při obrácené konfiguraci

Brooks, kapitola 9: Ukaž mi svůj kód a skryj datové struktury a nebudu rozumět. Ukaž mi datové struktury a obyčejně nebudu potřebovat tvůj kód, většinou mi bude jasný.

Ve skutečnosti řekl diagramy a tabulky. Ale po 30 letech kulturních a terminologických změn to znamená téměř to samé.

V tu dobu (začátek září 1996, asi 6 týdnů od času nula) jsem začal přemýšlet o změně jména. Konec konců, už to nebyl jenom POP klient. Ale váhal jsem, protože dosud v programu nebylo nic opravdu původního. Moje verze popclienta si teprve musela vytvořit vlastní identitu.

Vše se změnilo velmi radikálně, když se fetchmail naučil přesměrovávat poštu na SMTP port. K tomu se dostanu za okamžik. Jak jsem se již zmínil, rozhodl jsem se použít svůj projekt jako test toho, co Linus Torvalds udělal dobře. Jak jsem tedy testoval? Takto:

„Publikoval jsem časně a často, téměř vždy alespoň jednou za 10 dnů, při intenzivní práci jednou denně. Zvětšoval jsem seznam testerů o každého, kdo mne kontaktoval ohledně fetchmailu. Všem jsem rozesílal vzkazy, kdykoliv jsem něco publikoval, a povzbuzoval jsem je k spoluúčasti. A naslouchal jsem jim, dával si od nich schvalovat rozhodnutí a chválil je, když mi poslali nějakou opravu nebo komentář.“

Tato jednoduchá opatření se vyplatila téměř okamžitě. Od začátku projektu

jsem dostával hlášení o chybách ve kvalitě, pro kterou by většina vývojářů byla schopna zabít, a často bylo přiloženo i dobré řešení. Dostával jsem poštu, kterou byla zábava číst, promyšlenou kritiku a rozumné návrhy na nové možnosti. Takže:

10. Pokud zacházíte s vašimi testery, jako by byli vaším nejcennějším kapitálem, oni se vaším nejcennějším kapitálem skutečně stanou.

Jedno zajímavé měřítko úspěchu fetchmailu je velikost seznamu testerů, přátel fetchmailu. V době, kdy tento text píši, má 249 členů a dva až tři členové přibývají každý týden.

Ve skutečnosti, při revizi na konci května 1997 tento seznam již ztrácí ze svého maximálního počtu téměř 300 členů ze zajímavého důvodu. Několik lidí mne požádalo, abych je odhlásil ze seznamu, protože fetchmail už pracuje tak dobře, že nepotřebují sledovat, co je nového. Možná to patří k normálnímu životnímu cyklu vyspělého projektu ve stylu tržiště.

6 Popclient se stává fetchmailem

Skutečně klíčovým okamžikem v projektu byl okamžik, když mi Harry Hochheiser poslal svůj návrh kódu pro přesměrování pošty na SMTP počítače klienta. Já si téměř okamžitě uvědomil, že spolehlivá implementace této funkce učiní ostatní způsoby doručení zastaralé.

Mnoho týdnů jsem měnil fetchmail spíše po částech a cítil jsem, že uživatelské rozhraní slouží svému účelu, ale je nepříjemné a neelegantní. Zejména záplava nastavení pro export stažené pošty do souboru nebo na standardní výstup mě obzvláště tížila, ale já nevěděl proč.

Když jsem přemýšlel o SMTP přesměrování, tak se ukazovalo, že popclient se pokoušel dělat příliš mnoho věcí. Byl navržen zároveň jako mail transport agent (MTA) a local delivery agent (MDA). S SMTP přesměrováním se z něj mohl stát čistý MTA a předávat poštu jiným programům, tak jak to dělá sendmail.

Proč si přidělávat práci s celou složitostí konfigurace MDA, když port 25 je téměř určitě přítomen na všech platformách podporujících TCP/IP?

Zde se můžeme naučit několik lekcí. Za prvé, nápad se SMTP, to byla největší odměna za to, že jsem se pokoušel napodobit Linusovy metody. Tento skvělý nápad mi poskytl jeden z uživatelů, já pouze musel pochopit jeho důsledky.

11. Skoro stejně důležité, jako mít dobré nápady, je schopnost rozeznat dobré nápady vašich uživatelů. Občas je to druhé dokonce lepší.

Je zajímavé, že pokud jste opravdu k sobě upřímní, rychle zjistíte, jak mnoho dlužíte ostatním lidem, ačkoliv okolní svět vás bude považovat za původce všeho. Vy sami pak následkem toho začnete být skromnější v pohledu na vlastní schopnosti a Linus je toho dokonalým příkladem.

