

کلیسای جامع و بازار

Eric S. Raymond

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 سر تا سر جهان پراکنده اند و بوسیله اینترنت با هم در ارتباطند، شکل یابد؟

حقیقتاً، من چنین تصویری نمیکردم. اوائل سال ۱۹۹۳، زمانی که لینوکس به دایره ذهن من راه یافت، ده سال میشد که من درگیر یونیکس و جنبش باز-متن بودم. من یکی از اولین شرکت کنندگان پروژه GNU در اواسط دهه ۱۹۸۰ 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 پایه‌ای و مهم (سیستم عامل و ابزارهای بزرگی همچون Emacs) می‌بایست با مهارتی خاص و بوسیله نوآوری منحصر بفرد یا گروهی از دانشمندان منزوی، همانند یک کلیسا ساخته شوند؛ بدون اینکه نسخه بتائی از آن را قبل از زمان ارائه آن توزیع کنند.

روش برنامه نویسی لینوس توروالدز (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 روش های گوناگون و متنوع شباهت داشت (که شایستگی آن به خوبی بوسیله آرشیو سایتهای لینوکسی، که تائید نظرات افراد مختلف در آن وجود دارد، قابل درک است)؛ که در این صورت، یک سیستم اینچنین پایدار و منسجم ظاهراً 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 پروژه‌های مختلف، سعی میکردم تا بفهمم چرا دنیای لینوکس 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 ۱۹۹۶ حس کردم که به نتایجی برای درک این مطلب رسیده‌ام. شانس و اقبال مرا به سمت راه درستی برای تست تئوری‌ام، 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 خواهیم پرداخت و در میان آن، نکات مورد نیاز برای یک پروژه باز متن موفق را بیان خواهیم کرد. من تمامی این تجربیات را در ابتدا از دنیای لینوکس یاد نگرفتم، اما در ادامه خواهیم دید که چطور دنیای لینوکس به آنها ارزش ویژه‌ای می‌دهد. اگر نظر من درست باشد، به شما کمک خواهد کرد که تا دقیقاً دریابید که چه چیز، جامعه لینوکس را به چنین منبع ارزشمندی از نرم افزارهای خوب تبدیل کرده است – و همچنین به شما کمک خواهد کرد که خلاق تر و کارا تر شوید.

نامه باید به مقصد برسد

در سال ۱۹۹۳، من در حال کار در قسمت فنی یک ISP دسترسی-آزاد (Free Access) کوچک به نام CCIL در غرب Chester در Pennsylvania بودم. (من یکی از موسسان CCIL بودم و تنها نرم افزار تابلوی اعلانات (Bulletin Board) 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 ۱۹ خط پشتیبانی می‌کند) این کار به من امکان دسترسی ۲۴ ساعته به اینترنت را از طریق خط ۵۶K 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 SLIP بین ماشین خانگی‌ام (snark.thyrusus.com) و CCIL خیلی سخت بود. سرانجام زمانی که موفق شدم، انجام تناوبی عمل telnet برای چک کردن نامه‌هایم را آزردهنده یافتم. آنچه که می‌خواستم این بود که نامه‌هایم چنان بر روی سیستم خانگی‌ام دریافت شوند که زمانی که نامه‌ها می‌رسند، من متوجه شوم و بتوانم آنها را بوسیله تمامی ابزارهای محلی‌ام مدیریت کنم.

سیستم ارسال ساده پست الکترونیکی (Simple Sendmail Forwarding) جوابگو نبود، چرا که ماشین شخصی من همواره به شبکه متصل نبود و آدرس IP ثابتی نداشت. آنچه که نیاز داشتم، برنامه‌ای بود که به کانکشن SLIP من دسترسی پیدا کرده تا نامه‌هایم را در اختیارم قرار دهد. میدانستم که چنین چیزی وجود دارد و اکثر آنها از پروتکل ساده‌ای در لایه کاربرد به نام POP 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 آدرس‌های نامه‌های دریافت شده تا پاسخ نامه‌ها به درستی داده شود.

مشکل این بود: فرض کنید فردی با نام joe از locke برای من نامه‌ای فرستاده باشد. اگر من نامه را از snark دریافت 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 آدرس‌ها تا ccil.org@ به آن ضمیمه شود، سریعاً به کاری عذاب آور تبدیل شده بود.

