Original location: **http://www.netaxs.com/~esr/writings/cathedral-bazaar/**

Eric Raymond

The Cathedral and the Bazaar

22.11.1998

I anatomize a successful open-source project, fetchmail, that was run as a deliberate test of some surprising theories about software engineering suggested by the history of Linux. I discuss these theories in terms of two fundamentally different development styles, the ``cathedral'' model of most of the commercial world versus the ``bazaar'' model of the Linux world. I show that these models derive from opposing assumptions about the nature of the software-debugging task. I then make a sustained argument from the Linux experience for the proposition that ``Given enough eyeballs, all bugs are shallow'', suggest productive analogies with other self-correcting systems of selfish agents, and conclude with some exploration of the implications of this insight for the future of software.

Contents

Eric Raymond

Katedrála a tržiště

22.11.1998

Přeložil: Miloslav Nic [MNaaaa]
27.7.1999

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.

Obsah

TOC / OBSAH >>> EN CS EN/CS ZVON P/T

1. The Cathedral and the Bazaar

Linux is subversive. Who would have thought even five years ago that a world-class operating system could coalesce as if by magic out of part-time hacking by several thousand developers scattered all over the planet, connected only by the tenuous strands of the Internet?

Certainly not I. By the time Linux swam onto my radar screen in early 1993, I had already been involved in Unix and open-source development for ten years. I was one of the first GNU contributors in the mid-1980s. I had released a good deal of open-source software onto the net, developing or co-developing several programs (nethack, Emacs VC and GUD modes, xlife, and others) that are still in wide use today. I thought I knew how it was done.

Linux overturned much of what I thought I knew. I had been preaching the Unix gospel of small tools, rapid prototyping and evolutionary programming for years. But I also believed there was a certain critical complexity above which a more centralized, a priori approach was required. I believed that the most important software (operating systems and really large tools like Emacs) needed to be built like cathedrals, carefully crafted by individual wizards or small bands of mages working in splendid isolation, with no beta to be released before its

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 letl 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.

time.

Linus Torvalds's style of development - release early and often, delegate everything you can, be open to the point of promiscuity - came as a surprise. No quiet, reverent cathedral-building here -- rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from ***anyone***) out of which a coherent and stable system could seemingly emerge only by a succession of miracles.

Vývoj ve stylu Linuse Torvalda - 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.

The fact that this bazaar style seemed to work, and work well, came as a distinct shock. As I learned my way around, I worked hard not just at individual projects, but also at trying to understand why the Linux world not only didn't fly apart in confusion but seemed to go from strength to strength at a speed barely imaginable to cathedral-builders.

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.

By mid-1996 I thought I was beginning to understand. Chance handed me a perfect way to test my theory, in the form of an open-source project which I could consciously try to run in the bazaar style. So I did -- and it was a significant success.

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.

In the rest of this article, I'll tell the story of that project, and I'll use it to propose some aphorisms about effective open-source development. Not all of these are things I first learned in the Linux world, but we'll see how the Linux world gives them particular point. If I'm correct, they'll help you understand exactly what it is

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

that makes the Linux community such společenství takovou studnici dobrého
a fountain of good software -- and help software a pomůže i vám zvýšit
you become more productive yourself. produktivitu.

TOC / OBSAH >>> EN CS EN/CS ZVON P/T

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

2. The Mail Must Get Through

Since 1993 I'd been running the technical side of a small free-access ISP called Chester County InterLink (CCIL) in West Chester, Pennsylvania (I co-founded CCIL and wrote our unique multiuser bulletin-board software -- you can check it out by telnetting to **locke.ccil.org**. Today it supports almost three thousand users on thirty lines.) The job allowed me 24-hour-a-day access to the net through CCIL's 56K line -- in fact, it practically demanded it!

Accordingly, I had gotten quite used to instant Internet email. For complicated reasons, it was hard to get SLIP to work between my home machine (snark.thyrsus.com) and CCIL. When I finally succeeded, I found having to periodically telnet over to locke to check my mail annoying. What I wanted was for my mail to be delivered on snark so that I would be notified when it arrived and could handle it using all my local tools.

Simple sendmail forwarding wouldn't work, because my personal machine isn't always on the net and doesn't have a static IP address. What I needed was a program that would reach out over my SLIP connection and pull across my mail to be delivered locally. I knew such things existed, and that most of them used a

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 24 hodinový 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 dálei 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

simple application protocol called POP (Post Office Protocol). And sure enough, there was already a POP3 server included with locke's BSD/OS operating system.

I needed a POP3 client. So I went out on the net and found one. Actually, I found three or four. I used pop-perl for a while, but it was missing what seemed an obvious feature, the ability to hack the addresses on fetched mail so replies would work properly.

The problem was this: suppose someone named `joe' on locke sent me mail. If I fetched the mail to snark and then tried to reply to it, my mailer would cheerfully try to ship it to a nonexistent `joe' on snark. Hand-editing reply addresses to tack on `@ccil.org' quickly got to be a serious pain.

This was clearly something the computer ought to be doing for me. But none of the existing POP clients knew how! And this brings us to the first lesson:

1. Every good work of software starts by scratching a developer's personal itch.

Perhaps this should have been obvious (it's long been proverbial that ``Necessity is the mother of invention'') but too often software developers spend their days grinding away for pay at programs they neither need nor love. But not in the Linux world -- which may explain why the average quality of software originated in the Linux community is

(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" my 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 traví 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.

so high.

So, did I immediately launch into a furious whirl of coding up a brand-new POP3 client to compete with the existing ones? Not on your life! I looked carefully at the POP utilities I had in hand, asking myself ``which one is closest to what I want?

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. Because good programmers know what to write. Great ones know what to rewrite (and reuse).

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

While I don't claim to be a great programmer, I try to imitate one. An important trait of the great ones is constructive laziness. They know that you get an A not for effort but for results, and that it's almost always easier to start from a good partial solution than from nothing at all.

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 snažší začít od dobrých částečných řešení než od úplého počátku.

Linus Torvalds, for example, didn't actually try to write Linux from scratch. Instead, he started by reusing code and ideas from Minix, a tiny Unix-like OS for 386 machines. Eventually all the Minix code went away or was completely rewritten -- but while it was there, it provided scaffolding for the infant that would eventually become Linux.

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.

In the same spirit, I went looking for an existing POP utility that was reasonably well coded, to use as a development base.

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

The source-sharing tradition of the Unix world has always been friendly to code reuse (this is why the GNU project chose Unix as a base OS, in spite of serious reservations about the OS itself). The Linux world has taken this tradition nearly to its technological limit; it has terabytes

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 terabyty volných zdrojů. Jestliže zde

of open sources generally available. So spending time looking for some else's almost-good-enough is more likely to give you good results in the Linux world than anywhere else.

hledáte téměř vyhovující program, máte větší šanci na úspěch než kdekoliv jinde.