Když jsem tento článek četl na konferenci o Perlu v roce 1997, Larry Wall seděl v řadě přede mnou. Když jsem se dostal k řádkům výše uvedeným, zavolal hlasem starých kazatelů „Jen to řekni, řekni bratře!“. Celé publikum se smálo, protože vědělo, že vše fungovalo stejně i v případě vynálezce Perlu.

Jen několik málo týdnů po té, co jsem projekt rozběhl, jsem začal získávat podobnou chválu nejen od uživatelů, ale i od dalších lidí, ke kterým se zpráva donesla. Schoval jsem si některé e-maily. Podívám se ně, jestli někdy budu pochybovat, že můj život má smysl.

Jsou zde ovšem ještě dvě základnější, nepolitické lekce, které jsou společné veškerému designu.

12. Často to nejzajímavější a nejoriginálnější řešení se zrodí z toho, že si uvědomíte, že vaše chápání problému bylo mylné.

Já se pokoušel vyřešit nesprávný problém tím, že jsem chtěl popclienta jako kombinovaný MTA/MDA. Fetchmail musel být znovu koncipován od základu jako čistý MTA, součást normálního SMTP.

Když při vývoji narazíte do zdi, když zjistíte, že vás zajímá jen příští oprava,

je často okamžik se zeptat ne na to, zda máte správnou odpověď, ale jestli kladete správnou otázku. Možná je třeba problém přeformulovat.

Já jsem tedy problém přeformuloval. Je jasné, že správné bylo: 1. přenést podporu SMTP do obecného driveru 2. učinit z něj výchozí mód a 3. nakonec přestat podporovat ostatní možnosti přenosu.

Nad krokem 3 jsem nějaký čas váhal, obával jsem se, že si rozzlobím své staré uživatele, kteří závisejí na těchto mechanismech. Teoreticky jim stačilo přejít na .forward soubory nebo jejich ne-sendmail alternativy, aby získali to, co potřebují. Ve skutečnosti však mohl být tento přechod komplikovaný.

Ale když jsem to učinil, zisk byl ohromný. Nejpracnější část kódu zmizela. Konfigurace byla mnohem jednodušší, už žádné starosti o MDA uživatele a jeho poštovní schránku. Žádné starosti s tím, jestli operační systém umožňuje uzamykání souborů.

Rovněž zmizelo jediné riziko ztráty pošty. Pokud jste určili za příjemce pošty soubor a disk se zaplnil, poštu jste ztratili. Se SMTP se to stát nemůže.

Zlepšila se i rychlost a významné bylo i zjednodušení manuálu. Později jsem musel doplnit některé nestandardní podmínky, ale to už bylo mnohem jednodušší.

Morální poučení? Neváhejte se rozloučit se vším, co vám nezvyšuje efektivitu. Antoine de Saint-Exupery (což byl letec a letecký konstruktér, když nepsal klasické dětské knihy) řekl:

13. Konstrukční dokonalosti není dosaženo tehdy, když už není co přidat, ale tehdy, když už nemůžete nic odebrat.

Když se Váš kód zároveň zlepšuje a stává jednodušším, potom **víte**, že jste na správné cestě.

Nastal čas pro změnu jména. Nový program vypadal mnohem více jako dvojník sendmailu než starý popclient, a tak jsem jej po dvou měsících přejmenoval na fetchmail.

7 Fetchmail dospívá

V ruce jsem měl uspořádanou a originální konstrukci, kód, o kterém jsem věděl, že funguje, neboť jsem ho používal každý den a rostoucí seznam testerů. Postupně mi docházelo, že už se nezabývám programováním něčeho jednoduchého, co může být užitečné pro několik dalších lidí. Měl jsem v rukou program, který potřebuje každý hacker s Unixem a SLIP/PPP poštovním připojením.

Se SMTP přesměrováním se fetchmail dostal daleko do čela konkurence a stal se potenciálním vládcem kategorie, takovým, který vyplní svůj prostor tak dokonale, že alternativy nejsou pouze odmítnuty, ale i téměř zapomenuty.

Myslím si, že takový výsledek skutečně nemůžete plánovat. Musí vás k němu dovést vývojový plán tak silný, že při zpětném pohledu se zdají výsledky nevyhnutelné, téměř předpověditelné. Jedinou možností, jak dostat takový nápad, je mít mnoho nápadů, nebo mít inženýrský úsudek a dovést nápady někoho jiného až za hranici toho, co si autor původní myšlenky dokázal představit.