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

(۱) 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 هائی میکنند که نه به آن نیاز دارند و نه علاقه‌ای. اما نه در دنیای لینوکس! - توضیح خواهیم داد که چرا میانگین کیفیت نرم افزار هائی 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 بپردازم؟ خیر! من به دقت به ابزارهای POP که در اختیار داشتم نگاه کردم، و از خودم سوال کردم "کدامیک به آنچه که من می‌خواهم نزدیکتر است؟". به این دلیل که:

(۲) برنامه نویسان خوب، می‌دانند که چطور برنامه بنویسند. اما برنامه نویسان خبیره، می‌دانند که چطور برنامه‌ها 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 می کنید. 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 حقیقتاً تلاش نکرد تا سیستم عامل لینوکس را از ابتدا بنویسد. او با استفاده مجدد از کدها و ایده هایی که از MINIX، یک سیستم عامل کوچک شبیه یونیکس برای ماشین های ۳۸۶، گرفته بود کار خود را شروع کرد. عاقبت، تمام کدهای 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 می داد.

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.

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 قبلاً پیدا کرده بودم، جستجوی دوم من نه نتیجه را در بر داشت - `fetchpop`، `PopTart`، `get-mail`، `gwpop`، `pimp`، `pop-perl`، `popmail` و `upop`. در ابتدا تمرکز خود را بر روی `fetchpop` از سونگ-هانگ (Sueng-Hong Oh) 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 نویسندگان برنامه در نسخه ۱.۹ آنها را ترتیب اثر داد.

چند هفته بعد، به طور اتفاقی به برنامه `popclient` که کارل هریس (Carl Harris) آن را نوشته بود، برخورد کردم و دریافتم که اشتباهی مرتکب شدم. هر چند برنامه `fetchpop` حاوی ایده های خوبی بود (برای مثال حالت `daemon`، ولی تنها قادر به کار با POP3 بود و نسبتاً ناشیانه نیز نوشته شده بود (سونگ-هانگ فرد زیرکی بود، اما برنامه نویسی با تجربه ای نبود، این دو ویژگی، به خوبی در برنامه اش قابل مشاهده بود). کد Carl بهتر بود و کاملاً 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 می کردم؟ اگر برنامه ام را عوض می کردم، 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 APOP پشتیبانی نمی کردند، و من همچنین افکاری نه چندان جدی در ارتباط با اضافه کردن احتمالی پروتکل IMAP به برنامه نیز داشتم.

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

۳) اگر بخواهی که چیزی را کنار بگذاری، در هر صورت سرانجام این کار را میکنی!

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، احتمالاً اینقدر میدانی که چطور کارت 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 بسته‌های نرم‌افزاری popclient را برای کارل هریس در تاریخ ۲۵ ژوئن ۱۹۹۶ 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 نرم‌افزاری کوچک را برای نرم افزار نمی‌نوشتیم و به نگهداری و مدیریت کل برنامه می‌پرداختیم؛ و همان زمان بود که این ایده به ذهنم خطور کرد که از این به بعد احتمالا می‌توانم به تغییرات اساسی بپردازم.

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

(۴) 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:

(۵) 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.

اهمیت وجود تعدادی کاربر

و بدین گونه من نرم افزار popclient را به دست گرفتم و نتیجتا، 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 منبع برنامه در دسترس است، این امکان برای آنها وجود دارد که هرکدام 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 هائی را برای حل این مشکل پیشنهاد دهند و به بهبود برنامه نویسی خیلی سریعتر از حالتی که شما تنهائی به این کار می‌پرداختید، کمک کنند.

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

قدرت کارائی این نکته، بسادگی مورد کم توجهی قرار می‌گیرد. در حقیقت، تقریبا 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 لینوکسی بوده است. زمانی که این عقیده را در حضور او مطرح کردم، او خندید و سریعا آنچه‌ای که همیشه می‌گفت را تکرار کرد: "من اصولا آدم تنبلی هستم که می‌خواهد از کارهائی که حقیقتا دیگران انجام می‌دهند، استفاده‌ای ببرم". تنبل همچون روباه، یا همانطور که رابرت هینلین گاهی می‌گفت، آنقدر تنبل که ممکن است ببازی.