And it did for me. With those I'd found earlier, my second search made up a total of nine candidates -- fetchpop, PopTart, get-mail, gwpop, pimp, pop-perl, popc, popmail and upop. The one I first settled on was `fetchpop' by Seung-Hong Oh. I put my header-rewrite feature in it, and made various other improvements which the author accepted into his 1.9 release.

Obdobně vše fungovalo i v mém případě. Spolu s tím, co jsem nalezl již dříve, můj druhý průzkum odkryl celkem 9 kandidátů: fetchpop, PopTart, get-mail, 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.

A few weeks later, though, I stumbled across the code for `popclient' by Carl Harris, and found I had a problem. Though fetchpop had some good original ideas in it (such as its daemon mode), it could only handle POP3 and was rather amateurishly coded (Seung-Hong was a bright but inexperienced programmer, and both traits showed). Carl's code was better, quite professional and solid, but his program lacked several important and rather tricky-to-implement fetchpop features (including those I'd coded myself).

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.

Stay or switch? If I switched, I'd be throwing away the coding I'd already done in exchange for a better development base.

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.

A practical motive to switch was the presence of multiple-protocol support. POP3 is the most commonly used of the post-office server protocols, but not the only one.

Praktickým argumentem, který doporučoval změnu, byla podpora řady protokolů. POP3 je nejběžněji používaný protokol, ale není jediný. Fetchpop a další podobné programy neuměly POP2,

Fetchpop and the other competition didn't do POP2, RPOP, or APOP, and I was already having vague thoughts of perhaps adding **IMAP** (Internet Message Access Protocol, the most recently designed and most powerful post-office protocol) just for fun.

RPOP nebo APOP a já již začínal přemýšlet o přidání **IMAPu** (nejnovějšího a nejvšestrannějšího poštovního protokolu).

But I had a more theoretical reason to think switching might be as good an idea as well, something I learned long before Linux.

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

3. ``Plan to throw one away; you will, anyhow.'' (Fred Brooks, ``The Mythical Man-Month'', Chapter 11)

Počíteje s tím, že alespoň jednou budete muset vše přepsat, stejně vás to nemine ((Fred Brooks, ``The Mythical Man-Month'', Chapter 11)

Or, to put it another way, you often don't really understand the problem until after the first time you implement a solution. The second time, maybe you know enough to do it right. So if you want to get it right, be ready to start over ***at least*** once.

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.

Well (I told myself) the changes to fetchpop had been my first try. So I switched.

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

After I sent my first set of popclient patches to Carl Harris on 25 June 1996, I found out that he had basically lost interest in popclient some time before. The code was a bit dusty, with minor bugs hanging out. I had many changes to make, and we quickly agreed that the logical thing for me to do was take over the program.

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.

Without my actually noticing, the project had escalated. No longer was I just contemplating minor patches to an existing POP client. I took on maintaining an entire one, and there were ideas bubbling in my head that I knew would probably lead to major

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

changes.

In a software culture that encourages code-sharing, this is a natural way for a project to evolve. I was acting out this:

4. If you have the right attitude, interesting problems will find you.

But Carl Harris's attitude was even more important. He understood that

5. When you lose interest in a program, your last duty to it is to hand it off to a competent successor.

Without ever having to discuss it, Carl and I knew we had a common goal of having the best solution out there. The only question for either of us was whether I could establish that I was a safe pair of hands. Once I did that, he acted with grace and dispatch. I hope I will act as well when it comes my turn.

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 sami.

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.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

3. The Importance of Having Users

And so I inherited popclient. Just as importantly, I inherited popclient's user base. Users are wonderful things to have, and not just because they demonstrate that you're serving a need, that you've done something right. Properly cultivated, they can become co-developers.

Another strength of the Unix tradition, one that Linux pushes to a happy extreme, is that a lot of users are hackers too. Because source code is available, they can be ***effective*** hackers. This can be tremendously useful for shortening debugging time. Given a bit of encouragement, your users will diagnose problems, suggest fixes, and help improve the code far more quickly than you could unaided.

6. Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.

The power of this effect is easy to underestimate. In fact, pretty well all of us in the open-source world drastically underestimated how well it would scale up with number of users and against system complexity, until Linus Torvalds showed us differently.

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.

7. Pokud jednáte s uživateli jako se spolupracovníky, je to ta nejsnažší 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.

In fact, I think Linus's cleverest and most consequential hack was not the construction of the Linux kernel itself, but rather his invention of the Linux development model. When I expressed this opinion in his presence once, he smiled and quietly repeated something he has often said: ``I'm basically a very lazy person who likes to get credit for things other people actually do.'' Lazy like a fox. Or, as Robert Heinlein might have said, too lazy to fail.

In retrospect, one precedent for the methods and success of Linux can be seen in the development of the GNU Emacs Lisp library and Lisp code archives. In contrast to the cathedral-building style of the Emacs C core and most other FSF tools, the evolution of the Lisp code pool was fluid and very user-driven. Ideas and prototype modes were often rewritten three or four times before reaching a stable final form. And loosely-coupled collaborations enabled by the Internet, a la Linux, were frequent.

Indeed, my own most successful single hack previous to fetchmail was probably Emacs VC mode, a Linux-like collaboration by email with three other people, only one of whom (Richard Stallman, the author of Emacs and founder of the **FSF**) I have met to this day. It was a front-end for SCCS, RCS and later CVS from within Emacs that offered ``one-touch'' version control operations. It evolved from a tiny, crude sccs.el mode somebody else had written. And the development of VC succeeded because, unlike Emacs itself, Emacs Lisp code could go through release/test/improve generations very quickly.

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 precedens 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 a 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 .

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

4. Release Early, Release Often

Early and frequent releases are a critical part of the Linux development model. Most developers (including me) used to believe this was bad policy for larger than trivial projects, because early versions are almost by definition buggy versions and you don't want to wear out the patience of your users.

This belief reinforced the general commitment to a cathedral-building style of development. If the overriding objective was for users to see as few bugs as possible, why then you'd only release one every six months (or less often), and work like a dog on debugging between releases. The Emacs C core was developed this way. The Lisp library, in effect, was not -- because there were active Lisp archives outside the FSF's control, where you could go to find new and development code versions independently of Emacs's release cycle.

The most important of these, the Ohio State elisp archive, anticipated the spirit and many of the features of today's big Linux archives. But few of us really thought very hard about what we were doing, or about what the very existence of that archive suggested about problems in FSF's cathedral-building development model. I made one serious attempt around 1992 to get a lot of the Ohio code formally merged into the official Emacs Lisp library. I ran into political trouble and

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

was largely unsuccessful.

