

The Cathedral And The Bazaar

By : Eric S. Raymond

**Translated by : Nima Aboutaleb
(nima717a@yahoo.com)**

آنانی را دوست میدارم که ویران میکنند
تا دوباره بر آن ویرانه ها بنائی نو و بهتر بسازند
« فردریش نیچه »

ý Abstract

من به تشریح یک پروژه باز متن موفق به نام *fetchmail* خواهم پرداخت، که به عنوان یک آزمایش سنجیده از برخی تنوری های حیرت آور درباره مهندسی نرم افزار و با اشاره به تاریخچه لینوکس بیان میشود. این تنوری ها را در قالب دو طرح پایه ای متفاوت در توسعه نرم افزار شرح خواهم داد؛ مدل کلیسایی که بیانگر اکثر پروژه های کنونی تجاری جهان است، در برابر مدل بازار از دنیای لینوکس. سپس نشان خواهم داد که این مدل ها، از ساختار های متضادی درباره طبیعت عیب یابی نرم افزار سرچشمه میگیرند. سپس، استدلالی را در حمایت از تجربه لینوکس برای اثبات این قضیه که "با وجود چندین نگاه جستجوگر، تمامی خطاها برطرف میشوند"، بیان خواهم کرد؛ با اشاره به قیاس هایی ارزشمند با دیگر سیستم های خود تصحیح. عاملهای خودپسند و نتیجه گیری با شناسایی برخی از نتایج این بینش در آینده دنیای نرم افزار.

ý Contents

- √ The Cathedral and the Bazaar
 - √ The mail must get through
 - √ The importance of having users
 - √ Release early, Release often
 - √ When is a Rose not a Rose ?
 - √ Popclient becomes Fetchmail
 - √ Fetchmail grows up
 - √ A few more lessons from Fetchmail
 - √ Necessary preconditions for the Bazaar style
 - √ The social context of Open-Source software
- √ Epilog : Netscape Embraces the bazaar

* زیر نویس ها توسط مترجم اضافه شده است

ý The Cathedral and the Bazaar

ý کلیسا و بازار

سیستم عامل لینوکس، ویرانگر است. حتی پنج سال پیش، چه کسی میتوانست تصورش را بکند که یک سیستم عامل جهانی بتواند همچون یک معجزه از ابتکارهای پاره وقتی چندین هزار تن از برنامه نویسان که در سرتا سر جهان پراکنده اند و بوسیله اینترنت با هم در ارتباطند، شکل یابد؟

حقیقتا، من چنین تصویری نمیکردم. اوائل سال 1993، زمانی که لینوکس به دایره ذهن من راه یافت، ده ها سال میشد که من درگیر یونیکس و جنبش باز-متن¹ بودم. من یکی از اولین شرکت کنندگان پروژه GNU² در اواسط دهه 1980 بودم و تعداد زیادی از نرم افزارهای باز متن را بر روش شبکه انتشار دادم، و در توسعه بسیاری از نرم افزارها (نظیر nethack, Emacs VC and GUD modes, xlife و ...) که هنوز بطور گسترده مورد استفاده قرار میگیرند، نقش مستقیم یا غیر مستقیم داشتم. فکر میکردم که میدانم چه چیزی در حال رخ دادن است.

لینوکس ویرانگر تر از آنچه میبود که من فکر میکردم. من با ابزارهایی ساده در حال وعظ با انجیل یونیکس بودم!، و سالها به شکلی تکاملی و با ساختار موجود برنامه نویسی میکردم. اما همچنین بر این باور بودم که یک نوع پیچیدگی بحرانی خاصی در روش قبلی وجود دارد که نیازمند روش ساخته یافته تر و متمرکز تری است. من بر این عقیده بودم که نرم افزارهای پایه ای و مهم (سیستم عامل و ابزارهای بزرگی همچون Emacs) میبایست با مهارتی خاص و بوسیله نوابغی منحصر بفرد یا گروهی از دانشمندان منزوی، همانند یک کلیسا ساخته شوند؛ بدون اینکه نسخه بتانی از آن را قبل از زمان ارائه آن توزیع کنند.

روش برنامه نویسی لینوس توروالدز³ (انتشار زود و متناوب برنامه، اجازه کامل به شما تا هر کاری را که میخواهید بر روی برنامه انجام دهید، و آزاد بودن در قاعده و ساختارش) یک شگفتی بود. نه تنها دیگر خبری از روش کلیسائی نبود، بلکه جامعه لینوکس بیشتر به یک بازار شلوغ از روشهای گوناگون و متنوع شباهت داشت (که شایستگی آن به خوبی بوسیله آرشیو سایت های لینوکسی، که تائید نظرات افراد مختلف در آن وجود دارد، قابل درک است)؛ که در این صورت، یک سیستم اینچنین پایدار و منسجم ظاهرا تنها میتوانست بوسیله یک توالی از معجزه ها بوجود آید.

این واقعیت که این سبک بازار مانند کار میکرد، و البته خوب هم کار میکرد، یک شوک محرز بود. زمانی که راهم را پیدا کردم، علاوه بر کار سخت در پروژه های مختلف، سعی

¹ Open Source .

² GUN is not Unix .

³ Linus Torvalds ؛ خالق هسته سیستم عامل لینوکس .

میکردم تا بفهمم چرا دنیای لاینوکس نه تنها بدلیل اغتشاش و پریشانی از هم پاشیده نشد، بلکه با قدرت و سرعتی مثال زدنی از کلیسا سازان! پیشی میگیرد.

در اواسط سال 1996 حس کردم که به نتایجی برای درک این مطلب رسیده ام. شانس و اقبال مرا به سمت راه درستی برای تست تنوری ام، راهنمایی کرد. در قالب یک پروژه باز متن، عمداً مدل بازار را بر روی آن پیاده کردم و نتیجه آن موفقیتی پر معنی و مهم شد.

در ادامه این مقاله، به شرح داستان آن پروژه خواهم پرداخت و در میان آن، نکات مورد نیاز برای یک پروژه باز متن موفق را بیان خواهم کرد. من تمامی این تجربیات را در ابتدا از دنیای لاینوکس یاد نگرفتم، اما در ادامه خواهیم دید که چطور دنیای لاینوکس به آنها ارزش ویژه ای میدهد. اگر نظر من درست باشد، به شما کمک خواهد کرد که تا دقیقاً دریابید که چه چیز، جامعه لاینوکس را به چنین منبع ارزشمندی از نرم افزار های خوب تبدیل کرده است – و همچنین به شما کمک خواهد کرد که خلاق تر و کارا تر شوید.

ý The mail must get through

ý نامه باید به مقصد برسد

در سال 1993، من در حال کار در قسمت فنی یک ISP دسترسی-آزاد¹ کوچک به نام CCIL² در غرب Chester در Pennsylvania بودم. (من یکی از موسسان CCIL بودم و تنها نرم افزار تابلوی اعلانات³ چندکاربره مان را نوشتم - شما میتوانید بوسیله telnet به locke.ccil.org آن را آزمایش کنید که هم اکنون تقریباً سه هزار کاربر را در 19 خط پشتیبانی میکند) این کار به من امکان دسترسی 24 ساعته به اینترنت را از طریق خط 56K CCIL میداد. در حقیقت، دقیقاً منطبق با نیازم بود !

طبیعتاً، با پست الکترونیکی نیز زیاد سر و کار داشتم. بنا به دلایلی، کار کردن با پروتکل SLIP⁴ بین ماشین خانگی ام (snark.thyrus.com) و CCIL خیلی سخت بود. سرانجام زمانی که موفق شدم، انجام تناوبی عمل telnet برای چک کردن نامه هایم را آزردهنده یافتم. آنچه که میخواستم این بود که نامه هایم چنان بر روی سیستم خانگی ام دریافت شوند که زمانی که نامه ها میرسند، من متوجه شوم و بتوانم آنها را بوسیله تمامی ابزارهای محلی ام مدیریت کنم.

سیستم ارسال ساده پست الکترونیکی⁵ جوابگو نبود، چرا که ماشین شخصی من همواره به شبکه متصل نبود و آدرس IP ثابتی نداشت. آنچه که نیاز داشتم، برنامه ای بود که به کانکشن SLIP من دسترسی پیدا کرده تا نامه هایم را در اختیارم قرار دهد. میدانستم که چنین چیزی وجود دارد و اکثر آنها از پروتکل ساده ای در لایه کاربرد به نام POP⁶ استفاده میکنند. و همانطور که مطمئن بودم، یک سرویس دهنده POP3 با سیستم عامل BSD/OS در آن زمان وجود داشت.

من نیاز به یک سرویس گیرنده POP3 داشتم. بنابراین، به شبکه مراجعه کردم و یکی پیدا کردم. در واقع، سه یا چهار تا پیدا کردم. برای مدتی از pop-perl استفاده کردم، اما آن فاقد یک ویژگی بدیهی بود؛ یعنی توانایی برای هک کردن آدرس های نامه واکشی⁷ شده تا پاسخ نامه ها به درستی داده شود.

1 . Free - Access .

2 . Chester Country InterLink .

3 . Bulletin - Board .

4 . Serial Line IP ؛ پروتکلی که در سال 1984 توسط ریک آدامز برای اتصال ایستگاههای sun بوسیله یک خط سریال مثل خط تلفن ابداع شد و یکی از استانداردهای انتقال بر روی خطوط نقطه به نقطه در لایه اول از پروتکل TCP/IP بشمار میرود.

5 . Simple Sendmail Forwarding .

6 . Post Office Protocol .

7 . fetch .

مشکل این بود : فرض کنید فردی با نام joe از locke¹ برای من نامه ای فرستاده باشد. اگر من نامه را از snark² واکنشی کرده و بخواهم به آن پاسخ دهم، در این صورت سیستم نامه رسان من تلاش میکند تا آن را به فردی به نام joe در snark که وجود خارجی ندارد، برساند. با دست ویرایش کردن آدرس ها تا @ccil.org را به آن ضمیمه شود، سریعا به کاری عذاب آور تبدیل شده بود.

حقیقتا، این کاری بود که کامپیوتر میبایست برای من انجام دهد. اما هیچ کدام از سرویس گیرنده های POP نمیدانستند که چطور باید این کار را انجام دهند و این به ما درس اول را داد :

1. شروع هر نرم افزار خوب از مشکلات شخصی برنامه نویسان آن است.

شاید این مسئله بدیهی به نظر برسد (این یک ضرب المثل قدیمی است که " نیاز مادر اختراع است ") اما اکثر توسعه دهندگان نرم افزار زمان زیادی از عمرشان را بدلائل اقتصادی، صرف نوشتن برنامه هانی میکنند که نه به آن نیاز دارند و نه علاقه ای. اما نه در دنیای لینوکس ! - توضیح خواهم داد که چرا میانگین کیفیت نرم افزار هانی که از جامعه لینوکس سرچشمه میگیرد، اینقدر بالا است.