Andrew Tanenbaum přišel s původní myšlenkou vystavět jednoduchý Unix pro 386 jako učební pomůcku. Linus Torvalds dotáhl koncept Minixu dále, než si zřejmě Andrew mohl představit a vzniklo z něj něco úžasného. Stejným způsobem (ačkoliv v menším měřítku), jsem převzal nápady Carl Harrise a Harry Hochheisera a dotáhl jsem je do konce. Vždyť většina vědy, inženýrství a softwarového vývoje není vytvářena nějakým originálním géniem, ač to hackerské bájesloví tvrdí.

Výsledky byly přesto fantastické, ve skutečnosti to byl právě takový úspěch, po kterém každý hacker touží. A to znamenalo, že si musím sám pro sebe nastavit ještě náročnější kritéria. Abych učinil fetchmail tak dobrý, jak jen jsem si byl schopen představit, musel jsem psát nejen pro své vlastní potřeby, ale rovněž zahrnout podporu, kterou jsem nepotřeboval sám, ale jiní uživatelé. A při tom všem jsem musel udržet program jednoduchý a stabilní.

První a zdaleka nejdůležitější funkcí, kterou jsem přidal poté, co jsem si vše uvědomil, byla podpora vícenásobného stahování, možnost stáhnout poštu z poštovních schránek, které obsahovaly poštu více uživatelů a poté ji distribuovat jednotlivým příjemcům.

Rozhodl jsem se tuto funkci přidat, protože někteří uživatelé po ní toužili, ale zejména proto, že mě to přinutí řádně prohlédnout současnou verzi a objevit chyby, které ve snaze napsat mnohem obecnější postup vyplavou na povrch.

Ukázalo se, že se také jednalo a výborné konstrukční rozhodnutí.

14. Jakýkoliv nástroj by měl být užitečný očekávaným způsobem, ale opravdu velké nástroje se hodí na použití, které jste nikdy neočekával.

Tak tento přepsaný fetchmail byl neočekávaně využit při správě diskuzních skupin. Další důležitou změnou, kterou ode mne požadovali moji testeři, byla podpora 8-bitového MIME. To bylo docela snadné, neboť jsem si dával pozor, abych nijak nepozměnil osmý bit. Nebylo to proto, že bych předvídal tento požadavek, spíše jsem se řídil dalším pravidlem:

15. Pokud píšete zprostředkovatelský software jakéhokoliv druhu, snažte se vlastní data nijak neměnit a nikdy se nezbavujte žádné informace, pokud vás k tomu nedonutí příjemce.

Kdybych se tímto neřídil, podpora 8-bitového MIME by byla obtížná a s chybami. Takto mi stačilo přečíst si standard a přidat něco triviální logiky při generaci hlaviček. Někteří evropští uživatelé mne přemlouvali, abych přidal omezení počtu vzkazů stáhnutelných při jednom sezení (aby mohli kontrolovat cenu svých drahých telefonních linek). Dlouho jsem tomu vzdoroval a stále s tím nejsem zcela spokojen. Pokud ale píšete pro celý svět, musíte naslouchat svým zákazníkům, to se nemění jenom proto, že nejste placen penězi.

8 Několik dalších lekcí z fetchmailu

Než se vrátíme k obecným problémům softwarového inženýrství, je třeba se poučit z několika specifických lekcí, které přinesl fetchmail.

Syntaxe rc souborů zahrnuje nepovinný parametr „noise“, které parser zcela ignoruje. Takováto anglicky znějící syntaxe je mnohem čitelnější než tradiční zhuštěné páry `parametr=hodnota`.

Toto začalo jako noční experiment, když jsem si všiml, jak mnoho deklarace rc souborů připomínají příkazový minimalistický jazyk. (Proto jsem také změnil jeden z původních parametrů popclienta „server“ na „poll“.)

Zdálo se mi, že pokud tento jazyk bude více připomínat angličtinu, bude snáze použitelný. Ačkoliv jsem přesvědčený zastánce designerské školy, která se snaží učinit z příkazů jazyk, jako v případě HTML, Emacsu a řady databází, obvykle nemám příliš rád poangličtělou syntaxi.

Tradičně programátoři dávají přednost syntaxi, která je velmi přesná a kompaktní. Toto je dědictví z doby, kdy výpočetní zdroje byly drahé, takže procházení souborů muselo být levné a co nejjednodušší. Tehdy se zdála angličtina s 50 % přebytkem slov zcela nevhodná.