But by a year later, as Linux became widely visible, it was clear that something different and much healthier was going on there. Linus's open development policy was the very opposite of cathedral-building. The sunsite and tsx-11 archives were burgeoning, multiple distributions were being floated. And all of this was driven by an unheard-of frequency of core system releases.

Linus was treating his users as co-developers in the most effective possible way:

7. Release early. Release often. And listen to your customers.

Linus's innovation wasn't so much in doing this (something like it had been Unix-world tradition for a long time), but in scaling it up to a level of intensity that matched the complexity of what he was developing. In those early times (around 1991) it wasn't unknown for him to release a new kernel more than once a ***day!*** Because he cultivated his base of co-developers and leveraged the Internet for collaboration harder than anyone else, this worked.

But ***how*** did it work? And was it something I could duplicate, or did it rely on some unique genius of Linus Torvalds?

I didn't think so. Granted, Linus is a damn fine hacker (how many of us could engineer an entire production-quality operating system kernel?). But Linux didn't represent any awesome conceptual leap forward. Linus is not (or at least, not yet) an innovative

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 ranné 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 Torvalda?

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,

genius of design in the way that, say, Richard Stallman or James Gosling (of NeWS and Java) are. Rather, Linus seems to me to be a genius of engineering, with a sixth sense for avoiding bugs and development dead-ends and a true knack for finding the minimum-effort path from point A to point B. Indeed, the whole design of Linux breathes this quality and mirrors Linus's essentially conservative and simplifying design approach.

inovační genius designu, takový, jako je třeba Richard Stallman nebo James Gosling (NeWS, Java). Linus má spíše inženýrský génius, má šestý smysl pro obcházení chyb a slepých uliček, má opravdový cit pro nalezení 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.

So, if rapid releases and leveraging the Internet medium to the hilt were not accidents but integral parts of Linus's engineering-genius insight into the minimum-effort path, what was he maximizing? What was he cranking out of the machinery?

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

Put that way, the question answers itself. Linus was keeping his hacker/users constantly stimulated and rewarded -- stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant (even ***daily***) improvement in their work.

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 uspokující práci, odměňované tím, že vidí neustálá (dokonce ***denní***) vylepšení své práce.

Linus was directly aiming to maximize the number of person-hours thrown at debugging and development, even at the possible cost of instability in the code and user-base burnout if any serious bug proved intractable. Linus was behaving as though he believed something like this:

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. Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.

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

Or, less formally, ``Given enough eyeballs, all bugs are shallow.'' I dub this: ``Linus's Law''.

My original formulation was that every problem ``will be transparent to somebody''. Linus demurred that the person who understands and fixes the problem is not necessarily or even usually the person who first characterizes it. ``Somebody finds the problem,'' he says, ``and somebody ***else*** understands it. And I'll go on record as saying that finding it is the bigger challenge.'' But the point is that both things tend to happen quickly.

Here, I think, is the core difference underlying the cathedral-builder and bazaar styles. In the cathedral-builder view of programming, bugs and development problems are tricky, insidious, deep phenomena. It takes months of scrutiny by a dedicated few to develop confidence that you've winkled them all out. Thus the long release intervals, and the inevitable disappointment when long-awaited releases are not perfect.

In the bazaar view, on the other hand, you assume that bugs are generally shallow phenomena -- or, at least, that they turn shallow pretty quick when exposed to a thousand eager co-developers pounding on every single new release. Accordingly you release often in order to get more corrections, and as a beneficial side effect you have less to lose if an occasional botch gets out the door.

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 alepoň, 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.

And that's it. That's enough. If ``Linus's Law'' is false, then any system as complex as the Linux kernel, being hacked over by as many hands as the Linux kernel, should at some point have collapsed under the weight of unforseen bad interactions and undiscovered ``deep'' bugs. If it's true, on the other hand, it is sufficient to explain Linux's relative lack of bugginess.

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.

And maybe it shouldn't have been such a surprise, at that. Sociologists years ago discovered that the averaged opinion of a mass of equally expert (or equally ignorant) observers is quite a bit more reliable a predictor than that of a single randomly-chosen one of the observers. They called this the ``Delphi effect''. It appears that what Linus has shown is that this applies even to debugging an operating system -- that the Delphi effect can tame development complexity even at the complexity level of an OS kernel.

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 jakékohokoliv 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..

I am indebted to Jeff Dutky <dutky@wam.umd.edu> for pointing out that Linus's Law can be rephrased as ``Debugging is parallelizable''. Jeff observes that although debugging requires debuggers to communicate with some coordinating developer, it doesn't require significant coordination between debuggers. Thus it doesn't fall prey to the same quadratic complexity and management costs that make adding developers problematic.

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 spolupracovnící 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.

In practice, the theoretical loss of efficiency due to duplication of work by debuggers almost never seems to

Ve skutečnosti se teoretická ztráta účinnosti způsobená duplikací práce nezdá být problémem ve světě Linuxu.

be an issue in the Linux world. One effect of a ``release early and often policy'' is to minimize such duplication by propagating fed-back fixes quickly.

Jeden z výsledků časného a častého publikování je fakt, že případné duplikace jsou brzy odhaleny.

Brooks even made an off-hand observation related to Jeff's: ``The total cost of maintaining a widely used program is typically 40 percent or more of the cost of developing it. Surprisingly this cost is strongly affected by the number of users. ***More users find more bugs***.'' (my emphasis).

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.***

More users find more bugs because adding more users adds more different ways of stressing the program. This effect is amplified when the users are co-developers. Each one approaches the task of bug characterization with a slightly different perceptual set and analytical toolkit, a different angle on the problem. The ``Delphi effect'' seems to work precisely because of this variation. In the specific context of debugging, the variation also tends to reduce duplication of effort.

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.

So adding more beta-testers may not reduce the complexity of the current ``deepest'' bug from the ***developer's*** point of view, but it increases the probability that someone's toolkit will be matched to the problem in such a way that the bug is shallow ***to that person***.

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

Linus coppers his bets, too. In case there ***are*** serious bugs, Linux kernel version are numbered in such a way that potential users can make a choice either to run the last version designated ``stable'' or to ride the cutting edge and risk bugs in order to get new features. This tactic is not yet

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.

formally imitated by most Linux hackers, but perhaps it should be; the fact that either choice is available makes both more attractive.

Tuto taktiku ještě neuplatňuje většina Linuxových hackerů, ale asi by měla. Dostupnost obou alternativ přispívá k atraktivitě programu..