با نگاهی به گذشته، یک نمونه برای روش‌ها و موفقیت لینوکس را می‌توان در توسعه کتابخانه GNU Emacs Lisp و آرشیو کدهای Lisp مشاهده کرد. در قیاس با روش ساخت کلیسائی هسته Emacs C و بسیاری از ابزارهای دیگر بنیاد نرم‌افزار آزاد، تکامل کدهای Lisp بسیار روانتر و کاربر محورانه تر بود. ایده‌ها و پیش‌الگوها اغلب سه یا چهار بار قبل از رسیدن به حالت پایدار نهائی بازنویسی می‌شد و همکاری‌های دورادوری که بوسیله اینترنت انجام می‌گرفت، معمول بود.

در واقع، تنها هک موفقیت آمیز من قبل از fetchmail، احتمالاً VC Mode Emacs بود؛ یک همکاری لینوکس مانند که بوسیله ارسال email با سه نفر دیگر انجام گرفت. تنها یکی از آنها (ریچارد استالمن، پدید آورنده Emacs و موسس بنیاد نرم‌افزار آزاد یا FSF) را تا به امروز ملاقات کرده‌ام. آن برنامه مقدمه‌ای برای SCCS، RCS 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؛ چرا که، 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 پیش‌بینی نشده خط مشی FSF در تلاش برای قانونی همراه کردن کد برنامه با GPL این است که بدین روش استفاده از مدل بازار برای FSF مشکل‌تر می‌شود؛ چرا که باید مجوزهای کپی‌رایت را برای هر همکاری در برنامه که بیش از بیست خط صورت گرفته، اجرا کنند تا برنامه تحت لیسانس GPL از رویارویی با قوانین کپی‌رایت مصون بماند. کسانی که از قانون کپی‌رایت تحت لیسانس‌های 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.

زود منتشر کن، مرتب منتشر کن

زود منتشر کردن برنامه و انتشار متناوب نسخه‌های 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 پروژه‌های بزرگ است؛ تنها به این دلیل که ورژن‌های ابتدائی، نسخه‌های پر اشکالی هستند و شما نمی‌خواهید که صبر کاربرانان به پایان برسد.

این طرز فکر، افراد را به سبک کلیسائی برنامه نویسی هدایت خواهد کرد. اگر هدف اصلی این باشد که کاربران کمترین خطای ممکن را مشاهده کنند، آنوقت چرا شما تنها نسخه‌های برنامه‌تان را هر شش ماه یک بار (یا حتی کمتر) انتشار می‌دهید و شدیداً برای رفع خطاها در زمان بین انتشار نسخه‌های برنامه‌تان کار می‌کنید؟ هسته Emacs C بدین روش ساخته شد. کتابخانه Lisp، عملاً اینطور نبود – به این خاطر که آرشیوهای فعالی از Lisp در خارج از کنترل FSF وجود داشت، که شما می‌توانستید نسخه‌های 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 آرشیوهای بزرگ امروز لینوکسی پیشی گرفته بود. اما تنها تعداد کمی از ما واقعا درباره آنچه که انجام می‌دهیم، درست اندیشیده‌ایم؛ و یا درباره ماهیت واقعی چنان آرشیوی با در نظر گرفتن مشکلات مدل برنامه نویسی کلیسائی FSF، تعمق کرده‌ایم. حدوداً سال ۱۹۹۲ 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، همانطور که لینوکس به طور گسترده ظاهر شد، به خوبی مشخص بود که چیزی متفاوت و البته بهتر در حال اتفاق افتادن است. سیاست توسعه نرم‌افزار باز لینوکس کاملاً با ساختار کلیسائی در تضاد بود. آرشیوهای sunsite و tsx-11 شروع به رشد کردند، توزیع‌های زیادی بوجود آمدند؛ و همه اینها بوسیله انتشارهای متناوب و غیر معمول هسته سیستمی، بوجود آمد.