خوب، آیا من به سرعت و در یک حرکت آتشی به نوشتن یک سرویس گیرنده POP3 جدید پرداختم تا با انواع موجود به رقابت بپردازد ؟ خیر ! من به دقت به ابزار های POP که در اختیار داشتم نگاه کردم، و از خودم سوال کردم " کدام یک به آنچه که من میخواهم نزدیکتر است ؟ ". به این دلیل که ؛

2. برنامه نویسان خوب، میدانند که چطور برنامه بنویسند. اما برنامه نویسان خبره، میدانند که چطور برنامه ها را بازنویسی کنند (و دوباره به کار بگیرند)

با اینکه ادعا نمیکنم که یک برنامه نویس خبره هستم، تلاش کردم تا این کار را تقلید کنم !. یک ویژگی مهم از برنامه نویسان خبره، تنبلی سازنده و مفید آنهاست. آنها به خوبی میدانند که شما رتبه A را نه بخاطر تلاشتان، که به خاطر نتیجه کارتان دریافت میکنید. و همواره آسانتر آن است که از یک بخش خوب از راه حل شروع به کار کرد تا اینکه از هیچ.

برای مثال، لینوس توروالدز حقیقتا تلاش نکرد تا سیستم عامل لینوکس را از ابتدا بنویسد. او با استفاده مجدد از کد ها و ایده هانی که از MINIX³، یک سیستم عامل کوچک شبیه یونیکس برای ماشین های 386، گرفته بود کار خود را شروع کرد. عاقبت، تمام کد های MINIX، حذف شد و یا بطور کامل از نو نوشته شد - ولی در زمانی که وجود داشت، ساختاری را برای کودکی که سرانجام لینوکس شد، تشکیل میداد.

با تفکری مشابه، من به دنبال یک ابزار POP گشتم که به خوبی نوشته شده باشد، تا از آن به عنوان پایه ای برای برنامه نویسی ام استفاده کنم.

¹ locke.ccil.org

² snark.thyrsus.com

³ MIni uNIX ؛ سیستم عملی که توسط اندرو تننهام، بر پایه یونیکس و با اهداف آموزشی نوشته شد.

سنت اشتراک کد در دنیای یونیکس همواره موافق استفاده مجدد از کد بوده است. (به همین دلیل است که پروژه GNU یونیکس را به عنوان پایه برای سیستم عامل اش برگزید؛ صرف نظر از شروط مهمی که در ارتباط با سیستم عامل وجود دارد) دنیای لینوکس این سنت را تا حد زیادی در کنار تکنولوژی اش نگه داشته است؛ تعداد زیادی از پروژه های باز متن، هم اکنون در دسترس هستند. لذا، صرف کردن زمان برای یافتن برنامه های خوب در دنیای لینوکس، نتیجه بهتری را نسبت به هر جای دیگری برای شما به ارمغان خواهد داشت.

و چنین اتفاقی برای من نیز رخ داد. با مواردی که قبلا پیدا کرده بودم، جستجوی دوم من 9 نتیجه را در برداشت - fetchpop, PopTart, get-mail, gwpop, pimp, pop- و perl, popmail و upop. در ابتدا تمرکز خود را بر روی fetchpop از سونگ-هانگ¹ قرار دادم و طرح کلی برنامه ام را بر روی آن پیاده سازی کردم که حاصلش، پیشرفت های مختلفی در این نرم افزار بود که نویسنده برنامه در نسخه 1.9 آنها را ترتیب اثر داد.

چند هفته بعد، به طور اتفاقی به برنامه popclient که کارل هریس² آن را نوشته بود، برخورد کردم و دریافتم که اشتباهی مرتکب شدم. هرچند برنامه fetchpop حاوی ایده های خوبی بود (برای مثال حالت daemon)، ولی تنها قادر به کار با POP3 بود و نسبتا ناشیانه نیز نوشته شده بود (سونگ-هانگ فرد زیرکی بود، اما برنامه نویس با تجربه ای نبود، این دو ویژگی، به خوبی در برنامه اش قابل مشاهده بود). کد کارل بهتر بود و کاملا حرفه ای و قدرتمند نوشته شده بود، اما برنامه اش فاقد برخی از ویژگی های مهم و زیرکانه fetchpop بود (که برخی از آنها را من نوشته بودم).

با همان برنامه کار میکردم یا برنامه ام را عوض میکردم؟ اگر برنامه ام را عوض میکردم، آنوقت تلاشهایی را که برای بهتر کردن برنامه قبلی انجام داده بودم را بی ارزش کرده بودم.

انگیزه اصلی برای تعویض برنامه ام پشتیبانی از چندین پروتکل در برنامه جدید بود. POP3 معمولترین پروتکل در میان پروتکل های سرویس دهنده post-office بود، اما تنها گزینه موجود هم نبود. برنامه Fetchpop و انواع مشابه دیگر از POP2، RPOP و یا APOP پشتیبانی نمیکردند، و من همچنین افکاری نه چندان جدی در ارتباط با اضافه کردن احتمالی پروتکل IMAP³ به برنامه نیز داشتم.

اما دلیل منطقی تری برای این نظریه که تعویض برنامه میتواند گزینه بهتری باشد، داشتم. چیزی که مدت ها پیش از لینوکس یاد گرفتم.

3. اگر بخواهی که چیزی را کنار بگذاری، در هر صورت سرانجام این کار را میکنی!⁴

یا به بیانی دیگر، در اکثر موارد واقعا متوجه مشکلات نخواهی شد تا برای یک بار هم که شده به دنبال راه حلی بگردی. در این صورت دفعه بعد، احتمالا اینقدر میدانی که چطور کارت

¹ . Sueng-Hong Oh

² . Carl Harris

³ . Internet Message Access Protocol ؛ جدیدترین و قدرتمندترین پروتکل post office . IMAP برخلاف POP به

کاربر امکان میدهد تا پیام ها را از چندین کامپیوتر دریافت کند.

⁴ . Fred Brooks, The Mythical Man-Month, Chapter 11

را درست انجام دهی. پس اگر میخواهی درست عمل کنی، تلاش کن که این کار را حد اقل یک بار تجربه کنی.

خوب (به خودم گفتم) تغییرات در برنامه fetchpop اولین تلاش من بود. حالا برنامه ام را عوض میکنم.

بعد از اینکه اولین مجموعه از بسته های نرم افزاری popclient را برای کارل هریس در تاریخ 25 ژوئن 1996 فرستادم، فهمیدم که او دیگر اصلا به برنامه popclient علاقه ای ندارد. کد برنامه با مشکلات کوچکی که داشت، کمی قدیمی نیز شده بود. من تغییرات بسیاری را در برنامه به وجود آوردم و سرانجام به این نتیجه رسیدیم که تصمیم منطقی این است که مسئولیت برنامه را بدست بگیرم.

واقعیت این بود که بدون اینکه متوجه باشم، پروژه به سرعت در حال پیشرفت بود. زمان زیادی میشد که دیگر بسته های نرم افزاری کوچک را برای نرم افزار نمینوشتم و به نگهداری و مدیریت کل برنامه میپرداختم؛ و همان زمان بود که این ایده به ذهنم خطور کرد که از این به بعد احتمالا میتوانم به تغییرات اساسی پردازم.

در فرهنگ نرم افزاری که اشتراک کد را تبلیغ میکند، این یک روش طبیعی برای نمو یک پروژه است. با این منطق کار میکردم :

4. اگر گرایش درستی داشته باشی، به مسائل جالبی برخورد میکنی

اما عقیده کارل هریس حتی ارزشمند تر بود. او به این نتیجه رسیده بود که :

5. اگر علاقه ات را نسبت به کار بر روی برنامه ای از دست دادی، در این صورت آخرین وظیفه ات این است که آن را به فرد شایسته ای بدهی

بدون اینکه هرگز خواسته باشیم در این باره حرفی بزنیم، من و کارل میدانستیم که هدفی مشترک در کسب بهترین راه حل برای نرم افزار داریم. تنها سوال باقیمانده برای هر دوی ما این بود که آیا من میتوانم از عهدی کار بر بیایم. زمانی که به این باور رسیدم، پاسخ او با بزرگواری همراه بود و نرم افزار را به من واگذار کرد. امیدوارم که در وظیفه خود به خوبی او عمل کرده باشم.

ý The importance of having users

ý اهمیت کاربران

و بدین گونه من نرم افزار **popclient** را به دست گرفتم و نتیجتاً، کاربران این نرم افزار را نیز بدست آوردم. وجود کاربران برای نرم افزار بسیار ارزشمند است، نه فقط به این خاطر که نشان دهنده این مطلب هستند که نیازی را برآورده میکنید، بلکه به این دلیل که آنها بیانگر عمل درست شما هستند. اگر درست رفتار کنید، آنها هم میتوانند برنامه نویس شوند.

سنت قدرتمند دیگر یونیکس که لینوکس آن را به اوج رسانده است، این است که اکثر کاربران این سیستم ها، یک هکر نیز هستند. از آنجا که کد منبع¹ برنامه در دسترس است، این امکان برای آنها وجود دارد که هکر های ماهرتری شوند. این مسئله میتواند در کاهش زمان عیب یابی نرم افزار بسیار موثر باشد. تنها با ایجاد کمی انگیزه و دلگرمی، کاربران شما میتوانند مشکلات موجود در نرم افزار را تشخیص داده، راه حل هایی را برای حل این مشکل پیشنهاد دهند و به بهبود برنامه نویسی خیلی سریعتر از حالتی که شما تنهایی به این کار میپرداختید، کمک کنند.

6. همکار پنداشتن کاربران، راحت ترین روش برای تسریع پیشرفت برنامه نویسی و کاراترین روش برای عیب یابی نرم افزار است.

قدرت کارانی این نکته، بسادگی مورد کم توجهی قرار میگیرد. در حقیقت، تقریباً تمامی ما که در دنیای باز متن به فعالیت میپردازیم، این نکته را که کاربران تا چه حد میتوانند در رساندن برنامه به سطح بالاتری از برنامه نویسی نقش داشته باشند و با پیچیدگی سیستم مقابله کنند را دست کم میگیریم؛ تا زمانی که لینوس توروالدز این تفاوت را به ما نشان داد.

در حقیقت، من فکر میکنم بزرگترین و زیرکانه ترین ابتکار لینوس تنها ساخت هسته سیستم عامل لینوکس نبود، بلکه بیشتر خلق مدل برنامه نویسی لینوکسی² بوده است. زمانی که این عقیده را در حضور او مطرح کردم، او خندید و سریعاً آنچه را که همیشه میگفت را تکرار کرد: " من اصولاً آدم تنبلی هستم که میخواهد از کارهایی که حقیقتاً دیگران انجام میدهند، استفاده ای ببرم ". تنبل همچون روباه، یا همانطور که رابرت هینلین³ گاهی میگفت، آنقدر تنبل که ممکن است بیازی.

با نگاهی به گذشته، یک نمونه برای روش ها و موفقیت لینوکس را میتوان در توسعه کتابخانه **GNU Emacs Lisp** و آرشیو کد های **Lisp** مشاهده کرد. در قیاس با روش ساخت کلیسایی هسته **Emacs C** و بسیاری از ابزار های دیگر **FSF**⁴، تکامل کدهای

1. Source Code .

2. Linux Development Model .

3. Robert Heinlein .

4. مخفف Free Software Foundation ؛ بنیاد نرم افزار های آزاد .

Lisp بسیار روانتر و کاربر-محورانه تر بود. ایده ها و پیش الگوها اغلب سه یا چهار بار قبل از رسیدن به حالت پایدار نهایی بازنویسی میشد. و همکاری های دورادوری که بوسیله اینترنت انجام میگرفت، معمول بود.