To ovšem není můj důvod, proč se takovéto anglické syntaxi obvykle vyhýbám. Já ho zmiňuji pouze proto, abych jej vyvrátil. V dnešní době by už stručnost neměla být cílem kvůli sobě samé. Je důležitější, aby jazyk byl pohodlný pro uživatele, ne aby byl levný pro počítač.

Existují ovšem důvody k obezřetnosti. Jedním z nich je složitost parsingu. Nechcete její složitost zvýšit na úroveň, kdy se stává zdrojem častých chyb a začne plést i samotné uživatele. Dalším důvodem je, že pokud chcete, aby Váš jazyk zněl jako angličtina, musíte angličtinu značně pokroutit, a to natolik, že tento zpitvořený jazyk je stejně zmatečný, jako tradiční nesrozumitelná syntax. (Typickým příkladem jsou tzv. jazyky čtvrté generace a komerční jazyky pro přístup k databázím.)

Kontrolní systém fetchmailu se vyhnul těmto potížím proto, že jeho jazyková oblast je nesmírně omezená. Není to zdaleka jazyk obecný, to co říká, není vůbec složité, takže je zde jen malý prostor pro zmatek při myšlenkovém přechodu od jeho miniaturní podmnožiny angličtiny k skutečnému řídícímu jazyku. Možná nás to učí další lekci:

16. Pokud Váš jazyk není zdaleka kompletní (Turing-kompletní), syntaktický cukr může být přítelem.

Další lekce je o bezpečnosti přes utajení. Několik uživatelů mne požádalo, abych pozměnil program tak, aby ukládal hesla zašifrovaná v rc souboru a tím zabránil příležitostným čumilům v jejich náhodném odkrytí.

Já to neudělal, protože tím ve skutečnosti nezískáte žádnou ochranu. Každý, kdo získá přístupová práva ke čtení vašeho rc souboru bude moci sám spustit

fetchmail, a pokud chce vaše heslo, získá potřebný dekodér z kódu fetchmailu.

Zašifrování hesla v `.fetchmailrc` by pouze dalo falešný pocit jistoty lidem, kteří o všem důkladně nepřemýšlejí.

Bezpečnostní systém je pouze tak bezpečný jako jeho tajemství. Mějte se na pozoru před pseudotajemstvími.

9 Nutné podmínky pro styl tržiště

První čtenáři tohoto článku se neustále dotazovali na podmínky, které je třeba splnit pro úspěšný vývoj ve stylu tržiště. Ptali se, jaké musí mít kvality vůdce projektu a v jakém stavu musí být kód ve chvíli, kdy je zpřístupněn veřejnosti a snaží se získat další vývojáře.

Je celkem jasné, že není možné programovat od počátku ve stylu tržiště. Je možné testovat, hledat chyby a vylepšovat na tržišti, ale bylo by velmi obtížné projekt **zahájit** ve stylu tržiště. Linus se o to nepokusil a já také ne. Vaše vznikající společenství vývojářů potřebuje něco, co může testovat a s čím si může hrát.

Když začnete hledat spolupracovníky, potřebujete jim představit **uskutečnitelný cíl**. Váš program nemusí pracovat příliš dobře, může být nepohodlný, obsahovat chyby a špatně dokumentován. Musí však přesvědčit budoucí vývojáře o tom, že se v dohledné budoucnosti může vyvinout v něco skutečně užitečného.

Linux i fetchmail měly již v době prvého zpřístupnění silnou a atraktivní konstrukci. Mnoho lidí, kteří přemýšlejí o programování ve stylu tržiště, toto považují zcela správně za základ úspěchu. Z tohoto faktu však zbrkle docházejí k závěru, že další nutnou podmínkou je konstrukční intuice vedoucího projektu a jeho chytrost.

Linus ale převzal svoji konstrukci od Unixu a já tu svoji z předchůdce popclienta (ačkoliv ta se potom podstatně změnila, procentuálně vzato mnohem více než v případě Linuxu). Takže musí mít vedoucí/koordinátor pro projekt ve stylu tržiště neobyčejný konstrukční talent nebo mu stačí využívat talent ostatních?

Já se domnívám, že není zcela nutné, aby koordinátor byl schopen tvořit dokonalé konstrukce, ale je naprosto nutné, aby byl schopen **rozeznat dobré nápady ostatních**.

Linux i fetchmail to potvrzují. Linus, ačkoliv není (jak jsme již diskutovali) neobyčejně originální konstruktér, prokázal svoji výbornou schopnost rozpoznat dobré nápady a integrovat je do jádra Linuxu. A já jsem již popsal, že s tím nejlepším nápadem ohledně fetchmailu (SMTP přesměrování) přišel někdo jiný.