لینوکس با کاربرانش تا بیشترین حد ممکن، همانند همکاری‌هایش برخورد می‌کرد:

برنامه‌ات 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 ۱۹۹۱)، انتشار هسته‌ای جدید بیش از یک بار در هر روز، برای او عجیب نبود. چرا که 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 گوسلینگ (از NEWS و Java 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 ویژگی‌ها 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 بخش‌هایی 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 هرکها/کاربران خود را همواره ترغیب و از آنها قدردانی می‌کرد ترغیب بوسیله نمایش دورنمایی از کاری که خودشان در آن سهیمند و احساس رضامیت‌مندی درونی حاصل، و قدردانی با نمایاندن (حتی روزانه) پیشرفتی که در کارشان حاصل می‌شود.

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

با داشتن همکاران برنامه نویس و تست کننده‌های 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" نامیده‌ام.

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 اینجا است که هر دوی آنها سریعاً اتفاق می‌افتد.

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، مودری و دردسر ساز هستند. ماهها بررسی تعداد کمی از افراد که وقت خود را به این کار اختصاص داده‌اند، لازم است تا اطمینان حاصل کنید که تمامی خطاها را برطرف کرده اید. نتیجه این روش، وقفه های زمانی طولانی بین انتشار نسخه‌های مختلف نرم افزار، و یاسی اجتناب ناپذیر در حالتیکه که نسخه‌های دیر منتشر شده برنامه شما کارائی مورد نظر را ندارند، خواهد بود.

از طرف دیگر، در دیدگاه بازار، شما فرض می‌کنید که اشکالات برنامه ذاتاً اتفاقاتی سطحی هستند و یا حداقل، هنگامی که در معرض هزاران برنامه نویس مشتاق که منتظر دریافت نسخه‌ای جدید از برنامه هستند، قرار می‌گیرند، سطحی و ضعیف خواهند شد. بنابراین، شما برنامه خود را به تناوب منتشر می‌کنید تا برنامه‌تان بیشتر تصحیح شود و یکی از نتایج جانبی سودمند آن این است که اگر سهواً 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 هماهنگیها و اشکالات اساسی کشف نشده، با شکست مواجه میشد. از سوی دیگر، اگر این قانون دست باشد، تنها کفایت که به شرح دلایل کم خطا بودن لینوکس بپردازیم.

البته شاید دلیلی برای شگفتی هم وجود نداشته باشد، چراکه جامعه شناسان سالها قبل کشف کرده‌اند که میانگین نظر گروهی از افراد متخصص (و یا نادان)، قابل اعتمادتر از تک تک آنهاست. آنها چنین اصلی را "اثر دلفی" نامگذاری کرده‌اند. پیداست که آنچه که لینوس نشان داده است اینست که این مطلب حتی در عیب یابی نرم‌افزاری 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 داتکی 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.

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

بروکس، حتی چنین عقیده تخمینی را با توجه به نظر جف داشت: "هزینه کلی نگهداری برنامه‌ای که به طور گسترده مورد استفاده قرار می‌گیرد، حدود چهل درصد و یا بیشتر از هزینه ایجاد آن است. جالب اینجاست که این هزینه شدیداً تحت تاثیر تعداد کاربران آن نرم افزار قرار دارد. کاربران بیشتر، اشکالات بیشتری را نیز پیدا می‌کنند." (چیزی که همواره بر آن تاکید دارم.)

کاربران بیشتر، اشکالات بیشتری را نیز پیدا می‌کنند، چرا که افزایش یافتن تعداد کاربران، به افزایش روش‌های 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 شناسائی 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 کاربها نیز خواهد انجامید.

بنابراین، افزودن تعداد تست کننده‌های حالت بتای نرم افزار، به کاهش پیچیدگی مشکلات موجود از دیدگاه برنامه نویس 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 ویژگی‌ها 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.

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 ساده‌تر و کوچکتر بود).

اما اولین کاری که انجام دادم، سازماندهی دوباره و ساده سازی نرم افزار popclient بود. پیاده سازی کارل هریس (Carl Harris) خیلی دقیق بود، اما نوعی پیچیدگی اضافی در قیاس با خیلی از برنامه نویسان C در برنامه‌اش به چشم می‌خورد. او با کد برنامه به عنوان بخش اصلی و ساختمان داده‌ها 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 در کنار بهبود طرح برنامه و ساختار ساختمان داده آن داشتم. در واقع، می‌خواستم برنامه را کاملاً درک کنم. زیاد جالب نیست که مسئول درست کردن برنامه‌ای باشی که آن را درک نکرده‌ای!