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

5. When Is A Rose Not A Rose?

Having studied Linus's behavior and formed a theory about why it was successful, I made a conscious decision to test this theory on my new (admittedly much less complex and ambitious) project.

But the first thing I did was reorganize and simplify popclient a lot. Carl Harris's implementation was very sound, but exhibited a kind of unnecessary complexity common to many C programmers. He treated the code as central and the data structures as support for the code. As a result, the code was beautiful but the data structure design ad-hoc and rather ugly (at least by the high standards of this old LISP hacker).

I had another purpose for rewriting besides improving the code and the data structure design, however. That was to evolve it into something I understood completely. It's no fun to be responsible for fixing bugs in a program you don't understand.

For the first month or so, then, I was simply following out the implications of Carl's basic design. The first serious change I made was to add IMAP support. I did this by reorganizing the protocol machines into a generic driver and three method tables (for POP2, POP3, and IMAP). This and the previous changes illustrate a general principle that's good for programmers to keep in mind, especially in languages like C that don't

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.

naturally do dynamic typing:

9. Smart data structures and dumb code works a lot better than the other way around.

Brooks, Chapter 9: ``Show me your [code] and conceal your [data structures], and I shall continue to be mystified. Show me your [data structures], and I won't usually need your [code]; it'll be obvious.''

Actually, he said ``flowcharts'' and ``tables''. But allowing for thirty years of terminological/cultural shift, it's almost the same point.

At this point (early September 1996, about six weeks from zero) I started thinking that a name change might be in order -- after all, it wasn't just a POP client any more. But I hesitated, because there was as yet nothing genuinely new in the design. My version of popclient had yet to develop an identity of its own.

That changed, radically, when fetchmail learned how to forward fetched mail to the SMTP port. I'll get to that in a moment. But first: I said above that I'd decided to use this project to test my theory about what Linus Torvalds had done right. How (you may well ask) did I do that? In these ways:

I released early and often (almost never less often than every ten days; during periodsof intense development, once a day).

I grew my beta list by adding to it everyone who contacted me about fetchmail.

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 dobui (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

I sent chatty announcements to the beta list whenever I released, encouraging people to participate.

And I listened to my beta testers, polling them about design decisions and stroking them whenever they sent in patches and feedback.

The payoff from these simple measures was immediate. From the beginning of the project, I got bug reports of a quality most developers would kill for, often with good fixes attached. I got thoughtful criticism, I got fan mail, I got intelligent feature suggestions. Which leads to:

10. If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.

One interesting measure of fetchmail's success is the sheer size of the project beta list, fetchmail-friends. At time of writing it has 249 members and is adding two or three a week.

Actually, as I revise in late May 1997 the list is beginning to lose members from its high of close to 300 for an interesting reason. Several people have asked me to unsubscribe them because fetchmail is working so well for them that they no longer need to see the list traffic! Perhaps this is part of the normal life-cycle of a mature bazaar-style project.

. 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 testovateli, jako by byli vaším nejcennějším kapitálem, oni se vaším nejcennějším kapitálem skutečně stanou.

Jedno zajímavé měřítko úspěchu fetchmailu je velikost seznamu testovatelů, 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.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

6. Popclient becomes Fetchmail

The real turning point in the project was when Harry Hochheiser sent me his scratch code for forwarding mail to the client machine's SMTP port. I realized almost immediately that a reliable implementation of this feature would make all the other delivery modes next to obsolete.

For many weeks I had been tweaking fetchmail rather incrementally while feeling like the interface design was serviceable but grubby -- inelegant and with too many exiguous options hanging out all over. The options to dump fetched mail to a mailbox file or standard output particularly bothered me, but I couldn't figure out why.

What I saw when I thought about SMTP forwarding was that popclient had been trying to do too many things. It had been designed to be both a mail transport agent (MTA) and a local delivery agent (MDA). With SMTP forwarding, it could get out of the MDA business and be a pure MTA, handing off mail to other programs for local delivery just as sendmail does.

Why mess with all the complexity of configuring a mail delivery agent or setting up lock-and-append on a mailbox when port 25 is almost guaranteed to be there on any platform with TCP/IP support in the first place? Especially when this means retrieved mail is guaranteed to look like normal sender-initiated SMTP mail, which is really what we want

6. Popclient se stává fetchmailem

Skutečně klíčovým okamžikem v projektu byl okamžik, když mi Harry Hochheiser poslal svůj návrh kódu pro přesměrování pošty na SMPT 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 SMPT 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?

anyway.

There are several lessons here. First, this SMTP-forwarding idea was the biggest single payoff I got from consciously trying to emulate Linus's methods. A user gave me this terrific idea -- all I had to do was understand the implications.

11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.

Interestingly enough, you will quickly find that if you are completely and self-deprecatingly truthful about how much you owe other people, the world at large will treat you like you did every bit of the invention yourself and are just being becomingly modest about your innate genius. We can all see how well this worked for Linus!

(When I gave this paper at the Perl conference in August 1997, Larry Wall was in the front row. As I got to the last line above he called out, religious-revival style, ``Tell it, tell it, brother!''. The whole audience laughed, because they knew it had worked for the inventor of Perl too.)

After a very few weeks of running the project in the same spirit, I began to get similar praise not just from my users but from other people to whom the word leaked out. I stashed away some of that email; I'll look at it again sometime if I ever start wondering whether my life has been worthwhile :-).

But there are two more fundamental, non-political lessons here that are

Zde se můžeme naučit několik lekcí. Zaprvé, nápad se SMPT, to byla největší odměna za to, že jsem se pokoušel napodobit Linusovi 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

general to all kinds of design.

12. Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.

I had been trying to solve the wrong problem by continuing to develop popclient as a combined MTA/MDA with all kinds of funky local delivery modes. Fetchmail's design needed to be rethought from the ground up as a pure MTA, a part of the normal SMTP-speaking Internet mail path.

When you hit a wall in development -- when you find yourself hard put to think past the next patch -- it's often time to ask not whether you've got the right answer, but whether you're asking the right question. Perhaps the problem needs to be reframed.

Well, I had reframed my problem. Clearly, the right thing to do was (1) hack SMTP forwarding support into the generic driver, (2) make it the default mode, and (3) eventually throw out all the other delivery modes, especially the deliver-to-file and deliver-to-standard-output options.

I hesitated over step 3 for some time, fearing to upset long-time popclient users dependent on the alternate delivery mechanisms. In theory, they could immediately switch to .forward files or their non-sendmail equivalents to get the same effects. In practice the transition might have been messy.

But when I did it, the benefits proved huge. The cruftiest parts of the driver code vanished. Configuration got radically simpler -- no more grovelling around for the system MDA and user's mailbox, no more worries about

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 reformulovat.