Moji první čtenáři se mi snažili podbízet tím, že tvrdili, že jsem náchylný podceňovat originalitu konstrukce u projektů tržiště, neboť já sám jsem jí obdařen, a proto ji považuji za samozřejmou. V tom může být něco pravdy, konstrukce (na rozdíl od kódování nebo hledání chyb) je mojí nejsilnější stránkou.

S originalitou při konstrukci programů je ale problém. Začnete podvědomě hledat původní a složitá řešení tam, kde je na místě použít něco robustního a jednoduchého. Některé mé projekty dříve selhaly, protože jsem se dopustil podobných chyb, v případě fetchmailu se mi jich ale podařilo vyvarovat.

Já se domnívám, že projekt fetchmail uspěl částečně proto, že jsem omezil svoje tendence hledat chytrá řešení. To ale argumentuje proti tvrzení o nutnosti originality v projektu ve stylu tržiště. A vezměte si Linux. Řekněme, že by se

Linus Torvalds snažil zakomponovat originální nápady během vývoje systému. Je pravděpodobné, že by vzniklé jádro systému bylo tak stabilní a úspěšné?

Je zapotřebí mít jistou úroveň konstrukčních schopností i programátorského umění, nicméně si myslím, že prakticky každý, kdo o něčem takovém uvažuje bude na patřičné úrovni. Vnitřní trh otevřeného společenství tlačí na všechny, aby nezačínali projekty, které nejsou schopni uskutečnit. Dosud se zdá, že vše funguje.

Existuje ale další schopnost, která se obvykle nespojuje s rozvojem software a která je stejně důležitá, jako je schopnost konstruovat, a možná ještě důležitější. Vedoucí projektu tržiště musí být schopný komunikovat a zacházet s lidmi.

To by mělo být samozřejmé. Abyste mohli stavět vývojářskou společnost, musíte přitáhnout lidi, získat jejich zájem o to, co děláte a snažit se, aby byli spokojení s prací, kterou dělají. Technické zapálení hodně pomáhá, ale zdaleka není vším. Vaše povaha je také důležitá.

Není náhodné, že Linus je příjemný chlapík, kterého mají lidé rádi a chtějí mu pomoci. Není náhodné, že já jsem energický extrovert, který rád pracuje v davu a má některé instinkty a způsoby komika. Aby projekt ve stylu tržiště uspěl, velmi pomáhá, pokud dokážete alespoň trochu okouzlit lidi.

10 Společenský kontext otevřeného software

Je velkou pravdou, že nejlepší programy začínají tak, že autor sám potřebujete vyřešit své každodenní problémy, a šíří se proto, že se ukáže, že takový problém trápí mnoho ostatních uživatelů. To nás vede zpět k pravidlu 1, které je přeformulováno do možná užitečnější podoby:

18. Pokud chcete pracovat na zajímavém problému, začněte tím, že naleznete problém, který zajímá vás osobně.

Tak začal i Carl Harris s popclientem a já s fetchmailem. To se ale ví už dávno. To zajímavé, co nám Linux a fetchmail přinesl, je další krok. Vývoj programů ve velké skupině vývojářů a uživatelů.

V knize The Mythical Man-Month, Fred Brooks tvrdí, že programátorův čas není nahraditelný. Pokud přidáte programátory do projektu, který se opožďuje, opozdí se ještě více. Dokazuje, že složitost a problémy s komunikací se zvyšují se čtvercem počtu programátorů, zatímco množství práce stoupá pouze lineárně. Toto tvrzení se později stalo Brookovým zákonem, který je často považován za předmět víry. Kdyby ale tento zákon platil beze zbytku, Linux by nemohl existovat.

Klasická kniha Geralda Weinberga: „The Psychology Of Computer Programming“, obsahuje něco, co můžeme při zpětném pohledu považovat za nesmírně důležitou opravu Brookova tvrzení. V jeho diskuzi „nesobeckého programování“ Weinberg tvrdí, že ve skupinách, ve kterých si vývojáři nechrání svůj kód a vybízejí ostatní, aby jim pomohli vyhledávat chyby a navrhovat vylepšení, projekty probíhají mnohem rychleji než jinde.

Weinbergova terminologie možná zabránila tomu, aby jeho analýza získala takové přijetí, jaké si zasluhovala, člověk se musí usmát při myšlence, že internetovští hackeři jsou „nesobečtí“. Nicméně, myslím, že jeho argumenty nebyly nikdy tak aktuální jako dnes.