حدوداً در ماه اول، به پیروی از ساختار پایه‌ای 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 پروتکل‌ها در یک دراپور عمومی و سه جدول متد (برای POP2، POP3 و IMAP) انجام دادم. این تغییر و تغییرات قبلی نمایانگر یک اصل کلی هستند که خوب است برنامه نویسان آن را به خاطر بسپارند، به خصوص در زبان‌های برنامه نویسی از قبیل C که به طور پیش فرض با انواع پویا کار نمی‌کنند:

ساختمان داده‌ی 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، برنامه‌ات را به من نشان بده ولی ساختمان داده برنامه‌ات را پنهان کن. اما اگر ساختمان داده برنامه‌ات را به من نشان دهی، اغلب دیگر نیازی به دیدن کد برنامه‌ات ندارم؛ همه چیز واضح و مشخص است."

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

در این زمان (اوائل سپتامبر سال ۱۹۹۶، حدوداً شش 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 نامه‌های دریافت شده به پورت SMTP شد. درباره اش خواهم گفت؛ اما قبل از آن: قبلاً گفتیم که تصمیم داشتیم که این پروژه را برای تست تئوری‌ام درباره آنچه که لینوس توروالدز انجام داده بود، به کار بگیریم. ممکن است بپرسید چطور این کار را کردم؟ به روش‌های زیر:

(۱) برنامه‌ام 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)

(۲) هر کس را که با من به نحوی درباره fetchmail در تماس بود را به لیست بتای خودم اضافه می‌کردم.

هر زمانی که نسخه جدیدی را انتشار می‌دادم، آگهی‌های زیادی را به لیست بتای خودم ارسال می‌کردم تا آنها را به مشارکت در پروژه تشویق کنم.

(۳) به تست کننده‌های نسخه بتای نرم افزارم گوش می‌کردم، نظر آنها را درباره تصمیماتم جویا می‌شدم و هر زمان که آنها بسته‌های نرم افزاری و یا نظراتشان را برایم ارسال می‌کردند، از آنها تشکر می‌کردم.

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

اگر شما با تست کننده‌های نسخه بتای برنامه‌تان، چنان که آنها با ارزشترین منبع شما هستند، برخورد کنید، آنها به عنوان با ارزشترین منبع شما به شما پاسخ خواهند داد.

یک نکته ارزشمند از موفقیت fetchmail، تعداد اعضای لیست نسخه بتای پروژه، یعنی دوستان fetchmail-friends است. در زمان نوشتن این مقاله، لیست مذکور ۲۴۹ 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 ۱۹۹۷، 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.

نرم افزار popclient به fetchmail تبدیل میشود

تحول اصلی در پروژه زمانی صورت گرفت که هری هاچیسر (Harry Hochheiser) 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 هایی که برای کپی برداری نامه دریافت شده به یک فایل در صندوق پستی و یا یک دستگاه خروجی استاندارد تعبیه شده بودند، خاطریم را می آزرند، اما دلایل 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 الکترونیکی (MTA) و هم یک عامل تحویل موضعی (MDA) باشد. با قابلیت ارسال SMTP، می توانست از حوزه کاری MDA خارج شده و یک MTA محض باشد که به سادگی sendmail، نامه را به برنامه های دیگر برای تحویل موضعی تحویل می دهد.

چرا تمامی پیچیدگی های پیکربندی یک MDA یا عملیات قفل و الحاق (Lock and Append) بر روی یک صندوق پستی در هم آمیخته شوند، در حالیکه پورت ۲۵ تقریباً در تمامی پلتفرمهایی که از TCP/IP پشتیبانی می کنند، 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 در واقع همان چیزی است که ما می خواهیم.

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 می دادم این بود که مفاهیم را درک کنم.