Já jsem tedy problém reformuloval. Je jasné, že správné bylo 1. přenést podporu SMTP do generického 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

whether the underlying OS supports file locking.	systém umožňuje uzamykání souborů.
Also, the only way to lose mail vanished. If you specified delivery to a file and the disk got full, your mail got lost. This can't happen with SMTP forwarding because your SMTP listener won't return OK unless the message can be delivered or at least spooled for later delivery.	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.
Also, performance improved (though not so you'd notice it in a single run). Another not insignificant benefit of this change was that the manual page got a lot simpler.	Zlepšila se i rychlost a významné bylo i zjednodušení manuálu.
Later, I had to bring delivery via a user-specified local MDA back in order to allow handling of some obscure situations involving dynamic SLIP. But I found a much simpler way to do it.	Později jsem musel doplnit některé nestandardní podmínky, ale to už bylo mnohem jednodušší.
The moral? Don't hesitate to throw away superannuated features when you can do it without loss of effectiveness. Antoine de Saint-Exupery (who was an aviator and aircraft designer when he wasn't being the author of classic children's books) said:	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. ``Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.''	***13. Konstrukční dokonalosti není dosaženo tehdy, když už není co přidat, ale tehdy, když už nemůžete nic odebrat.***
When your code is getting both better and simpler, that is when you ***know*** it's right. And in the process, the fetchmail design acquired an identity of its own, different from the ancestral popclient.	Když se Váš kód zároveň zlepšuje a stává jednodušším, potom ***víte***, že jste na správné cestě.
It was time for the name change. The new design looked much more like a dual of sendmail than the old popclient had; both are MTAs, but where sendmail pushes then delivers, the new	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.

popclient pulls then delivers. So, two months off the blocks, I renamed it fetchmail.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

7. Fetchmail Grows Up 7. Fetchmail dospívá

There I was with a neat and innovative design, code that I knew worked well because I used it every day, and a burgeoning beta list. It gradually dawned on me that I was no longer engaged in a trivial personal hack that might happen to be useful to few other people. I had my hands on a program every hacker with a Unix box and a SLIP/PPP mail connection really needs.

V ruce jsem měl uspořádanou a originální konstrukci, kód, o kterém jsem věděl, že funguje, neboť jsem ho používal každý den a rostoucí seznam testovatelů. 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.

With the SMTP forwarding feature, it pulled far enough in front of the competition to potentially become a ``category killer'', one of those classic programs that fills its niche so competently that the alternatives are not just discarded but almost forgotten.

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.

I think you can't really aim or plan for a result like this. You have to get pulled into it by design ideas so powerful that afterward the results just seem inevitable, natural, even foreordained. The only way to try for ideas like that is by having lots of ideas -- or by having the engineering judgment to take other peoples' good ideas beyond where the originators thought they could go.

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 had the original idea to build a simple native Unix for the 386, for use as a teaching tool. Linus Torvalds pushed the Minix concept further than Andrew probably thought it could go -- and it grew into something wonderful. In the same way (though on a smaller

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

scale), I took some ideas by Carl Harris and Harry Hochheiser and pushed them hard. Neither of us was `original' in the romantic way people think is genius. But then, most science and engineering and software development isn't done by original genius, hacker mythology to the contrary.

Harry Hochheisera a dotáhl jsem je do konce. Vždyť většina vědy, inženýrství a softwarového vývoje není vytvářena nějakým originálním geniem, ač to hackerské bájesloví tvrdí.

The results were pretty heady stuff all the same -- in fact, just the kind of success every hacker lives for! And they meant I would have to set my standards even higher. To make fetchmail as good as I now saw it could be, I'd have to write not just for my own needs, but also include and support features necessary to others but outside my orbit. And do that while keeping the program simple and robust.

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.

The first and overwhelmingly most important feature I wrote after realizing this was multidrop support -- the ability to fetch mail from mailboxes that had accumulated all mail for a group of users, and then route each piece of mail to its individual recipients.

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 obsahovali poštu více uživatelů a poté ji distribuovat jednotlivým příjemcům.

I decided to add the multidrop support partly because some users were clamoring for it, but mostly because I thought it would shake bugs out of the single-drop code by forcing me to deal with addressing in full generality. And so it proved. Getting **RFC 822** parsing right took me a remarkably long time, not because any individual piece of it is hard but because it involved a pile of interdependent and fussy details.

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.

But multidrop addressing turned out to be an excellent design decision as well. Here's how I knew:	Ukázalo se, že se také jednalo a výborné konstrukční rozhodnutí.
14. Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.	***14. Jakýkoliv nástroj by měl být užitečný očekávaným způsobem, ale opravdu velké nástroje se hodí na použití, které jste nikdy neočekával***
The unexpected use for multi-drop fetchmail is to run mailing lists with the list kept, and alias expansion done, on the ***client*** side of the SLIP/PPP connection. This means someone running a personal machine through an ISP account can manage a mailing list without continuing access to the ISP's alias files.	Tak tento přepsaný fetchmail byl neočekávaně využit při správě diskuzních skupin.
Another important change demanded by my beta testers was support for 8-bit MIME operation. This was pretty easy to do, because I had been careful to keep the code 8-bit clean. Not because I anticipated the demand for this feature, but rather in obedience to another rule:	Další důležitou změnou, kterou ode mne požadovali moji testovatelé, byla podpora 8-bit MIME. To bylo docela snadné, neboť jsem si dával pozor, abych nijak nepozměnil 8 bit. Nebylo to proto, že bych předvídal tento požadavek, spíše jsem se řídil dalším pravidlem:
15. When writing gateway software of any kind, take pains to disturb the data stream as little as possible -- and *never* throw away information unless the recipient forces you to!	***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.***
Had I not obeyed this rule, 8-bit MIME support would have been difficult and buggy. As it was, all I had to do is read **RFC 1652** and add a trivial bit of header-generation logic.	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.
Some European users bugged me into adding an option to limit the number of messages retrieved per session (so they can control costs from their expensive phone networks). I resisted this for a long time, and I'm still not entirely happy about it. But if you're	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

writing for the world, you have to listen to your customers -- this doesn't change just because they're not paying you in money.	píšete pro celý svět, musíte naslouchat svým zákazníkům, to se nemění jenom proto, že nejste placen penězi.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

8. A Few More Lessons From Fetchmail

Before we go back to general software-engineering issues, there are a couple more specific lessons from the fetchmail experience to ponder.

The rc file syntax includes optional `noise' keywords that are entirely ignored by the parser. The English-like syntax they allow is considerably more readable than the traditional terse keyword-value pairs you get when you strip them all out.

These started out as a late-night experiment when I noticed how much the rc file declarations were beginning to resemble an imperative minilanguage. (This is also why I changed the original popclient `server' keyword to `poll').