در واقع، تنها هک موفقیت آمیز من قبل از fetchmail ، احتمالاً Emacs VC mode بود؛ یک همکاری لینوکس-مانند که بوسیله ارسال email با سه نفر دیگر انجام گرفت. تنها یکی از آنها (ریچارد استالمن¹، پدید آورنده Emacs و موسس بنیاد نرم افزار آزاد یا FSF) را تا به امروز ملاقات کرده ام. آن برنامه مقدمه ای برای RCS ، SCCS و بعدها CVS از Emacs بود که ورژن عملیات کنترلی " one-touch " را ارائه میداد. برنامه مذکور از مدل کوچک و خامی بوجود آمد که فرد دیگری آن را نوشته بود. توسعه VC موفقیت آمیز بود؛ چراکه، برخلاف خود Emacs، این امکان برای کد Emacs Lisp وجود داشت که مراحل انتشار/تست/بهبودی را خیلی سریع طی کند.

یکی از اثرات جانبی پیش بینی نشده خط مشی FSF در تلاش برای قانونی همراه کردن کد برنامه با GPL² این است که بدین روش استفاده از مدل بازار برای FSF مشکل تر میشود؛ چرا که باید مجوزهای کپی رایت را برای هر همکاری در برنامه که بیش از بیست خط صورت گرفته، اجرا کنند تا برنامه تحت لیسانس GPL از رویارویی با قوانین کپی رایت مصون بماند. کسانی که از قانون کپی رایت تحت لیسانس های کنسرسیوم MIT X و BSD استفاده میکنند، چنین مشکلی نخواهند داشت؛ چرا که آنها تلاشی برای حفظ ارزشهایی که ممکن است افرادی را به مقابله تحریک کند، انجام نمیدهند.

Richard Stallman .¹
General Public License .²

ý Release early, Release Often

ý برنامه ات را زود و به تناوب منتشر کن

زود منتشر کردن برنامه و انتشار متناوب نسخه های بعدی آن، یک بخش مهم از مدل برنامه نویسی لینوکسی است. اکثر برنامه نویسان (از جمله خود من) معمولاً فکر میکنند که این روش، یک سیاست نادرست برای پروژه های بزرگ است؛ تنها به این دلیل که ورژن های ابتدائی، نسخه های پر اشکالی هستند و شما نمیخواهید که صبر کاربران به پایان برسد.

این طرز فکر، افراد را به سبک کلیسایی برنامه نویسی هدایت خواهد کرد. اگر هدف اصلی این باشد که کاربران کمترین خطای ممکن را مشاهده کنند، آنوقت چرا شما تنها نسخه های برنامه تان را هر شش ماه یک بار (یا حتی کمتر) انتشار میدهید و شدیداً برای رفع خطاها در زمان بین انتشار نسخه های برنامه تان کار میکنید. هسته Emacs C بدین روش ساخته شد. کتابخانه Lisp، عملاً اینطور نبود – به این خاطر که آرشیو های فعالی از Lisp در خارج از کنترل FSF وجود داشت، که شما میتوانستید نسخه های جدید و بهتری از برنامه را که مستقل از چرخه انتشار Emacs بود، پیدا کنید.

مهمترینشان، آرشیو Ohio State elisp بود که در بسیاری جهات از آرشیو های بزرگ امروز لینوکسی پیشی گرفته بود. اما تنها تعداد کمی از ما واقعا درباره آنچه که انجام میدهیم، درست اندیشیده ایم؛ و یا درباره ماهیت واقعی چنان آرشیوی با در نظر گرفتن مشکلات مدل برنامه نویسی کلیسایی FSF، تعمق کرده ایم. حدوداً سال 1992 بود که تلاشی جدی را در ارتباط با ادغام بسیاری از کدهای Ohio در کتابخانه رسمی Emacs Lisp، انجام دادم؛ که به مشکلات اساسی برخورد کردم و کاملاً شکست خوردم.

اما یک سال بعد، همونطور که لینوکس به طور گسترده ظاهر شد، به خوبی مشخص بود که چیزی متفاوت و البته بهتر در حال اتفاق افتادن است. سیاست توسعه نرم افزار باز لینوس کاملاً با ساختار کلیسایی در تضاد بود. آرشیو های sunsite و tsx-11 شروع به رشد کردند، توزیع های زیادی بوجود آمدند؛ و همه اینها بوسیله انتشارهای متناوب و غیر معمول هسته سیستمی، بوجود آمد.

لینوس با کاربرانش تا بیشترین حد ممکن، همانند همکارانش برخورد میکرد :

7. برنامه ات را زود و به تناوب منتشر کن و به مشتریان گوش بده

نوآوری لینوس در این زمینه زیاد نبود (مسائلی از این دست در دنیای یونیکس، به سنتی قدیمی تبدیل شده بود)؛ اما در ارتقاء این سنت تا حدی که بر پیچیدگی کاری که او در حال انجام آن بود، غلبه کند و با آن سازگار شود، بسیار ارزشمند بود. در روزهای نخستین (حدود سال 1991)، انتشار هسته ای جدید بیش از یک بار در هر روز، برای او عجیب نبود.

چراکه او همکاران برنامه نویس خود را بیش از هر فرد دیگری از طریق اینترنت به همکاری تشویق میکرد.

اما چطور این کار انجام شد؟ آیا میتوانستیم از آن الگو برداری کنیم، و آیا این مسئله به نبوغ منحصر به فرد توروالدز بستگی داشت؟

فکر نمیکنم. مسلماً لینوس یک هکر خیلی ماهر است (چند نفر از ما میتواند یک پروژه به بزرگی و جامعیت هسته سیستم عامل را انجام دهد؟). اما لینوس جهش نابغه وار عظیمی از خود نشان نداد. لینوس یک نابغه خلاق در طراحی، به نوعی که برای مثال ریچارد استالمن یا جیمز گوسلینگ¹ (از Java و NEWS) هستند، نیست (و یا حداقل، هنوز نشده است). بلکه، لینوس در نگاه من یک مهندس نابغه است؛ با حس ششمی برای اجتناب از خطا؛ و برنامه نویسی حرفه ای با مهارتی مثال زدنی برای یافتن بهترین و کوتاهترین مسیرها. در واقع، ساختار کلی لینوکس بر اساس این ویژگی ها پایه ریزی شده و بازتاب ذات هوشیار لینوس و روش ساده سازی طراحی اش است.

خوب، اگر انتشارهای پی در پی و استفاده ابزاری از رسانه اینترنت تصادفی نبوده باشد، بلکه بخش هایی از زیرکی و نبوغ مهندسی لینوس در کوتاه کردن مسیر به سوی هدفش باشد، در اینصورت او به دنبال چه بود و چه چیزی را از این سیستم طلب میکرد؟

با این فرض، پاسخ در خود سوال نهفته است. لینوس هکرها/کاربران خود را همواره ترغیب و از آنها قدردانی میکرد - ترغیب بوسیله نمایش دورنمایی از کاری که خودشان در آن سهیمند و احساس رضامایتمندی درونی حاصل، و قدردانی با نمایاندن (حتی روزانه) پیشرفتی که در کارشان حاصل میشود.

لینوس، دقیقاً قصد بیشینه کردن تعداد افراد و ساعاتی را داشت که صرف برنامه نویسی و عیب یابی میشدند؛ حتی اگر این امر به قیمت بی ثباتی در برنامه و خستگی ناشی از سختی حل مشکلی دشوار در برنامه، میشد. لینوس طوری رفتار میکرد که گویا به چنین درکی رسیده باشد:

8. با داشتن همکاران برنامه نویس و تست کننده های بتای زیاد، تقریباً هر مشکلی سریعاً پیدا شده و بوسیله فردی از افراد کاملاً برطرف میشوند.

یا خودمانی تر: "با وجود چندین نگاه جستجوگر، تمامی خطاها برطرف میشوند". چیزی که من آنرا "قانون لینوکس" نامیده ام.

شعار اصلی من هم در این زمینه این بود که هر مشکلی "بوسیله فردی پیدا میشود". لینوس تردید داشت کسی که مشکلات را درک کرده و آنها را برطرف میکند، لزوماً و یا حتی معمولاً همان کسی باشد که آن را پیدا کرده است. او میگفت: "فردی مشکلات را پیدا میکند و فرد دیگری آن را حل میکند. و من میگویم که پیدا کردن مشکل سخت تر است." اما نکته در اینجاست که هر دوی اینها سریعاً اتفاق می افتد.

¹ James Gosling .

فکر میکنم هسته اصلی تفاوت بین سبک بازار و سبک کلیسایی در اینجا نهفته است. از دیدگاه برنامه نویسی کلیسایی، اشکالات برنامه و مشکلات برنامه نویسی، مودی و دردرس ساز هستند. ماهها بررسی تعداد کمی از افراد که وقت خود را به این کار اختصاص داده اند، لازم است تا اطمینان حاصل کنید که تمامی خطاها را برطرف کرده اید. نتیجه این روش، وقفه های زمانی طولانی بین انتشار نسخه های مختلف نرم افزار، و یاسی اجتناب ناپذیر در حالتیکه که نسخه های دیر منتشر شده برنامه شما کارایی مورد نظر را ندارند، خواهد بود.

از طرف دیگر، در دیدگاه بازار، شما فرض میکنید که اشکالات برنامه ذاتاً اتفاقاتی سطحی هستند - و یا حد اقل، هنگامی که در معرض هزاران برنامه نویس مشتاق که منتظر دریافت نسخه ای جدید از برنامه هستند، قرار میگیرند، سطحی و ضعیف خواهند شد. بنابراین، شما برنامه خود را به تناوب منتشر میکنید تا برنامه تان بیشتر تصحیح شود و یکی از نتایج جانبی سودمند آن این است که اگر سهوا اشتباهی مرتکب شدید، چیز کمتری برای از دست دادن خواهید داشت.

و نکته در همین بود. همین و بس. اگر "قانون لینوکس" اشتباه میبود، آنوقت هر سیستمی به پیچیدگی هسته لینوکس، که بوسیله افراد بسیاری توسعه یافته بود، میبایست زیر فشار عدم هماهنگی ها و اشکالات اساسی کشف نشده، با شکست مواجه میشد. از سوی دیگر، اگر این قانون دست باشد، تنها کافیسیت که به شرح دلایل کم خطا بودن لینوکس پردازیم.

البته شاید دلیلی برای شگفتی هم وجود نداشته باشد، چراکه جامعه شناسان سالها قبل کشف کرده اند که میانگین نظر گروهی از افراد متخصص (و یا نادان)، قابل اعتماد تر از تک تک آنهاست. آنها چنین اصلی را "اثر دلفی"¹ نام گذاری کرده اند. پیداست که آنچه که لینوکس نشان داده است اینست که این مطلب حتی در عیب یابی نرم افزارهای چون سیستم عامل نیز قابل تعمیم است - اینکه اثر دلفی میتواند در آسان کردن پیچیدگی برنامه نویسی، حتی در حد و اندازه پیچیدگی هسته یک سیستم عامل نیز، موثر باشد.