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 خلقش 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 ۱۹۹۷ ارائه دادم، لری وال در ردیف اول نشست بود. وقتی که به آخرین خط پاراگراف فوق رسیدم، او همچون مبلغ های مذهبی فریاد زد: "بگو، بگو براد!". تمامی حضار خندیدند، چرا که می دانستند 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 زندگی بی ارزش بوده، یک نگاهی به آنها می‌اندازم

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

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، برنامه‌ام تمامی انواع تحویل محلی 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؛ (۱) قابلیت ارسال **SMTP** را در قالب یک درایور عمومی بگنجانم؛ (۲) آن را حالت پیشفرض قرار دهم؛ و (۳) 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 ۳ به تردید می‌افتادم، و از این می‌ترسیدم که کاربران قدیمی **popclient** که به مکانیزم های تحویل دیگر عادت کرده‌اند، از این کار ناراحت شوند. در 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 بهبود 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 دست دادن نامه ها نیز بود. اگر شما فایلی را به عنوان محل تحویل نامه هایتان در نظر می‌گرفتید 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 **SMTP** شما تا زمانی که نامه شما تحویل داده نشود یا حداقل برای تحویل در زمانی دیگر اسپول (**Spool**) نشود، پیغام **OK** را بر نمی‌گرداند.

کارائی 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 کاربر را برای ایجاد امکان مدیریت بعضی از حالات نادری که در ارتباط با **SLIP** پویا بوجود می‌آمد، مجدداً به برنامه اضافه کنم. اما راه بسیار ساده‌تری را برای انجام آن پیدا کردم.

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

رسیدن به حد کمال (در طراحی) در حالتیکه که چیز دیگری برای اضافه کردن وجود ندارد، بدست نمی‌آید، 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 می شود، 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 می خورد؛ هر دو به نوعی MTA بودند، اما برنامه popclient جدید متفاوت عمل می کرد. در نهایت و پس از دو ماه، نام برنامه را به fetchmail تغییر دادم.

برنامه fetchmail رشد می کند

حال، من طرح برنامه ای ابداعی و منظم را در اختیار داشتم و به خوبی می دانستم که در کارم موفق بوده ام چرا که هر روز از آن استفاده می کردم، و لیست بتائی که صحت این ادعا را تأیید می کرد. بتدریج حس می کردم که ابتکارهای کوچکی که ممکن است برای افراد کمی سودمند باشد، دیگر مرا ارضاء نمی کند. من در نوشتن برنامه های دست داشتم که برای هر هکری که از یک نسخه یونیکس و یک کانکشن میل SLIP/PPP استفاده می کند، 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.

حدس می زنم که حتی نتوانید چنین نتیجه ای را متصور شوید. تنها زمانی می توانید، 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 ذهنستان حاصل می شود - 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 تنبام ایده اصلی ساخت یک سیستم عامل کوچک محلی شبیه یونیکس را برای ماشین های ۳۸۶ 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 ایده های 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 می کنند، 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 می کند! و همه اینها به این معنی بود که من می توانستم سطح استانداردهایم را حتی فراتر از این نیز ببرم. برای آنکه fetchmail را به آن خوبی که در خورش بود تبدیل کنم، می بایست 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 می کردم. و البته همه این کارها را به گونه ای انجام دهم که برنامه همچنان ساده اما قدرتمند باقی بماند.

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

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 توده‌ای 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:

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 فهرست‌های پستی با نگهداشتن لیست، و نیز امکان بسط نامهای مستعار، 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 فایل‌های نام های مستعار ISP داشته باشد.

تغییر مهم دیگری که تست کننده‌های بتای نرم‌افزار خواستار آن شده بودند، پیشیبانی از عملیات ۸-بیتی MIME بود. انجام این کار بسیار ساده بود، چرا که حواسم به کدهای ۸-بیتی بود؛ نه به خاطر این که من احتمال درخواست این ویژگی را از قبل پیش بینی می‌کردم، بلکه بدلیل پیروی از اصلی دیگر:

وقتی که نرم افزار دروازه ای، 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 پشتیبانی 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 گزینه‌ای برای محدود ساختن تعداد پیغام‌هایی که در هر جلسه بازیابی میشود، شدند (که در این صورت، آنها می‌توانستند هزینه‌های گران شبکه‌های تلفنشان را کنترل کنند). من مدت زمان زیادی بر این موضوع پافشاری 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.