It seemed to me that trying to make that imperative minilanguage more like English might make it easier to use. Now, although I'm a convinced partisan of the ``make it a language'' school of design as exemplified by Emacs and HTML and many database engines, I am not normally a big fan of ``English-like'' syntaxes.

Traditionally programmers have tended to favor control syntaxes that are very precise and compact and have no redundancy at all. This is a cultural legacy from when computing resources were expensive, so parsing stages had to be as cheap and simple as possible. English, with about 50% redundancy, looked like a very

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.

Rc syntax zahrnuje nepovinný parametr 'noise', které parser zcela ignoruje. Takováto anglicky znějící syntax 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 syntax.

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.

inappropriate model then.

This is not my reason for normally avoiding English-like syntaxes; I mention it here only to demolish it. With cheap cycles and core, terseness should not be an end in itself. Nowadays it's more important for a language to be convenient for humans than to be cheap for the computer.

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.

There are, however, good reasons to be wary. One is the complexity cost of the parsing stage -- you don't want to raise that to the point where it's a significant source of bugs and user confusion in itself. Another is that trying to make a language syntax English-like often demands that the ``English'' it speaks be bent seriously out of shape, so much so that the superficial resemblance to natural language is as confusing as a traditional syntax would have been. (You see this in a lot of so-called ``fourth generation'' and commercial database-query languages.)

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.)

The fetchmail control syntax seems to avoid these problems because the language domain is extremely restricted. It's nowhere near a general-purpose language; the things it says simply are not very complicated, so there's little potential for confusion in moving mentally between a tiny subset of English and the actual control language. I think there may be a wider lesson here:

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. When your language is nowhere near Turing-complete, syntactic sugar can be your friend.

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

Another lesson is about security by obscurity. Some fetchmail users asked me to change the software to store

Další lekce je o bezpečnosti přes utajení. Několik uživatelů mne požádalo, abych pozměnil program

passwords encrypted in the rc file, so snoopers wouldn't be able to casually see them.

I didn't do it, because this doesn't actually add protection. Anyone who's acquired permissions to read your rc file will be able to run fetchmail as you anyway -- and if it's your password they're after, they'd be able to rip the necessary decoder out of the fetchmail code itself to get it.

All .fetchmailrc password encryption would have done is give a false sense of security to people who don't think very hard. The general rule here is:

17. A security system is only as secure as its secret. Beware of pseudo-secrets.

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 pseudo tajemstvími.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

9. Necessary Preconditions for the Bazaar Style

Early reviewers and test audiences for this paper consistently raised questions about the preconditions for successful bazaar-style development, including both the qualifications of the project leader and the state of code at the time one goes public and starts to try to build a co-developer community.

It's fairly clear that one cannot code from the ground up in bazaar style. One can test, debug and improve in bazaar style, but it would be very hard to ***originate*** a project in bazaar mode. Linus didn't try it. I didn't either. Your nascent developer community needs to have something runnable and testable to play with.

When you start community-building, what you need to be able to present is a ***plausible promise***. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is convince potential co-developers that it can be evolved into something really neat in the foreseeable future.

Linux and fetchmail both went public with strong, attractive basic designs. Many people thinking about the bazaar model as I have presented it have correctly considered this critical, then jumped from it to the conclusion that a

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

high degree of design intuition and cleverness in the project leader is indispensable.

faktu však zbrkle docházejí k závěru, že další nutnou podmínkou je konstrukční intuice vedoucího projektu a jeho chytrost.

But Linus got his design from Unix. I got mine initially from the ancestral popclient (though it would later change a great deal, much more proportionately speaking than has Linux). So does the leader/coordinator for a bazaar-style effort really have to have exceptional design talent, or can he get by on leveraging the design talent of others?

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?

I think it is not critical that the coordinator be able to originate designs of exceptional brilliance, but it is absolutely critical that the coordinator be able to ***recognize good design ideas from others***.

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.***

Both the Linux and fetchmail projects show evidence of this. Linus, while not (as previously discussed) a spectacularly original designer, has displayed a powerful knack for recognizing good design and integrating it into the Linux kernel. And I have already described how the single most powerful design idea in fetchmail (SMTP forwarding) came from somebody else.

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.

Early audiences of this paper complimented me by suggesting that I am prone to undervalue design originality in bazaar projects because I have a lot of it myself, and therefore take it for granted. There may be some truth to this; design (as opposed to coding or debugging) is certainly my strongest skill.

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 (narozdíl od kódování nebo hledání chyb) je mojí nejsilnější stránkou.

But the problem with being clever and original in software design is that it gets to be a habit -- you start reflexively

S originalitou při konstrukci programů je ale problém. Začnete podvědomě hledat původní a složitá

making things cute and complicated when you should be keeping them robust and simple. I have had projects crash on me because I made this mistake, but I managed not to with fetchmail.

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.

So I believe the fetchmail project succeeded partly because I restrained my tendency to be clever; this argues (at least) against design originality being essential for successful bazaar projects. And consider Linux. Suppose Linus Torvalds had been trying to pull off fundamental innovations in operating system design during the development; does it seem at all likely that the resulting kernel would be as stable and successful as what we have?

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?

A certain base level of design and coding skill is required, of course, but I expect almost anybody seriously thinking of launching a bazaar effort will already be above that minimum. The open-source community's internal market in reputation exerts subtle pressure on people not to launch development efforts they're not competent to follow through on. So far this seems to have worked pretty well.

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.

There is another kind of skill not normally associated with software development which I think is as important as design cleverness to bazaar projects -- and it may be more important. A bazaar project coordinator or leader must have good people and communications skills.

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.

This should be obvious. In order to build a development community, you need to attract people, interest them in what you're doing, and keep them happy about the amount of work they're doing. Technical sizzle will go a long way

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,

towards accomplishing this, but it's far from the whole story. The personality you project matters, too.	ale zdaleka není vším. Vaše povaha je také důležitá.
It is not a coincidence that Linus is a nice guy who makes people like him and want to help him. It's not a coincidence that I'm an energetic extrovert who enjoys working a crowd and has some of the delivery and instincts of a stand-up comic. To make the bazaar model work, it helps enormously if you have at least a little skill at charming people.	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.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

10. The Social Context of Open-Source Software

It is truly written: the best hacks start out as personal solutions to the author's everyday problems, and spread because the problem turns out to be typical for a large class of users. This takes us back to the matter of rule 1, restated in a perhaps more useful way:

18. To solve an interesting problem, start by finding a problem that is interesting to you.

So it was with Carl Harris and the ancestral popclient, and so with me and fetchmail. But this has been understood for a long time. The interesting point, the point that the histories of Linux and fetchmail seem to demand we focus on, is the next stage -- the evolution of software in the presence of a large and active community of users and co-developers.