من مدیون جف داتکی² هستم چرا که او قانون لینوکس را در قالب جمله "عیب یابی میتواند موازی و همزمان انجام شود" به زیبایی بیان کرد. جف به این نتیجه رسیده بود که هرچند عیب یابی نرم افزار نیازمند برنامه نویسان عیب یابی است که با برنامه نویسان مربوطه مشورت و رایزنی کنند، اما نیازمند هماهنگی بخصوصی بین عیب یاب ها نیست. بنابراین، به مضاعف شدن پیچیدگی و هزینه های مدیریتی که اضافه کردن برنامه نویسان را مشکل میکند، نخواهد انجامید.

عملاً، ضعف نظری عملکردی بعلت دوباره کاریهایی که در عیب یابی نرم افزار ممکن است رخ دهد، تقریباً هیچ گاه در دنیای لینوکس، در کارایی تاثیر منفی نخواهد داشت. یکی از نتایج "سیاست زود و و به تناوب منتشر کردن برنامه"، کمینه کردن چنین دوباره کاریهایی است که بوسیله انتشار مشکلات حل شده به سرعت قابل حل است.

بروکس³، حتی چنین عقیده تخمینی را با توجه به نظر جف داشت: "هزینه کلی نگهداری برنامه ای که به طور گسترده مورد استفاده قرار میگیرد، حدود چهل درصد و یا بیشتر از

¹ Delphi Effect .

² Jeff Dutky, <dutky@wam.umd.edu> .

³ Fred Brooks .

هزینه ایجاد آن است. جالب اینجاست که این هزینه شدیداً تحت تأثیر تعداد کاربران آن نرم افزار قرار دارد. کاربران بیشتر، اشکالات بیشتری را نیز پیدا میکنند. " (چیزی که همواره بر آن تأکید دارم)

کاربران بیشتر، اشکالات بیشتری را نیز پیدا میکنند، چرا که افزایش یافتن تعداد کاربران، به افزایش روش های مختلفی برای آزمایش نرم افزار خواهد انجامید. این نکته زمانی تقویت میشود که کاربران نرم افزار، برنامه نویسان کمکی برنامه مذکور نیز باشند. هر کدام رهیافتی جدید برای شناسایی مشکلات را با بینشی متفاوت و تحلیلی گوناگون ارائه میدهند، که نمایانگر زاویه ای جدید از مشکل خواهد بود. این تفاوت ها کاملاً بیانگر کارائی " اثر دلفی " هستند. در حالت خاصی از عیب یابی، این تنوع و گوناگونی به کاهش دوباره کاریها نیز خواهد انجامید.

بنابراین، افزودن تعداد تست کننده های حالت بتای نرم افزار، به کاهش پیچیدگی مشکلات موجود از دیدگاه برنامه نویسی نخواهد انجامید، بلکه موجب افزایش این احتمال خواهد شد که رهیافت شخصی ممکن است در حل مشکل توسط فرد مذکور، به کار آید.

لینوس زرنگی هم کرد. هر وقت که مشکلات جدی در برنامه وجود داشت، ورژن هسته لینوکس به نحوی شماره گذاری شده بود تا این امکان برای کاربران وجود داشته باشد که بتوانند آخرین نسخه " پایدار " برنامه را انتخاب کنند و یا اینکه ریسک کنند و با ترجیح ویژگی ها و امکانات نسخه جدید بر مشکلاتش، از نسخه جدید برنامه استفاده کنند. این تاکتیک، هنوز به طور رسمی توسط بسیاری از هکر های لینوکسی استفاده نشده است، اما به نظرم باید این کار صورت گیرد؛ این حقیقت که حق انتخابی در این بین وجود دارد، هر دو را جذاب تر خواهد کرد.

ی When is a Rose not a Rose ?

آ زمانی که برنامه دیگر آن برنامه قبلی نبود

با درک نحوه عملکرد لینوس و خلق یک فرضیه درباره دلایل موفقیت آن، تصمیمی هوشیارانه برای تست این تئوری در پروژه جدیدم گرفتم (که در قیاس با لینوکس، مسلماً بسیار ساده تر و کوچک تر بود).

اما اولین کاری که انجام دادم، سازماندهی دوباره و ساده سازی نرم افزار popclient بود. پیاده سازی کارل هریس¹ خیلی دقیق بود، اما نوعی پیچیدگی اضافی در قیاس با خیلی از برنامه نویسان C در برنامه اش به چشم میخورد. او با کد برنامه به عنوان بخش اصلی و ساختمان داده ها به عنوان پشتیبان کد برخورد میکرد. نتیجتاً، برنامه بسیار زیبا سازماندهی شده بود، اما نوع طرح ریزی ساختمان داده برنامه، روشی معمول نبود و تقریباً زشت سازماندهی شده بود (حداقل در قیاس با استانداردهای سطح بالای این هکر قدیمی LISP)

جدای این مسائل، دلیل دیگری برای بازنویسی برنامه درکنار بهبود طرح برنامه و ساختار ساختمان داده آن داشتم. در واقع، میخواستم برنامه را کاملاً درکش کنم. زیاد جالب نیست که مسئول درست کردن برنامه ای باشی که آن را درک نکرده ای !.

حدوداً در ماه اول، به پیروی از ساختار پایه ای طرح کارل پرداختم. اولین تغییر اساسی که در برنامه انجام دادم، اضافه کردن پشتیبانی از پروتکل IMAP به برنامه بود. این کار را بوسیله سازماندهی مجدد پروتکل ها در یک درایور عمومی و سه جدول متد (برای POP2، POP3 و IMAP) انجام دادم. این تغییر و تغییرات قبلی نمایانگر یک اصلی کلی هستند که خوب است برنامه نویسان آن را به خاطر بسپارند، به خصوص در زبان های برنامه نویسی از قبیل C که به طور پیش فرض با انواع پویا کار نمیکنند :

9. ساختمان داده ی خوب و برنامه ضعیف بسیار بهتر از برنامه خوب و ساختمان داده بی ارزش کار میکند.

فصل نهم از کتاب بروکس : " اگر میخواهی مرا گیج کنی، [برنامه ات] را به من نشان بده ولی [ساختمان داده] برنامه ات را پنهان کن. اما اگر [ساختمان داده] برنامه ات را به من نشان دهی، اغلب دیگر نیازی به دیدن [کد] برنامه ات ندارم؛ همه چیز واضح و مشخص است "

البته منظور او "فلوچارت ها" و " جداول " بود. البته با در نظر گرفتن حرکت سی ساله علمی و فنی/فرهنگی، این دو² تقریباً یکی هستند.

¹ Carl Harris .

² منظور نویسنده تبدیل « ساختمان داده و برنامه » به « فلوچارت و جداول » است.

در این زمان (اوائل سپتامبر سال 1996، حدوداً 6 هفته پس از شروع کار)، به این فکر افتادم که بهتر است که نام نرم افزار را تغییر دهم. پس از همه این تغییرات، برنامه مذکور تنها یک POP client ساده نبود. اما دچار تردید شدم، چرا که تا آن زمان هیچ کار کاملاً جدیدی در ساختار برنامه انجام نداده بودم. نسخه popclient من، هنوز هویت قبلی خود را داشت.

این تغییر بنیادی زمانی بوجود آمد که برنامه fetchmail قادر به ارسال کردن نامه های واکنشی شده به پورت SMTP شد. درباره اش خواهم گفت؛ اما قبل از آن : قبلاً گفتم که تصمیم داشتم که این پروژه را برای تست تنوری ام درباره آنچه که لینوس توروالدز انجام داده بود، به کار بگیرم. ممکن ست بپرسید چطور این کار را کردم ؟ به روش های زیر :

1. برنامه ام را زود و به تناوب منتشر میکردم (تکرارش تقریباً کمتر از هر ده روز یک بار نمیشد، اما در طی دوران اوج برنامه نویسی، هر روز این کار انجام میشد)
2. هر کس را که با من به نحوی درباره fetchmail در تماس بود را به لیست بتای خودم اضافه میکردم.
3. هر زمانی که نسخه جدیدی را انتشار میدادم، آگهی های زیادی را به لیست بتای خودم ارسال میکردم تا آنها را به مشارکت در پروژه تشویق کنم.
4. و به تست کننده های نسخه بتای نرم افزارم گوش میکردم، نظر آنها را درباره تصمیماتم جویا میشدم و هر زمان که آنها بسته های نرم افزاری و یا نظراتشان را برایم ارسال میکردند، از آنها تشکر میکردم.

نتیجه نهایی این کمک های کوچک، بسیار سریع اثر خود را نشان داد. از آغاز پروژه ، با گزارش های با ارزشی در ارتباط با خطاهای برنامه مواجه شدم که برنامه نویسان حاضرند به خاطرش جان بدهند ! ، که اغلب با راه حل های خوبی نیز همراه بود. من متفکرانه مورد نقد قرار گرفتم، نامه های بسیاری بدستم رسید و نظرات هوشمندانه ای نیز دریافت کردم. که همگی مرا به این نکته هدایت کرد :

10. اگر شما با تست کننده های نسخه بتای برنامه تان، چنان که آنها با ارزشترین منبع شما هستند، برخورد کنید، آنها به عنوان با ارزش ترین منبع شما به شما پاسخ خواهند داد.

یک نکته ارزشمند از موفقیت fetchmail، تعداد اعضای لیست نسخه بتای پروژه، یعنی دوستان fetchmail¹ است. در زمان نوشتن این مقاله، لیست مذکور 249 عضو داشت که هر دو یا سه هفته یکبار به تعداد اعضای آن اضافه میشود.

علا، با اصلاح نرم افزار در اواخر ماه مه سال 1997، لیست مذکور از آن تعداد زیاد اعضایش که به سیصد نفر میرسید، به دلیل جالبی تعدادی از اعضایش را از دست داد. خیلی از افراد از من میخواستند تا آنها را از لیست خارج کنم، چرا که برنامه fetchmail به حدی خوب عمل میکرد، که دیگر نیازی به بودن در لیست احساس نمیکردند ! شاید این یک بخش از چرخه حیات پروژه کاملی باشد که در سبک بازار نوشته شده است.

¹ . fetchmail-friends

ý Popclient becomes Fetchmail

ý نرم افزار popclient به fetchmail تبدیل میشود

تحول اصلی در پروژه زمانی صورت گرفت که هری هاچیسر¹ کد برنامه اش را در ارتباط با ارسال نامه به پورت SMTP ماشین سرویس گیرنده برایم فرستاد. من سریعاً به این نتیجه رسیدم که یک پیاده سازی درست از این ساختار تمامی روش های دیگر را به کنار خواهد زد.

هفته های زیادی را به سرو کله زدن با fetchmail پرداختم، تا طرح واسط برنامه که قابل استفاده اما بی قواره بود، را بهبود بخشم - برنامه ظاهری ناهنجار داشت، با گزینه های زیادی که به طور نامنظم در سراسر آن پراکنده بودند. مخصوصاً گزینه هائی که برای کپی برداری نامه واکشی شده به یک فایل در صندوق پستی² و یا یک دستگاه خروجی استاندارد تعبیه شده بودند، خاطرهم را می آزرده، اما دلایلش را نمیتوانستم درک کنم.