Historie Unixu nás měla připravit na to, co se nyní učíme z Linuxu (a co jsem v menším měřítku ověřil experimentálně tak, že jsem úmyslně kopíroval Linusovy metody). Tedy to, že zatímco kódování zůstává víceméně samotářskou aktivitou, opravdu velké myšlenky se uskutečňují tehdy, pokud je využita pozornost a mozky celé společnosti. Vývojář, který využívá pouze vlastní mozek v uzavřeném projektu musí zaostat za vývojářem, který dokáže iniciovat otevřený evoluční projekt, ve kterém se odhalování chyb a vylepšení věnují stovky lidí.

Tradičnímu Unixovému světu bylo zabráněno v dotažení tohoto přístupu několika faktory. Jedním z nich byla právní omezení, různé licence, obchodní tajemství a zájmy. Další překážkou (při zpětném pohledu) bylo to, že internet ještě nebyl na dostatečné úrovni.

Před levným internetem existovala prostorově omezená společenství, jejichž kultura podporovala Weinbergovo „nesobecké programování“, ve kterých mohl programátor snadno přilákat mnoho diváků a spolupracovníků. Bellovy laboratoře,

MIT AI laboratoře, UC Berkeley se staly legendárními domovy vynálezů a jsou stále velmi plodné.

Linux byl prvým projektem, který vědomě a úspěšně využil celý **svět** jako svoji studnici talentů. Nemyslím si, že je náhodné, že Linux vznikal v době zrodu WWW a že Linux opustil svá batolecí léta se začátkem nástupu internetu do běžného povědomí (1993-1994). Linus byl první, který se naučil hrát podle nových pravidel, které nastolil všudypřítomný internet.

Zatímco levný internet byl nutnou podmínkou pro to, aby se linuxový model uplatnil, myslím, že to nebyla podmínka dostačující. Dalším nesmírně důležitým faktorem byl vývoj stylu vedení a vytvoření zvyků, které umožnily vývojářům získat další spolupracovníky a využít všechny možnosti tohoto media.

Ale jaký je styl vedení a jaké jsou tyto zvyky? Tyto zvyklosti nemohou být založeny na moci, a i pokud by byly, tak by vedení donucením nemohlo přinést výsledky, které vidíme. Weinberg cituje z autobiografie „Memoáry revoluce“, kterou v 19. století napsal ruský anarchista Petr Alexejevič Kropotkin.

„Jelikož jsem se narodil v rodině, která vlastnila nevolníky, započal jsem svůj aktivní život, jako všichni mladí muži v mém věku, s pevnou vírou v nutnost povelů, příkazů, trestů atd. Brzy jsem ale musel řídit důležité záležitosti a jednat se svobodnými muži, a zde měla každá chyba závažné následky. Začal jsem oceňovat rozdíl mezi činností na základě příkazů a disciplíny a činností na základě společného porozumění. To prvé obdivuhodně funguje při vojenské přehlídce, ale nemá žádnou cenu ve skutečném životě, kde cíle může být dosaženo pouze úsilím mnoha spolupracujících myslí.“

Velké úsilí mnoha spolupracujících mozků, to je přesně to, co projekt jako Linux vyžaduje, a model řízení založený na povelech je zcela nemožný v prostředí dobrovolníků v anarchistickém ráji, který nazýváme internet. Aby mohli pracovat a soutěžit efektivně, programátoři, kteří chtějí vést projekty založené na spolupráci, se musí naučit, jak získat a povzbuzovat skupiny lidí se společným zájmem, jak neurčitě naznačuje Kropotkin svým „principem porozumění“. Musí se naučit využívat Linusův zákon.

Dříve jsem se zmínil o „Delphi efektu“ jako o možném vysvětlení Linusova zákona. Ještě silnější analogie s přizpůsobivými systémy ale nabízí biologie a ekonomie. Linusův svět se v mnoha ohledech chová jako volný trh nebo ekologie, společnost sobeckých individuí, která se snaží maximalizovat užitek a při tomto ději vzniká sebeopravující se spontánní řád více propracovaný a účinnější, než může dosáhnout jakékoliv centrální plánování. Zde bychom měli hledat „princip porozumění“.

Užitková funkce, kterou linuxoví programátoři maximalizují není klasicky ekonomická, ale lze ji spojit s uspokojením vlastního já a získáním dobré reputace mezi ostatními programátory. (Někdo by mohl nazvat toto chování altruistické, ale tento pohled ignoruje fakt, že altruismus je sám o sobě formou uspokojení

vlastního já pro altruistu). Dobrovolná společenství, která pracují podobným způsobem nejsou ve skutečnosti vzácná, podobným, ve kterém jsem zapojen již velmi dlouho je okruh přátel science fiction, ve kterém je výslovně uznáváno, že hlavním motivem pro spolupráci je zvýšení vlastní reputace mezi ostatními.