In ``The Mythical Man-Month'', Fred Brooks observed that programmer time is not fungible; adding developers to a late software project makes it later. He argued that the complexity and communication costs of a project rise with the square of the number of developers, while work done only rises linearly. This claim has since become known as ``Brooks's Law'' and is widely regarded as a truism. But if Brooks's Law were the whole picture, Linux would be impossible.

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 nastavitelný. 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.

Gerald Weinberg's classic ``The Psychology Of Computer Programming'' supplied what, in hindsight, we can see as a vital correction to Brooks. In his discussion of ``egoless programming'', Weinberg observed that in shops where developers are not territorial about their code, and encourage other people to look for bugs and potential improvements in it, improvement happens dramatically faster than elsewhere.

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.

Weinberg's choice of terminology has perhaps prevented his analysis from gaining the acceptance it deserved -- one has to smile at the thought of describing Internet hackers as ``egoless''. But I think his argument looks more compelling today than ever.

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.

The history of Unix should have prepared us for what we're learning from Linux (and what I've verified experimentally on a smaller scale by deliberately copying Linus's methods). That is, that while coding remains an essentially solitary activity, the really great hacks come from harnessing the attention and brainpower of entire communities. The developer who uses only his or her own brain in a closed project is going to fall behind the developer who knows how to create an open, evolutionary context in which bug-spotting and improvements get done by hundreds of people.

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.

But the traditional Unix world was prevented from pushing this approach to the ultimate by several factors. One was the legal contraints of various licenses, trade secrets, and

Tradičnímu Unixovému světu bylo zabráněno v dotažení tohoto přístupu několika faktory. Jedním z nich byla právní omezení, různé licence, obchodní tajemství a zájmy. Další

commercial interests. Another (in hindsight) was that the Internet wasn't yet good enough.

Before cheap Internet, there were some geographically compact communities where the culture encouraged Weinberg's ``egoless'' programming, and a developer could easily attract a lot of skilled kibitzers and co-developers. Bell Labs, the MIT AI Lab, UC Berkeley -- these became the home of innovations that are legendary and still potent.

Linux was the first project to make a conscious and successful effort to use the entire ***world*** as its talent pool. I don't think it's a coincidence that the gestation period of Linux coincided with the birth of the World Wide Web, and that Linux left its infancy during the same period in 1993-1994 that saw the takeoff of the ISP industry and the explosion of mainstream interest in the Internet. Linus was the first person who learned how to play by the new rules that pervasive Internet made possible.

While cheap Internet was a necessary condition for the Linux model to evolve, I think it was not by itself a sufficient condition. Another vital factor was the development of a leadership style and set of cooperative customs that could allow developers to attract co-developers and get maximum leverage out of the medium.

But what is this leadership style and what are these customs? They cannot be based on power relationships -- and even if they could be, leadership by coercion would not produce the results we see. Weinberg quotes the

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.

Zatimco 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 síle, a i pokud by byly, tak by vedení z pozice síly nemohlo přinést výsledky, které vidíme. Weinberg cituje z autobiografie

autobiography of the 19th-century Russian anarchist Pyotr Alexeyvich Kropotkin's ``Memoirs of a Revolutionist'' to good effect on this subject:

"Memoáry revoluce", kterou v 19. století napsal ruský anarchista Petr Alexejevič Kropotkin.

``Having been brought up in a serf-owner's family, I entered active life, like all young men of my time, with a great deal of confidence in the necessity of commanding, ordering, scolding, punishing and the like. But when, at an early stage, I had to manage serious enterprises and to deal with [free] men, and when each mistake would lead at once to heavy consequences, I began to appreciate the difference between acting on the principle of command and discipline and acting on the principle of common understanding. The former works admirably in a military parade, but it is worth nothing where real life is concerned, and the aim can be achieved only through the severe effort of many converging wills.''

"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.

The ``severe effort of many converging wills'' is precisely what a project like Linux requires -- and the ``principle of command'' is effectively impossible to apply among volunteers in the anarchist's paradise we call the Internet. To operate and compete effectively, hackers who want to lead collaborative projects have to learn how to recruit and energize effective communities of interest in the mode vaguely suggested by Kropotkin's ``principle of understanding''. They must learn to use Linus's Law.

"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.

Earlier I referred to the ``Delphi effect'' as a possible explanation for Linus's Law. But more powerful

Dříve jsem se zmínil o "Delphi efektu" jako o možném vysvětlení Linusova zákona. Ještě silnější analogie s

analogies to adaptive systems in biology and economics also irresistably suggest themselves. The Linux world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved. Here, then, is the place to seek the ``principle of understanding''.

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".

The ``utility function'' Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers. (One may call their motivation ``altruistic'', but this ignores the fact that altruism is itself a form of ego satisfaction for the altruist). Voluntary cultures that work this way are not actually uncommon; one other in which I have long participated is science fiction fandom, which unlike hackerdom explicitly recognizes ``egoboo'' (the enhancement of one's reputation among other fans) as the basic drive behind volunteer activity.

"Užitková funkce, kterou linuxový 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, by successfully positioning himself as the gatekeeper of a project in which the development is mostly done by others, and nurturing interest in the project until it became self-sustaining, has shown an acute grasp of Kropotkin's ``principle of shared understanding''. This quasi-economic view of the Linux world enables us to see how that understanding is applied.

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 Linuxův svět nám umožňuje pochopit, jak se tento princip uplatňuje.

We may view Linus's method as a way to create an efficient market in ``egoboo'' -- to connect the selfishness of individual hackers as firmly as possible to difficult ends that can only be achieved by sustained cooperation. With the fetchmail project I have shown (albeit on a smaller scale) that his methods can be duplicated with good results. Perhaps I have even done it a bit more consciously and systematically than he.

Můžeme nahlížet na Linusovy metody jako na způsob, jak vytvořit účinný trh v uspokojování vlastního já, tím, že připoutá sobectví individuelních 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.

Many people (especially those who politically distrust free markets) would expect a culture of self-directed egoists to be fragmented, territorial, wasteful, secretive, and hostile. But this expectation is clearly falsified by (to give just one example) the stunning variety, quality and depth of Linux documentation. It is a hallowed given that programmers ***hate*** documenting; how is it, then, that Linux hackers generate so much of it? Evidently Linux's free market in egoboo works better to produce virtuous, other-directed behavior than the massively-funded documentation shops of commercial software producers.

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.