آنچه که در حین فکر درباره قابلیت ارسال SMTP به ذهنم خطور کرد، این بود که نرم افزار popclient سعی داشت تا کارهای زیادی را با هم انجام دهد. این برنامه به نحوی طراحی شده بود تا هم یک عامل انتقال پست الکترونیکی³ و هم یک عامل تحویل موضعی⁴ باشد. با قابلیت ارسال SMTP، میتوانست از حوزه کاری MDA خارج شده و یک MTA محض باشد که به سادگی sendmail⁵، نامه را به برنامه های دیگر برای تحویل موضعی تحویل میدهد؛

چرا تمامی پیچیدگی های پیکربندی یک MDA یا عملیات قفل و الحاق⁶ بر روی یک صندوق پستی در هم آمیخته شوند، در حالیکه پورت 25 تقریباً در تمامی پلت فرم هائی⁷ که از TCP/IP پشتیبانی میکنند، به این کار اختصاص داده شده است؟ بخصوص زمانی که این مسئله به معنی شباهت ساختاری نامه دریافتی با نامه معمولی یک فرستنده SMTP باشد؛ که درواقع همان چیزی است که ما میخواهیم.

درس های زیادی در این نکته وجود دارد. در ابتدا، ایده ارسال SMTP بزرگترین نتیجه ای بود که من از تلاش هوشیارانه ام برای شبیه سازی روش لینوس گرفتم. یکی از کاربران، این ایده فوق العاده را به من داد - تنها کاری که باید انجام میدادم این بود که مفاهیم را درک کنم.

1 . Harry Hochheiser .

2 . mail box .

3 . Mail Transport Agent (MTA) .

4 . Local Delivery Agent (MDA) .

5 . نوعی پروتکل ارسال نامه ساده (SMTP) که برای تحویل email نوشته شده است. این پروتکل مبتنی بر unix و باز متن است.

6 . lock-and-append .

7 . platform .

11. روش کارای بعدی برای داشتن ایده های خوب، کسب ایده های خوب از کاربران است. بعضی وقتها این روش دوم بهتر است.

با کمی دقت، سرعت خواهید فهمید که اگر کاملاً با خودتان درباره آنچه که به دیگران بدهکارید صادق باشید، دنیا با تمام بزرگی اش همانطور با شما رفتار کرده است که شما با هر تکه از نوآریتان که خلقت کرده اید؛ و این تنها شایسته بخشی از نبوغ درونی شماست. همه ما دیدیم که این مطلب چقدر برای لینوس موثر افتاد.

(وقتی که این مقاله را در کنفرانس Perl در ماه اوت سال 1997 ارائه دادم، لری وال¹ در ردیف اول نشسته بود. وقتی که به آخرین خط پاراگراف فوق رسیدم، او همچون مبلغ های مذهبی فریاد زد: " بگو، بگو برادر! ". تمامی حضار خندیدند، چرا که میدانستند که چنین چیزی برای سازنده perl نیز اتفاق افتاده بود.)

بعد از گذشت تنها چند هفته از اجرای پروژه در آن حال و هوا، تعریف و تمجید های مشابهی را نه از کاربرانم، بلکه از سایر افرادی که سخنانم به گوش آنها رسیده بود، دریافت کردم. من هنوز تعدادی از آن نامه ها را دارم. هر وقت حس میکنم که زندگیم بی ارزش بوده، یک نگاهی به آنها می اندازم (-).

اما دو درس پایه ای دیگر وجود دارد که برای هر نوع طراحی نرم افزار، مناسب است.

12. در اغلب اوقات، اکثر راه حل های ابداعی و برجسته از درک این نکته ناشی میشود که تحلیل شما از مسئله اشتباه بوده است.

زمانی من تلاش میکردم تا با ادامه توسعه نرم افزار popclient به عنوان یک MTA/MDA ترکیب شده، برنامه ام تمامی انواع تحویل محلی² را انجام دهد؛ در واقع میخواستم مسئله اشتباهی را حل کنم. ساختار برنامه fetchmail میبایست مجدداً مورد بررسی و تحلیل قرار میگرفت تا به یک MTA محض تبدیل شود.

زمانی که به یک بن بست در برنامه نویسی میرسید، - زمانی که فکر کردن به مرحله بعدی برایتان سخت میشود - آنوقت به دنبال جواب درست نباشید، بلکه به فکر یافتن سوال درست باشید. احتمالاً صورت مسئله نیازمند این است که از نو بیان شود.

با این وجود، مشکلم را از نو مورد بررسی قرار دادم. بوضوح مشخص بود که کار درست این است که؛ (1) قابلیت ارسال SMTP را در قالب یک درایور عمومی³ بگنجانم؛ (2) آن را حالت پیشفرض قرار دهم؛ و (3) سرانجام تمامی حالات دیگر تحویل را نادیده بگیرم، بخصوص گزینه های ارسال به فایل و ارسال به خروجی استاندارد.

بعضی اوقات در مورد اجرای بند 3 به تردید می افتادم، و از این میترسیدم که کاربران قدیمی popclient که به مکانیزم های تحویل دیگر عادت کرده اند، از این کار ناراحت شوند. در

¹ Larry Wall .

² Local Delivery Modes .

³ Generic Driver .

تئوری، این امکان برایشان وجود داشت که با روش های مشابه بتوانند مشکلشان را برطرف کنند. اما در عمل، این تحول میتوانست برایشان سخت و دشوار باشد.

اما زمانی که این کار را انجام دادم، بهبودی زیادی در روند کار ایجاد شد. مشکلاتی که در کد درایور وجود داشت به صفر رسید. پیکربندی برنامه بسیار ساده تر شد - دیگر برنامه بیهوده درگیر MDA و صندوق پستی کاربر نمیشد؛ همچنین دیگر نگرانی در ارتباط با پشتیبانی از سیستم عامل مورد استفاده در ارتباط با قفل کردن فایل وجود نداشت.

همچنین، این راه تنها روش برای جلوگیری از از دست دادن نامه ها نیز بود. اگر شما فایلی را به عنوان محل تحویل نامه هایتان در نظر میگیرفتید و دیسک پر میشد، آنوقت شما نامه تان را از دست میدادید. این مسئله هیچگاه با ارسال بوسیله SMTP اتفاق نخواهد افتاد، چرا که شنونده SMTP¹ شما تا زمانی که نامه شما تحویل داده نشود یا حداقل برای تحویل در زمانی دیگر اسپول² نشود، پیغام OK را برنمیگرداند.

کارایی نرم افزار نیز افزایش یافت (البته انتظار نداشته باشید که با یک بار اجرای نرم افزار به این نکته پی ببرید). فایده مهم دیگری که نمیتوان از آن به سادگی گذشت، کاهش محسوس تعداد صفحات راهنما بود.

بعداً، مجبور شدم تا قابلیت تحویل از طریق یک MDA محلی تعیین شده توسط کاربر³ را برای ایجاد امکان مدیریت بعضی از حالات نادری که در ارتباط با SLIP پویا بوجود می آمد، مجدداً به برنامه اضافه کنم. اما راه بسیار ساده تری را برای انجام آن پیدا کردم.

چطور؟ در کنار گذاشتن ویژگی های کهنه در حالی که میتوانید بدون آنها، همچنان با همان کارایی کار کنید، تردید نکنید. به گفته انتونی سینت-اگزوپری⁴ (زمانی که خلبان و طراح هواپیما بود و هنوز نویسنده کتاب های کلاسیک کودکان نشده بود) :

13. رسیدن به حد کمال (در طراحی) در حالیکه که چیز دیگری برای اضافه کردن وجود ندارد، بدست نمی آید، بلکه این مهم زمانی حاصل میشود که نتوان چیزی را از طرح مورد نظر کم کرد.

زمانی که برنامه شما هم بهتر و هم ساده تر میشود، بدانید که درست عمل کرده اید. و در پروسه انجام کار، طرح برنامه fetchmail هویت خاص خود را پیدا کرد، که کاملاً با نسخه قدیمی popclient متفاوت بود.

زمان تغییر نام برنامه فرا رسیده بود. طرح جدید بیشتر از آن برنامه popclient قدیمی به برنامه ارسال نامه الکترونیکی میخورد؛ هر دو به نوعی MTA بودند، اما برنامه popclient جدید متفاوت عمل میکرد. در نهایت و پس از دو ماه، نام برنامه را به fetchmail تغییر دادم.

¹ SMTP Listener .

² SPOOL (Simultaneous Peripheral Operation On-Line) .

³ user-specified local MDA .

⁴ Antonie de Saint-Exupery .

ý Fetchmail grows up

ý برنامه fetchmail رشد میکند

حال، من طرح برنامه ای ابداعی و منظم را در اختیار داشتم و به خوبی میدانستم که در کارم موفق بوده ام چرا که هر روز از آن استفاده میکردم، و لیست بتائی که صحت این ادعا را تائید میکرد. بتدریج حس میکردم که ابتکارهای کوچکی که ممکن است برای افراد کمی سودمند باشد، دیگر مرا ارضاء نمیکند. من در نوشتن برنامه ای دست داشتم که برای هر هکری که از یک نسخه یونیکس و یک کانکشن میل SLIP/PPP استفاده میکند، حقیقتاً مفید بود.

با قابلیت ارسال SMTP، برنامه به شدت از سایر رقیبان خود پیشی گرفته بود و به چنان برنامه قدرتمندی تبدیل شده بود که نه تنها سایر رقیبان خود را از راه بدر کرده بود، بلکه آنها را به بوته فراموشی سپرده بود.

حس میزنم که حتی نتوانید چنین نتیجه ای را متصور شوید. تنها زمانی میتوانید، به این درک برسید که چنان قدرت تحلیلی در طراحی نرم افزار داشته باشید که نتیجه کار برایتان محرز و طبیعی جلوه کند. تنها روش برای حصول چنین افکاری، با داشتن ایده های زیادی در ذهنتان حاصل میشود – و یا بوسیله یک نوع تیز بینی مهندسی وار، تا ایده های خوب دیگر افراد را در ماوراء نحوه نگرش آنها به چنگ آورید.

اندرو تننبام¹ ایده اصلی ساخت یک سیستم عامل کوچک محلی شبیه یونیکس را برای ماشین های 386 داشت تا از آن به عنوان ابزاری برای تدریس استفاده کند. لینوس توروالدز ایده Minix را به حدی ارتقاء داد که تصورش برای اندرو بعید بود – و آن را به چیزی ارزشمند و شگفت انگیز تبدیل کرد. به همین روش (هر چند در مقیاس کوچکتری)، من ایدهائی را از کارل هریس و هری هاچیسر دریافت کردم و آنها را به سطح بالاتری ارتقا دادم. ما مبتکرانی چنانکه مردم در تصورشان آن را به نابغه تعبیر میکنند، نبودیم. از سوی دیگر، اکثر پیشرفت های علوم مختلف و رشته های فنی مهندسی و نرم افزار نیز بوسیله مبتکرانی نابغه صورت نگرفته است؛ البته جدای برخی از اساطیر برنامه نویس.

نتایج حاصل مست کننده بود – در حقیقت، نوعی از موفقیت که هر هکری برای آن و به امید آن زندگی میکند! و همه این ها به این معنی بود که من میتوانستم سطح استاندارد هایم را حتی فراتر از این نیز ببرم. برای آنکه fetchmail را به آن خوبی که در خورش بود تبدیل کنم، مبیایست علاوه بر برآورده کردن نیازهای شخصی ام در نوشتن برنامه، جنبه های مورد نیاز دیگر افراد را هر چند که خارج از حوزه نیازهای من باشد، لحاظ میکردم. و البته همه این کار ها را به گونه ای انجام دهم که برنامه همچنان ساده اما قدرتمند باقی بماند.