Linus tím, že se úspěšně postavil do role ochránce projektu, který je z velké části rozvíjen někým jiným a tím, že získával podporu pro projekt dokud se projekt nestal sebeudržujícím, ukázal, že plně pochopil Kropotkinovo pravidlo sdíleného porozumění. Kvasi-ekonomický pohled na Linusův svět nám umožňuje pochopit, jak se tento princip uplatňuje.

Můžeme nahlížet na Linusovy metody jako na způsob, jak vytvořit účinný trh v uspokojování vlastního já, tím, že připoutá sobectví individuálních programátorů co nejpevněji k obtížným cílům, které mohou být dosaženy pouze vytrvalou spoluprací. V projektu fetchmailu jsem ukázal (ačkoli v menším měřítku), že tato metoda může být napodobena s dobrými výsledky. Možná, že jsem vše dělal s plnějším vědomím a systematičtěji než Linus sám.

Mnoho lidí, zejména těch, kteří politicky nedůvěřují volnému trhu, očekávají, že společnost sebeřídících egoistů bude fragmentována, bude ochraňovat svá území, bude plýtvat zdroji, vše tajit a bude k sobě nepřátelská. Ale toto očekávání je jasně vyvráceno například překvapující různorodostí, kvalitou a hloubkou dokumentace Linuxu. Je to zázrak, když si uvědomíme, jak programátoři **nenávidí** psaní dokumentace. Jak je tedy možné, že programátoři Linuxu ji produkují tolik? Je zřejmé, že Linuxův volný trh v sebeuspokojování funguje lépe při nastolení ctnostného chování než masivně financované dokumentační útvary poskytovatelů komerčního software.

Fetchmail i Linux ukázaly, že pokud dostatečně odměním ego mnoha jiných programátorů, silný vývojář-koordinátor může využít internet, aby získal mnoho spolupracovníků, aniž se projekt zhroutí v chaotický zmatek. Takže já navrhuji následující protiargument k Brookově zákonu.

19. Pokud má koordinátor projektu k dispozici médium alespoň tak dobré jako internet a dokáže vést bez příkazů, mnoho hlav je nevyhnutelně lepší než jedna.

Myslím si, že budoucnost otevřeného software bude stále více patřit lidem, kteří vědí, jak se chovat v Linusově hře, lidem, kteří opustí katedrálu a vezmou si tržiště za své. Tím nechci říci, že už nezáleží na individuálních vizích a velkých schopnostech. Spíše si myslím, že budoucnost patří lidem, kteří začnou s individuální vizí a velkými schopnostmi a ty potom zmnohonásobí ve vytvořených skupinách se společným zájmem.

A možná nejen budoucnost **otevřeného software**. Žádný vývojář v uzavřeném projektu nemůže mít takovou nabídku talentů pro vyřešení problému, jako se nachází ve společenství Linuxu. Jen málokdo si může dovolit najmout více než 200 lidí, kteří přispívali do fetchmailu.

Možná že nakonec kultura otevřeného software zvítězí ne proto, že je morálně správná, nebo že hamounění software je morálně špatné (za předpokladu, že tomu druhému věříte, já ani Linus ne), ale jednoduše proto, že uzavřené projekty nemohou vyhrát v evolučním zápase s otevřeným systémem, který může vynaložit o několik řádů více kvalifikovaného času na řešení problémů.

11 Poděkování

Tento článek byl vylepšen díky diskuzím s mnoha lidmi. Zejména děkuji Jeffovi Dutkymu, který navrhl termín „hledání chyb je paralelizovatelné“ a pomohl vyvinout analýzu, která z něj vychází. Rovněž děkuji Nancy Lebovitz za její návrh emulace Weinberga citací z Kropotkina. Vnímavou kritikou rovněž přispěli Joan Eslinger a Marty Franz. Jsem vděčný členům PLUG (Skupina uživatelů Linuxu ve Philadelphii), kteří se stali mým prvním testovacím publikem pro prvou veřejnou verzi článku. A nakonec, děkuji Linusovi Torvaldsovi za jeho cenné připomínky a za povzbuzení, které se mi dostalo od samého počátku.

12 K dalšímu čtení

Několikrát jsem citoval z klasické práce Frederika P. Brooka „The Mythical Man-Month“. Vřele doporučuji edici k 25 výročí (ISBN 0-201-83595-9), ke které je přidán článek z roku 1986 „No Silver Bullet“.