Both the fetchmail and Linux kernel projects show that by properly rewarding the egos of many other hackers, a strong developer/coordinator can use the Internet to capture the benefits of having lots of co-developers without having a project collapse into a chaotic mess. So to Brooks's Law I counter-propose the following:

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: Provided the development coordinator has a medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

I think the future of open-source software will increasingly belong to people who know how to play Linus's game, people who leave behind the cathedral and embrace the bazaar. This is not to say that individual vision and brilliance will no longer matter; rather, I think that the cutting edge of open-source software will belong to people who start from individual vision and brilliance, then amplify it through the effective construction of voluntary communities of interest.

And perhaps not only the future of ***open-source*** software. No closed-source developer can match the pool of talent the Linux community can bring to bear on a problem. Very few could afford even to hire the more than two hundred people who have contributed to fetchmail!

Perhaps in the end the open-source culture will triumph not because cooperation is morally right or software ``hoarding'' is morally wrong (assuming you believe the latter, which neither Linus nor I do), but simply because the closed-source world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem.

19. Pokud má koordinátor projektu k dispozici medium 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.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

11. Acknowledgements

This paper was improved by conversations with a large number of people who helped debug it. Particular thanks to Jeff Dutky <dutky@wam.umd.edu>, who suggested the ``debugging is parallelizable'' formulation, and helped develop the analysis that proceeds from it. Also to Nancy Lebovitz <nancyl@universe.digex.net> for her suggestion that I emulate Weinberg by quoting Kropotkin. Perceptive criticisms also came from Joan Eslinger <wombat@kilimanjaro.engr.sgi.com> and Marty Franz <marty@net-link.net> of the General Technics list. I'm grateful to the members of PLUG, the Philadelphia Linux User's group, for providing the first test audience for the first public version of this paper. Finally, Linus Torvalds's comments were helpful and his early endorsement very encouraging.

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 Torvaldovi za jeho cenné připomínky a za povzbuzení, které se mi dostalo od samého počátku.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

12. For Further Reading

I quoted several bits from Frederick P. Brooks's classic ***The Mythical Man-Month*** because, in many respects, his insights have yet to be improved upon. I heartily recommend the 25th Anniversary edition from Addison-Wesley (ISBN 0-201-83595-9), which adds his 1986 ``***No Silver Bullet***'' paper.

The new edition is wrapped up by an invaluable 20-years-later retrospective in which Brooks forthrightly admits to the few judgements in the original text which have not stood the test of time. I first read the retrospective after this paper was substantially complete, and was surprised to discover that Brooks attributes bazaar-like practices to Microsoft! (In fact, however, this attribution turned out to be mistaken. In 1998 we learned from the **Halloween Documents** that Microsoft's internal developer community is heavily balkanized, with the kind of general source access needed to support a bazaar not even truly possible.)

Gerald M. Weinberg's ***The Psychology Of Computer Programming*** (New York, Van Nostrand Reinhold 1971) introduced the rather unfortunately-labeled concept of ``egoless programming''. While he was nowhere near the first person to realize the futility of the ``principle of command'', he was probably the first to recognize and argue the point in

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 principz tržiště Microsoftu!.(Ve skutečnosti se ale toto tvrzení ukázalo mylné. V roce 1998 jsme se z uniklých dokumenů, tzv. **Halloween Documents** dozvěděli, že vnitřní vývojářská komunita Microsoftu je balkanizovaná, bez možností obecného přístupu ke zdrojům, které styl tržiště vyžaduje.

Gerald M. Weinberg's 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

particular connection with software development.

toto stanovisko s ohledem na vývoj software.

Richard P. Gabriel, contemplating the Unix culture of the pre-Linux era, reluctantly argued for the superiority of a primitive bazaar-like model in his 1989 paper ***Lisp: Good News, Bad News, and How To Win Big***. Though dated in some respects, this essay is still rightly celebrated among Lisp fans (including me). A correspondent reminded me that the section titled ``Worse Is Better'' reads almost as an anticipation of Linux. The paper is accessible on the World Wide Web at **http://www.naggum.no/worse-is-better.html**.

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**

De Marco and Lister's ***Peopleware: Productive Projects and Teams*** (New York; Dorset House, 1987; ISBN 0-932633-05-6) is an underappreciated gem which I was delighted to see Fred Brooks cite in his retrospective. While little of what the authors have to say is directly applicable to the Linux or open-source communities, the authors' insight into the conditions necessary for creative work is acute and worthwhile for anyone attempting to import some of the bazaar model's virtues into a commercial context.

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.

Finally, I must admit that I very nearly called this paper ``The Cathedral and the Agora'', the latter term being the Greek for an open market or public meeting place. The seminal ``agoric systems'' papers by Mark Miller and Eric Drexler, by describing the emergent properties of market-like computational ecologies, helped prepare me to think clearly about analogous phenomena in the

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

open-source culture when Linux rubbed my nose in them five years later. These papers are available on the Web at **http://www.agorics.com /agorpapers.html**.

http://www.agorics.com /agorpapers.html.

<<< TOC / OBSAH >>> EN CS EN/CS ZVON P/T

<<< TOC / OBSAH EN CS EN/CS ZVON P/T

13. Epilog: Netscape Embraces the Bazaar!

It's a strange feeling to realize you're helping make history....

On January 22 1998, approximately seven months after I first published this paper, Netscape Communications, Inc. announced plans to **give away the source for Netscape Communicator**. I had had no clue this was going to happen before the day of the announcement.

Eric Hahn, Executive Vice President and Chief Technology Officer at Netscape, emailed me shortly afterwards as follows: ``On behalf of everyone at Netscape, I want to thank you for helping us get to this point in the first place. Your thinking and writings were fundamental inspirations to our decision.''

The following week I flew out to Silicon Valley at Netscape's invitation for a day-long strategy conference (on Feb 4 1998) with some of their top executives and technical people. We designed Netscape's source-release strategy and license together, and laid some more plans that we hope will eventually have far-reaching and positive impacts on the open-source community. As I write, it is a bit too soon to be more specific; but details should be forthcoming within weeks.

Netscape is about to provide us with a large-scale, real-world test of the bazaar model in

13. Epilog: Netscape přijala 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, keré, jak doufáme, budou mít dalekosáhlé a kladné účinky.

Netscape nám poskytuje velký reálný test modelu ve stylu

the commercial world. The open-source culture now faces a danger; if Netscape's execution doesn't work, the open-source concept may be so discredited that the commercial world won't touch it again for another decade.

On the other hand, this is also a spectacular opportunity. Initial reaction to the move on Wall Street and elsewhere has been cautiously positive. We're being given a chance to prove ourselves, too. If Netscape regains substantial market share through this move, it just may set off a long-overdue revolution in the software industry.

The next year should be a very instructive and interesting time.

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.

<<< TOC / OBSAH EN CS EN/CS ZVON P/T