¹ Andrew Tanenbaum .

اولین و مهمترین ویژگی که من بعد از درک این نکته به برنامه ام اضافه کردم، قابلیت پیشتیبانی از multidrop در برنامه ام بود – امکان واکنشی نامه از میل باکس هائی که تمامی نامه ها را برای یک گروه از کاربران نگهداری میکنند، و سپس هر نامه را به گیرنده مخصوص خود ارسال میکنند.

تصمیم اضافه کردن این قابلیت به برنامه، کمی به دلیل اصرار های پیاپی بعضی از کاربران، و بیشتر به این خاطر بود که فکر میکردم که اجبار حاصل در کار با آدرس های اینچینی میتواند خطاهای برنامه قبلی را کاهش دهد؛ و همینطور نیز شد. بررسی RFC 822 زمان زیادی برد، نه به این دلیل که هیچ بخشی از آن سخت بوده باشد، بلکه به این خاطر که حاوی توده ای از اطلاعات بهم مرتبط و جزئی بود.

اما نحوه آدرس دهی multidrop، نتیجه تصمیم یک طرح عالی را به خوبی نشان داد. اینجا بود که فهمیدم :

14. هر ابزاری در جای خود، مفید و سودمند است، اما یک ابزار واقعا سودمند آنچنان در استفاده منعطف است که حتی تصوش را هم نمیکند.

کاربرد غیر منتظره قابلیت multidrop در برنامه fetchmail، در کار با فهرست های پستی با نگهداشتن لیست، و نیز امکان بسط نامهای مستعار¹، در سمت سرویس گیرنده یک کانکشن SLIP/PPP بود. این امر به این معنی بود که شخصی که در پشت یه ماشین شخصی نشسته و از یک اکانت ISP استفاده میکند، قادر به مدیریت فهرست پستی خواهد بود، بدون اینکه نیازی به رجوع به فایل های نام های مستعار² ISP داشته باشد.

تغییر مهم دیگری که تست کننده های بتای نرم افزار خواستار آن شده بودند، پیشتیبانی از عملیات 8-بیتی MIME³ بود. انجام این کار بسیار ساده بود، چرا که حواسم به کد های 8-بیتی بود؛ نه به خاطر این که من احتمال درخواست این ویژگی را از قبل پیش بینی میکردم، بلکه بدلیل پیروی از اصلی دیگر :

15. وقتی که نرم افزار دروازه ای⁴، از هر نوعی میخواهد باشد، مینویسید، تلاش بسیاری کنید تا جریان داده شما تا حد امکان کوچک شود – و *هیچگاه* اطلاعات را دور نریزید، مگر اینکه گیرنده شما را مجبور به این کار کند.

اگر از این اصل پیروی نمیکردم، آنوقت پیشتیبانی از MIME هشت بیتی، سخت و مشکل ساز میشد. در آن صورت، مجبور بودم تا تمامی RFC 1652 را بخوانم تا چند بیت به ساختار تولید سرایند اضافه کنم.

برخی از کاربران اروپائی، مصرانه خواستار گنجاندن گزینه ای برای محدود ساختن تعداد پیغام هائی که در هر جلسه بازبایی میشود، شدند (که در این صورت، آنها میتوانند هزینه های گران شبکه های تلفنشان را کنترل کنند). من مدت زمان زیادی بر این موضوع افشاری

¹ alias (نام مستعار) ؛ نامی که برای هدایت پیام های پست الکترونیکی به شخص یا اشخاصی بر روی شبکه استفاده میشود.

² alias files .

³ Multipurpose Internet Mail Extensions .

⁴ Gateway Software .

و با درخواستشان مخالفت می‌کردم، و هنوز هم واقعا از آن تغییر دل خوشی ندارم. اما زمانی که شما برای کاربران جهان برنامه مینویسید، باید به نظرات مشتریان گوش کنید – این مسئله حتی در زمانی که آنها پولی به شما نمیدهند، هم صادق است.

ý A few more lessons from Fetchmail

ý درسهای دیگری از پروژه fetchmail

قبل از اینکه به نتایج کلی مهندسی نرم افزار بپردازیم، نکات مهم دیگری در ارتباط با پروژه fetchmail وجود دارد که شایسته بررسی و تحلیل است.

در سینتکس¹ فایل های rc بعضا کلمات کلیدی بی ارزش یافت میشود، که تماما توسط تجزیه گر² برنامه نادیده گرفته میشود. سینتکس انگلیسی-مانند³ بسیار خوانا تر از کلمات کلیدی مختصر و مرسوم است که عموما استفاده میشود.

این نکته زمانی به ذهنم خطور کرد که دریافتم که چقدر تعاریف فایل های rc، میتوانند در تشبیه یک زبان کوچک نقش داشته باشند (به همین دلیل بود که کلمه کلیدی server را در برنامه popclient به poll تغییر دادم)

حس کردم که هرچه واژه های اصلی برنامه به انگلیسی محاوره ای نزدیکتر باشد، کاربرد برنامه نیز ساده تر خواهد شد. هم اکنون، هر چند من از حامیان اصلی مکتب " از آن یک نوع زبان بساز " در طراحی نرم افزار هستم که نمونه های آن را میتوان در Emacs و HTML و خیلی از موتورهای دیتابیس⁴ یافت، معمولا علاقه زیادی به استفاده از اصطلاحات روزمره انگلیسی نیز ندارم.

برنامه نویسان قدیمی به استفاده از واژه هائی کاملا دقیق، مختصر و بدون هیچ گونه زیاده گویی گرایش دارند. این سبک، میراثی فرهنگی از زمانی است که منابع کامپیوتری گران بودند و لذا مراحل عمل تجربه میبایست تا حد امکان ارزان و ساده میبود. در این صورت طبیعی بود که زبان انگلیسی با افزونگی در حدود 50%، یک مدل نامناسب بوده باشد.

این موضوع، دلیل من برای اجتناب عادت گونه ام از واژه های محاوره ای انگلیسی نبود؛ به آن اشاره کردم تا اشتباه بودنش را گوشزد کنم. با هسته و سیکل هائی که ارزان تمام شوند، اجازه حاصل به خودی خود یک فرجام نیست. این روزها مهمتر این است که زبان برنامه نویسی برای انسان راحت تر باشد تا اینکه برای ماشین ارزانتر تمام شود.

هرچند، دلایل بسیاری برای محتاط بودن وجود دارد. یکی، هزینه پیچیدگی در مرحله تجزیه است – شما قطعا نمیخواهید به جانی برسید که مجموعه کاملی از اشکالات و عوامل مختلف پریشانی کاربر، برنامه تان را فرا بگیرد. دیگری این است که تلاش برای تبدیل سینتکس زبان به حالتی محاوره ای و انگلیسی مانند، اغلب نیازمند این است که انگلیسی مورد

1 . syntax

2 . parser

3 . English-Like Syntax

4 . DataBase Engines

استفاده تا حد زیادی برای همسو شدن با هدف مورد نظر از قالب اصلی خود خارج شود؛ آنقدر که این تشابه ظاهری به زبان طبیعی مورد نظر، به اندازه همان سینتکس سنتی گنج کننده خواهد بود (این مطلب را در بسیاری از به اصطلاح " نسل چهارمی " ها و زبان های پرس و جوی دیتابیس های تجاری ¹ میبینید)

میتوان گفت که سینتکس مدیریتی برنامه fetchmail از این مشکلات بدور بود، چرا که دامنه ادبیاتی آن محدود و اندک بود و با ساختار یک زبان همه منظوره فاصله داشت. مطالبی که به کاربر میرساند، ساده بود و حداقل خیلی پیچیده نبود، لذا پتانسیل کمی برای پریشانی و گمراهی کاربر در تقابل ذهنی بین مجموعه ای کلمات روزمره انگلیسی و زبان مدیریتی واقعی نرم افزار، داشت. فکر میکنم درس بزرگی در این نکته نهفته است :

16. وقتی فاصله بین ادبیات مورد استفاده برنامه شما و ادبیات محاوره ای زیاد است، اصطلاحات خودمانی میتوانند مفید واقع شوند .

نکته دیگر امنیت در تاریکی است ! بعضی از کاربران fetchmail از من میخواستند تا نرم افزار را به گونه ای تغییر دهم تا پسورد ها را به شیوه رمز گذاری شده در فایل های rc به نحوی ذخیره کند که از دسترسی دیگر افراد مصون بماند.

من چنین کاری را انجام ندادم، چرا که عملاً این کار امنیت را افزایش نمیداد. هر کسی که مجوز های خواندن فایل های rc شما را داشته باشد، توانایی اجرای برنامه شما را نیز همانند شما خواهد داشت – و اگر به دنبال پسورد شما باشند، با استفاده از رمز گشاهای خاصی قادر به یافتن پسورد شما خواهند بود.

نتیجه رمز گذاری پسورد در برنامه، تنها به ایجاد حس اعتمادی واهی در افرادی که این کاره نیستند، می انجامید. قانون کلی این است که :

17. کل امنیت یک سیستم به اسرار آن است . از کم کاری در بحث امنیت حذر کنید .

ی Necessary preconditions for the bazaar style

ی پیش نیاز های لازم برای سبک بازار

منتقدان و شنوندگان اولیه این مقاله، همگی سوالاتی را درباره پیش نیاز های لازم برای یک پیاده سازی موفق از سبک بازار در توسعه نرم افزار مطرح میکردند، که مباحثی نظیر صلاحیت های سرپرست پروژه و سبک برنامه نویسی درحالتیکه فردی در اجتماع سعی بر ایجاد انجمنی از کمک-برنامه نویسان¹ را دارد، را شامل میشد.

کاملاً مشخص است که کسی نمیتواند در سبک بازار، برنامه نویسی را از ابتدا شروع کند. هر فرد، قادر به تست برنامه، اشکال زدایی و بهبود نرم افزار است، اما انجام یک پروژه از ابتدا و کار انفرادی بر روی آن در مدل بازار بسیار سخت خواهد بود. لینوس این کار را امتحان نکرد، من هم چنین کاری نکردم. جامعه برنامه نویسانِ پاکرفته شما نیازمند پروژه ای قابل اجرا و تست است تا با آن سرگرم شود.

زمانی که شما شروع به ایجاد انجمنی اینچینی میکنید، آنچه که برای ارائه نیازمند آن هستید، قوی است که دیگران آن را باور داشته باشند. ممکن است که برنامه تان چنان که باید درست کار نکند؛ ممکن است منطقتان خام، کدتان پراشکال، برنامه تان ناقص، و مستند سازیتان ضعیف باشد. چیزی که نباید در آن کوتاهی کنید متقاعد کردن درونی همکاران برنامه نویستان است که میتواند برنامه تان را در آینده ای نزدیک به برنامه ای کارا و خوب تبدیل کند.

لینوکس و fetchmail، هر دو با طرحهای پایه ای قدرتمند و جذاب، پا به جامعه گذاشتند. افراد بسیاری که پس از ارائه این مقاله به مدل بازار می اندیشند، یه این نکته مهم پی برده اند؛ سپس از کل مطالب به این نتیجه میرسند که سطح بالای بینش طراحی نرم افزار و هوشمندی در سرپرستی پروژه، از هم تفکیک ناپذیرند.