Tato nová edice je doplněna neocenitelným zpětným pohledem na uplynulých 20 let, ve kterém se Brook upřímně přiznává k tvrzením, která nebyla potvrzena dalším vývojem. Já jsem toto zpětné ohlédnutí poprvé četl ve chvíli, kdy tento článek byl z velké části hotov a překvapilo mne, že Brooks přičítá principy tržiště Microsoftu! (Ve skutečnosti se ale toto tvrzení ukázalo mylné. V roce 1998 jsme se z uniklých dokumentů, tzv. Halloween Documents dozvěděli, že vnitřní vývojářská komunita Microsoftu je balkanizovaná, bez možností obecného přístupu ke zdrojům, které styl tržiště vyžaduje.)

Gerald M. Weinberg v knize The Psychology Of Computer Programming (New York, Van Nostrand Reinhold 1971) zavedl poněkud nešťastně označený pojem „nesobecké programování“. Ačkoliv nebyl zdaleka prvým, kdo si uvědomil nedostatečnost „principu založeném na příkazech“, byl pravděpodobně první, který rozpoznal a zdůvodňoval toto stanovisko s ohledem na vývoj software.

Richard P. Gabriel, který přemýšlel nad Unixovou kulturou v období před Linuxem, váhavě argumentoval o přednostech jednoduchých stylů tržiště ve svém článku z roku 1989 Lisp: Dobré zprávy, špatné zprávy a jak zvítězit. Ačkoliv již v některých ohledech zastaralá, tento esej je stále uctíván mezi příznivci Lispu (včetně mne). Byl jsem upozorněn, že část nazvaná „Horší je lepší“ lze vyložit jako předpověď Linuxu. Tento článek je přístupný na WWW na adrese http://www.naggum.no/worse-is-better.html

Kniha De Marca a Listera Peopleware: Productive Projects and Teams (New York; Dorset House, 1987; ISBN 0-932633-05-6) je nedoceněný klenot. Byl jsem velmi potěšen, když Fred Brooks jej citoval ve své retrospektivě. Ačkoliv málo z toho, co autoři píší je přímo aplikovatelné na Linux, jejich náhled na nutné podmínky pro tvořivou práci je přesný a cenný pro každého, který chce přenést některé přednosti tržiště do komerčního prostředí.

Na závěr musím připustit, že jsem článek téměř nazval „Katedrála a Agora“. Agora je řecký výraz pro otevřený trh nebo místo veřejných setkání. Články Marka Millera a Erica Drexlera popisující vlastnosti tržních počítačových ekologií, mne pomohly ujasnit si analogické jevy v otevřeném společenství, když jsem narazil na Linux o pět let později. Tyto články jsou dostupné na síti na adrese `www.agorics.com/agorpapers.html`.

13 Epilog: Netscape přijal tržiště

Je to zvláštní pocit, když si uvědomíte, že pomáháte vytvářet historii...

26. ledna 1998, asi sedm měsíců poté, co jsem poprvé publikoval tento článek, Netscape Communications, Inc. oznámil své plány zveřejnit zdrojový kód Netscape Communicator. Neměl jsem žádné tušení o tom, že se k tomu schyluje do dne oznámení.

Eric Hahn, náměstek ředitele a hlavní technolog v Netscape mi krátce poté napsal tento email: „Jménem nás všech v Netscape Vám chci poděkovat za to, že jste nás k tomuto nápadu přivedl. Vaše myšlenky a články byly základní inspirací pro naše rozhodnutí."

Následující týden jsem byl pozván společností Netscape do Silicon Valley na celodenní strategickou konferenci s jejich hlavními manažery a techniky. Navrhli jsme strategii pro zveřejnění zdrojů a pro licence a připravili další plány, které, jak doufáme, budou mít dalekosáhlé a kladné účinky.

Netscape nám poskytuje velký reálný test modelu ve stylu tržiště v komerčním světě. Stojíme před velkým nebezpečím. Pokud tento projekt selže, může být celý koncept tak zdiskreditován, že komerčním svět na něj zanevře na další desetiletí.

Na druhé straně je to také skvělá příležitost. Počáteční reakce Wall Streetu i jinde byla obezřetně kladná. Byla nám dána šance se osvědčit. Pokud získá Netscape díky tomuto tahu podstatný podíl na trhu, může to způsobit dlouho očekávanou revoluci v softwarovém průmyslu.

Příští rok bude velmi zajímavý.