اما لینوس، طرح برنامه خود را از یونیکس گرفت. من طرح خودم را از نسخه های اولیه popclient گرفتم (هر چند تغییرات زیادی کرد، و در قیاس با لینوکس، خیلی بیشتر در ارتباط با آن گفته شد). در این صورت، آیا سرپرست/هماهنگ کننده یک پروژه در سبک بازار واقعا نیازمند داشتن استعداد استثنائی طراحی نرم افزار است، یا اینکه او میتواند از قدرت استعداد طراحی دیگران بهره ببرد ؟

فکر میکنم، زیاد مهم نیست که هماهنگ کننده توانایی خلق طرح هائی با نبوغ استثنائی را داشته باشد، بلکه آنچه که مهم است این است که هماهنگ کننده پروژه قادر باشد تا ایده های خوب را از دیگر ایده ها تشخیص بدهد.

¹ . co-developer

هر دو پروژه لینوکس و fetchmail گواهی بر این ادعا هستند. لینوس، نه در مقام یک طراح مبتکر برجسته (همانطور که پیشتر بحث شد)، ابتکار فوق العاده ای را از خود برای تشخیص طرح های خوب و یکپارچه کردنشان با هسته لینوکس نشان داد. و من نیز پیش از این، به شرح چگونگی شکل گیری یک ایده قدرتمند طراحی در برنامه fetchmail (ارسال SMTP) که متعلق به فرد دیگری بود، پرداختم.

اولین شنوندگان این مقاله، از من با اشاره به این که ابتکار طراحی را در پروژه های تحت بازار راحت جلوه داده ام، تعریف و تمجید کردند؛ به این دلیل که خیلی از آنها را خودم دارا هستم و در نتیجه این مسائل برایم سهل و آسان است. شاید تا حدی حق با آنها باشد؛ فاز طراحی (در قیاس با برنامه نویسی و یا عیب یابی) مطمئناً قویترین مهارت من است.

اما مشکل زیرک و مبتکر بودن در طراحی نرم افزار این است که این خصوصیت بعد از چندی به یک عادت تبدیل میشود - شما خودتان همه کارها را میکنید و زمانی که مجبور میشوید که آنها را به فرمی ساده و قدرتمند در آورید، گیج میشوید. پروژه هائی را بیاد می آورم که در آنها به مشکل برخوردیم، چرا که مرتکب این اشتباه شدیم؛ اما حواسم بود که این اشتباه را در fetchmail مرتکب نشوم.

بنابراین، به این باور رسیدم که پروژه fetchmail تاحدی موفقیت آمیز بود، چرا که زیرکی ام را کنترل کردم؛ این استدلال (حداقل) در تضاد با ابتکار در طراحی، برای پروژه هائی که سبک بازار اجرا میشوند، لازم و ضروری است. و لینوکس را در نظر بگیرید. فرض کنید لینوس توروالدز سعی میکرد نوآوری های بنیادین در طرح سیستم عامل را در طی توسعه پروژه اش را خودش انجام دهد؛ در این صورت آیا این احتمال وجود داشت که هسته حاصل از این کار، به همان پایداری و موفقیتی باشد که اکنون در دست ماست ؟

البته یک سطح پایه ای مشخصی از توانایی طراحی و برنامه نویسی ضروری است، اما من معتقدم تقریباً هر کسی که بطور جدی درباره اجرای کاری در سبک بازار می اندیشد، از آن سطح مینیمم گفته شده بالاتر است. بازار داخلی جامعه باز متن، فشار زیرکانه ای را بر روی افراد اعمال میکند تا به دنبال کارهائی در برنامه نویسی که توانائی ادامه آن را ندارند، نروند. تا کنون، گویا این قضیه صادق بوده و به خوبی عمل کرده است.

مهارت دیگری وجود دارد که به طور مستقیم با توسعه نرم افزاری در ارتباط نیست؛ فکر میکنم اهمیتش به اندازه زیرکی در طراحی پروژه های سبک بازار باشد - و ممکن است مهمتر باشد. هماهنگ کننده و یا مدیر یک پروژه در سبک بازار میبایست از مهارت های ارتباطی خوب در برخورد با افراد، برخوردار باشد.

این مسئله بدیهی است. برای ساخت یک انجمن توسعه نرم افزار، آنچه که نیاز دارید این است که افراد را جذب کنید؛ آنها را به کاری که میخواهید انجام دهید، علاقمند سازید؛ و از بابت کاری که انجام میدهند، از آنها تشکر کنید؛ سرو صداها و تبلیغات فنی پروژه، روشی قدیمی برای انجام این کار است؛ اما این جدا از موضوع بحث است. خصوصیات خود شما نیز در پروژه نقش اساسی دارد.

این تصادفی نیست که لینوس یک مرد نازنین و دوست داشتنی است که سبب میشود تا دیگران هم مانند خود او شوند و بخواهند که به او کمک کنند. این یک اتفاق نیست که من

یک فرد برون گرای فعال هستم که از کار کردن با دیگران لذت میبرم و مهارت زیادی در طنز دارم. برای اینکه مدل بازار پا برجا بماند، اندک توانائی شما در مجذوب ساختن افراد میتواند بسیار موثر واقع شود.

ý The social context of open-source software

ý مفهوم اجتماعی نرم افزار باز متن

حقیقت دارد که : بهترین ها از راه حل های شخصی برای مشکلات هر روزه مبتکران شروع میشوند؛ و سپس به طور گسترده مورد استفاده قرار میگیرند، چرا که مشکل فوق، میتواند مشکل گروه گسترده ای از کاربران باشد. این ما را به دلیل قانون 1 باز میگرداند، که اگر بخواهیم آن را به شیوه ای بهتر بیان کنیم، میتوانیم بگوئیم :

18. برای حل یک مسئله جالب، از یافتن مسئله ای شروع کنید که برایتان جالب باشد

برای من، کارل هریس و ورژن های اولیه popclient جرقه های اولیه شروع بود، که نتیجه اش برنامه fetchmail شد. اما درک این نکته نیازمند گذشت زمان نسبتا زیادی است. نکته جالب توجه، این است که بجا بودن انتخاب سیر حرکت لینوکس و fetchmail برای بررسی، ما را به مرحله بعدی هدایت میکند – سیر تکاملی نرم افزار در بین انجمن های زیاد و فعالی از کاربران و برنامه نویسان.

فرد بروکس¹ در کتابش²، نشان داد که زمان برنامه نویسی قابل تعویض نیست، و اضافه کردن برنامه نویسی جهت تسریع در پروژه ای که به تاخیر افتاده است، موجب بیشتر به تاخیر افتادن آن نیز خواهد شد. او به بحث در این مورد پرداخت که پیچیدگی و هزینه های ارتباطی یک پروژه با تعداد برنامه نویسان آن، هنگامی که نمودار پیشرفت پروژه خطی است، نسبت مستقیم دارد. این ادعا "قانون بروکس"³ نام گرفت و به عنوان قانونی بدیهی پذیرفته شد. اما اگر قانون بروکس همه ماجرا بود، در این صورت شکل گیری لینوکس غیر ممکن میشد.

تنها چند سال بعد، جerald وینبرگ⁴ در کتاب روانشناسی برنامه نویسی کامپیوتر⁵، بیان کرد که قانون بروکس باید به طور اساسی تصحیح شود. در بحث " برنامه نویسی با دیگران"⁶، وینبرگ نشان داد که در حالاتی که برنامه نویسان خودشان تمامی کارها را بر روی کدشان انجام نمیدهند، و دیگر افراد را به پیدا کردن مشکلات برنامه و بهبودی در نرم افزار تشویق میکنند، چنان پیشرفتی در برنامه ایجاد میشود که نمونه اش را نمیتوان در هیچ مدل دیگری یافت.

میتوان گفت که واژه های انتخابی وینبرگ، تحلیلش را از پذیرشی که شایسته و سزاوار آن بود، بر حذر داشت – مثلا بکار بردن واژه " egoless" برای هکر های اینترنتی، برای

¹ Fred Brooks .

² The Mythical Man-Month .

³ Brooks's Law .

⁴ Gerald Weinberg .

⁵ The Psychology of Computer Programming .

⁶ egoless programming .

برخی خنده دار بود. اما من فکر میکنم که استدلال او، امروزه بیش از زمان دیگری به واقعیت نزدیک است.

تاریخچه یونیکس باید قاعدتا افکار ما را برای آنچه که باید از لینوکس یاد بگیریم، آماده کرده باشد (و تجربه ای که من در قالب کوچکتری با الگو برداری تعمدی از روش لینوس، شرح دادم). یعنی، زمانی که برنامه نویسی را به عنوان فعالیتی اساسی به طور مجزا در نظر بگیریم، ابتکار های بسیار بزرگی از هدایت پروژه به توجه و قدرت خلاق جوامع برنامه نویسی حاصل میشود. برنامه نویسی که تنها از قدرت فکر خود در یک پروژه بسته استفاده میکند، در قیاس با برنامه نویسی که به خوبی میداند که چطور زمینه یک پروژه باز و تکاملی را که عیب یابی و پیشرفت آن به کمک صدها برنامه نویس دیگر انجام میشود، فراهم کند، شکست خورده است.

اما دنیای سنتی یونیکس، بنا به دلایل متعددی مانع از به کمال رسیدن این خط مشی شد. از جمله این دلایل میتوان به محدودیت های مجوز های گوناگون، اسرار تجاری، و سود های تجاری اشاره کرد. دیگری این بود که اینترنت آن زمان به وسعت امروز نبود.

پیش از اینکه اینترنت ارزان شود، تنها چند انجمن محدود با سبک برنامه نویسی "egoless" وار¹ وینبرگ به فعالیت میپرداختند، و یک برنامه نویس به راحتی میتوانست نظر بسیاری از افراد حرفه ای و برنامه نویسان دیگر را جلب کند. آزمایشگاه Bell، MIT AI، و UC Berkeley، به مهد نوآوری های افسانه ای و بزرگ تبدیل شده بودند.

لینوکس اولین پروژه ای بود که تلاشی هوشیارانه و موفق را در استفاده از تمام جهان به عنوان منبع نوآوری اش انجام داد. فکر نمیکنم که همزمانی و همگامی دوران رشد لینوکس و WWW تصادفی بوده باشد، و اینکه لینوکس دوران کودکی خود را در حدود سالهای 1993-1994 همزمان با گسترش صنعت سرویس دهندگان اینترنت و انفجار تقاضا برای اینترنت سپری کرد. لینوس اولین کسی بود که چگونگی استفاده از اصول جدیدی را آموخت که فراگیر شدن اینترنت آنها را ممکن ساخته بود.

با این حال که اینترنت ارزان قیمت نیازی ضروری برای رشد مدل لینوکس بود، اما فکر میکنم که این امر به خودی خود کافی نبود. عامل حیاتی دیگر، پیشرفت سبک رهبری پروژه و مجموعه روشهای همکاری بود که این امکان را به برنامه نویسان میداد تا نظر برنامه نویسان دیگر را جلب کنند و از این رسانه بیشترین استفاده را ببرند.

اما این سبک رهبری و روشهای همکاری چیستند؟ این دو را نمیتوان نتیجه قدرت ارتباطات دانست - و حتی اگر بتوان چنین تصور کرد، رهبری با اجبار، چنین نتیجه ای که امروز شاهدش هستیم، در بر نخواهد داشت. وینبرگ از کتاب زندگینامه پایتر الکسیویچ کروپتکین²، آنارشیست قرن نوزدهم روسیه، با عنوان "خاطرات یک انقلابی"¹، مطالبی را در تانید این موضوع نقل قول کرده است:

¹ ego در زبان انگلیسی به معنی ضمیر، نفس، خود فرد میباشد؛ منظور از egoless programming؛ همانطور که پیشتر گفته شد، استفاده از کمک و همکاری دیگران در پروژه و به نوعی "برنامه نویسی با دیگران" است. اما برای رساندن اصل مطلب و نبود واژه فارسی که دقیقاً مطابق با معادل انگلیسی آن باشد، از همان واژه انگلیسی در متن استفاده شده است.

² Pyotr Alexeyvich Kropotkin.

" پس از گذراندن دوران کودکی در خانواده ای فقیر، مثل هر مرد جوان دیگری در آن دوران، وارد زندگی واقعی شدم؛ با باور لزوم حرف شنوی، فرماتپذیری، تحمل سرزنش و تنبیه، و ... اما طولی نکشید که تصمیم گرفتم تا این وضعیت را عوض کنم و با اربابان مقابله کنم؛ و در شرایطی که کوچکترین اشتباهی، عواقب وخیمی برای مرتکب شونده آن به همراه داشت، تصمیم گرفتم تا تفاوت بین پیروی از قانون حرف شنوی با پیروی اصولی که دیگر مردم با آن موافق بودند، را حس کنم. اولی، بیشتر مناسب رژه های نظامی بود که هیچ تقارنی ساختار زندگی نداشت؛ و رسیدن به این هدف تنها با تلاشی سخت در همسو شدن اراده عمومی ممکن بود. "

عبارت " تلاشی سخت در همسو شدن اراده عمومی " کاملاً گویاست که پروژه ای همانند لینوکس به چه چیزی نیاز دارد - و " قانون حرف شنوی " را به هیچ وجه نمیتوان در میان داوطلبانی که در بهشت آناشویست ها !! که ما آن را اینترنت مینامیم، بسر میبرند، بکار برد. برای اینکه خوب به کار و رقابت بپردازید، هکرهایی که میخواهند پروژه های گروهی را رهبری کنند، باید چگونگی جذب افراد و تزریق انرژی و شوق به کار را در انجمن های علاقمند را در حالی که بطور سربسته توسط " اصل توافق " کروپتکین به آن اشاره شد، یاد بگیرند. باید این مسائل را یاد بگیرند تا بتوانند از قانون لینوس استفاده کنند.

پیشتر، از " اثر دلفی " به عنوان تفسیری قابل قبول برای قانون لینوکس، یاد کردم. اما قیاس های قدرتمند دیگری نسبت به سیستم های تطبیقی در زیست شناسی و اقتصاد یافت میشود که به این موضوع اشاره دارند. دنیای لینوکس در بسیاری از جهات همانند یک بازار آزاد یا یک اکولوژی عمل میکند؛ یک مجموعه از عوامل خودخواه در تلاش برای هر چه بیشتر کردن سودی هستند که در فرایند انجام کار، نهایتاً به یک آرایش خود بخود اصلاح شده ای خواهد انجامید که بسیار کارتر و حرفه ای تر از هر پروژه متمرکزی است. اینجاست که باید به دنبال نتایج " اصل توافق " بگردیم.

منظور از تلاش برای " عملکرد بر اساس منفعت " در هکر های لینوکسی، تنها به معنی کلاسیک اقتصادی آن نیست، بلکه به معنی حس معنوی حاصل از رضایتمندی درونی آنها و شهرتی است که در میان هکر های دیگر بدست می آورند. (ممکن است این انگیزه، نوع دوستانه عنوان شود، اما نمیتوان انکار کرد که نوع دوستی نیز به خودی خود، نوعی از رضایتمندی درونی است که به فرد نوع دوست دست میدهد). فرهنگ های اختیارگرا که از این روش استفاده میکنند، کم نیستند. یکی از آنها که من مدت زیادی است در آن مشارکت دارم، گروه هواداران داستان های علمی-تخیلی است، برخلاف آنچه که هکرها آنرا صریحاً به " egoboo " (افزایش شهرت کسی در میان دیگر هواداران)² می شناسد، محرک اصلی فعالیت های داوطلبانه آن است.

لینوس، با درست قرار دادن خویش به عنوان دروازه بان پروژه ای که توسعه اش اکثراً توسط دیگران انجام گرفت، و پرورش علاقمندی در پروژه تا زمانی که پروژه روی پای خود ایستاد، نشان داد که درک زیرکانه ای از " اصل توافق عمومی " کروپتکین بدست آورده

1 . Memoirs of a Revolutionist .

2 . egoboo : the enhancement of one's reputation among other fans .

است. این دیدگاه شبه اقتصادی از دنیای لینوکس، ما را قادر میسازد تا ببینیم که چطور این توافق عملی شده است.

میتوانیم به روش لینوس به عنوان روشی برای خلق بازاری کارا در " egoboo " بنگریم – تا خودپسندی تک تک هکرها را تا آنجا که ممکن است بهم متصل سازد تا با دشواریهای مبارزه کند که تنها بوسیله همکاریهای تنگاتنگ و مداوم، ممکن بود. در پروژه fetchmail (البته در مقیاس کوچکتر) نشان دادم که الگوبرداری از روش او به نتایج خوبی خواهد انجامید. شاید حتی من این روش را هوشیارانه تر و سیستماتیک تر از او هم بکار برده باشم.

افراد زیادی (بخصوص آنهایی که اصولاً نسبت به بازارهای آزاد بدبین هستند)، انتظار یک فرهنگ خودخواهانه یک طرفه ای دارند که از هم گسیخته، شخصی، بی فایده، مرموز و خصومت آمیز باشد. اما این انتظار، کاملاً توسط (تنها برای اینکه مثالی ذکر کنم) تنوع گنج کننده، کیفیت و عمق مستندات لینوکس مردود است. اگر برنامه نویسان از مستند سازی متنفرند، پس چگونه است که هکر های لینوکسی حجم انبوهی از مستندات را تولید کردند؟ بدیهی است که بازار آزاد لینوکس در egoboo بهتر در تولید محصولات ارزشمند تلاش میکند؛ با روشی متفاوت از آنچه در بخش های مستند سازی پر هزینه تولید کننده های نرم افزار های تجاری رخ میدهد.

هر دو پروژه هسته سیستم عامل لینوکس و fetchmail نشان داد که با تشویق درست خیلی از هکر های دیگر، یک برنامه نویس/هماهنگ کننده قدرتمند میتواند بواسطه اینترنت از وجود خیلی از همکاران برنامه نویس دیگر سود برد، بدون اینکه پروژه اش را به هرج و مرج و بی نظمی بکشانند. بنابراین، با توجه به قانون بروکس ، من میگویم :

19. اگر هماهنگ کننده پروژه، از رسانه ای همچون اینترنت استفاده کند، و بداند که چگونه بدون استفاده از جبر به رهبری پردازد، مطمئناً روش رهبری بهتر میشود.

فکر میکنم که آینده نرم افزار های باز متن تا حد زیادی به افرادی تعلق خواهد داشت که میدانند چطور تجربه لینوس را تکرار کنند، کسانی که به روش کلیسائی پشت میکنند و بازار را در آغوش میگیرند. منظورم این نیست که قدرت فردی و نبوغ دیگر اهمیت ندارد، بلکه، فکر میکنم که پیشگامان نرم افزار های باز متن، کسانی خواهند بود که از قدرت فردی و نبوغ شخصی، کار خود را شروع میکنند و سپس، آن را بوسیله ساخت داوطلبانه و موثر انجمن هائی از علاقمندان، تقویت میکنند.

و شاید، این امر تنها منحصر به نرم افزار های باز متن نباشد. برنامه نویسی با اهداف غیر تجاری، که در دریای استعداد جامعه لینوکس قرار میگیرد، میتواند بر مشکل خود غلبه کند. به سختی بتوان از عهده کرایه بیش از دویست نفر که در پروژه fetchmail به من کمک کردند، برآمد.

احتمالاً، در انتها، جنبش باز متن پیروز خواهد بود؛ نه به این دلیل که تشریک مساعی از نظر اخلاقی کاری پسندیده است یا به این خاطر که " احتکار نرم افزار " اخلاقاً مطرود است (با فرض اینکه نکته آخر را باور دارید، همانطور که من و لینوکس با آن مخالفیم)، بلکه تنها به این خاطر که دنیای تجارت نمیتواند در یک مسابقه فنی در حال پیشرفت، در جدال با

انجمن های باز-متنی که از زمان متخصصان بیشتری برای حل مشکلاتش بهره میبرد، پیروز شود.

ý Epilog : Netscape embraces the bazaar

ý نت اسکپ¹ به بازار میپیوندد

حالت عجیبی به آدم دست میدهد زمانی که حس کنی در حال رقم زدن تاریخ هستی ..

در 22 ژانویه سال 1998 ، حدودا هفت ماه پس از اولین انتشار این مقاله، شرکت نت اسکپ ، اعلام کرد که قصد دارد کد منبع برنامه netscape communicator را انتشار دهد. من از این ماجرا تا روز انتشار آن خبری نداشتم.

اریک هان²، نایب رئیس و رئیس بخش تکنولوژی در نت اسکپ، بعدا نامه ای مختصر به شرح زیر برای من نوشت : " از طرف تمامی افرادی که در نت اسکپ کار میکنند، میخواستم در درجه اول از شما برای کمکی که به ما کردید تشکر کنم. تفکر و مقاله شما الهام اصلی برای تصمیم ما بود. "

در هفته های بعد، من مرتبا به دعوت نت اسکپ به silicon valley دعوت میشدم تا در کنفرانس های استراتژیک با برخی از مدیران بلند پایه و مهندسان فنی آنها شرکت کنم. ما به تدوین استراتژی انتشار کد منبع نت اسکپ و مجوز مربوطه اش پرداختیم و در مورد طرح های بلند مدت و اثرات آن در جامعه باز-متن نیز گفتگو کردیم. الان که من در حال نوشتن این پس گفتار هستم، کمی زود است که بخواهم به شرح آن پردازم، اما جزئیات آن به زودی اعلام خواهد شد.

نت اسکپ، اکنون زمینه را برای یک آزمایش بزرگ و جهانی از پیاده سازی مدل بازار در دنیای تجارت فراهم کرده است. اکنون خطری فرهنگ باز – متن را تهدید میکند؛ اگر نت اسکپ در کار خود موفق نشود، تفکرات دنیای باز متن چنان بی اعتبار خواهد شد که دنیای تجارت تا یک دهه بعد به آن نزدیک نخواهد شد.

از سوی دیگر، این یک فرصت استثنائی است. واکنش های اولیه از Wall Street و نقاط دیگر کاملا مثبت بوده است. فرصتی برای پیشرفت برایمان فراهم شده است. اگر نت اسکپ با این حرکت بتواند سهم قابل توجهی از بازار را بدست آورد، زمینه را برای انقلابی بزرگ در صنعت کامپیوتر فراهم کرده است.

سال بعد باید سال جالب و آموزنده ای باشد.

Netscape .¹
Eric Hahn .²