درس‌های دیگری از پروژه Fetchmail

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

در سینتکس فایل های rc بعضا کلمات کلیدی بی‌ارزش یافت میشود، که تماما توسط تجزیه‌گر برنامه نادیده گرفته میشود. سینتکس انگلیسی-مانند بسیار خواناتر از کلمات کلیدی مختصر و مرسوم است که عموما استفاده میشود.

این نکته زمانی به ذهنم خطور کرد که دریافتم که چقدر تعریف فایل های rc، میتواند 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 واژه‌های اصلی برنامه به انگلیسی محاوره ای نزدیکتر باشد، کاربرد برنامه نیز ساده‌تر خواهد شد. هم‌اکنون، هر چند من از حامیان اصلی مکتب " از آن زبان بساز " در طراحی نرم‌افزار هستم که نمونه‌های آن را میتوان در Emacs و HTML و خیلی از موتورهای دیتابیس یافت، معمولا علاقه زیادی به استفاده از سینتکس انگلیسی-وار ندارم.

برنامه‌نویسان قدیمی به استفاده از واژه‌هایی کاملا دقیق، مختصر و بدون هیچ گونه زیاده گوئی گرایش دارند. این سبک، میراثی فرهنگی از زمانی است که منابع کامپیوتری گران بوند و لذا مراحل عمل تجربه میبایست تا حد امکان ارزان و ساده میبود. در این صورت طبیعی بود که زبان انگلیسی با افزودگی در حدود ۵۰٪، یک مدل نامناسب بوده باشد.

این موضوع، دلیل من برای اجتناب عادت گونه‌ام از واژه‌های محاوره‌ای انگلیسی نبود؛ به آن اشاره کردم تا اشتباه بودنش را گوشزد کنم. با هسته و سیکل هائی که ارزان تمام شوند، ایجاز حاصل به خودی خود یک فرجام نیست. این روزها مهمتر این است که زبان برنامه‌نویسی برای انسان راحت‌تر باشد تا اینکه برای ماشین ارزانتر تمام شود.

هرچند، دلایل بسیاری برای محتاط بودن وجود دارد. یکی، هزینه پیچیدگی در مرحله تجزیه است - شما قطعاً نمیخواهید به جایی برسید که مجموعه کاملی از اشکالات و عوامل مختلف پریشانی کاربر، برنامه‌تان را فرا بگیرد. دیگری این است که تلاش برای تبدیل سینتکس زبان به حالتی محاوره‌ای و انگلیسی مانند، اغلب نیازمند این است که انگلیسی مورد 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 گیج کننده خواهد بود (این مطلب را در بسیاری از به اصطلاح " نسل چهارمی "ها و زبان‌های پرس و جوی دیتابیس‌های تجاری 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 مجموعه‌ای کلمات روزمره انگلیسی و زبان مدیریتی واقعی نرم افزار، داشت. فکر میکنم درس بزرگی در این نکته نهفته است:

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

نکته دیگر تامین امنیت واهی است! بعضی از کاربران fetchmail از من میخواستند تا نرم‌افزار را به گونه‌ای تغییر دهم تا پسوردها را به شیوه رمزگذاری 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، توانائی اجرای برنامه شما را نیز همانند شما خواهد داشت - و اگر به دنبال پسورد شما باشند، با استفاده از رمزگشاهای 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:

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

پیشنیازهای لازم برای سبک بازار

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

کاملاً مشخص است که کسی نمیتواند در سبک بازار، برنامه‌نویسی را از ابتدا شروع کند. هر فرد، قادر به تست برنامه، اشکال‌زدایی و بهبود نرم‌افزار 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 برنامه‌نویسانِ پاگرفته شما نیازمند پروژه‌ای 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، فولی است که دیگران آن را باور داشته باشند. ممکن است که برنامه‌تان چنان که باید درست کار نکند؛ ممکن است منطقتان خام، کدتان پراشکال، برنامه‌تان ناقص، و مستند سازیتان ضعیف باشد. چیزی که نباید در آن کوتاهی کنید متقاعد کردن درونیِ همکاران برنامه‌نویستان است که میتواند برنامه‌تان را در آینده‌ای نزدیک به برنامه‌ای کارا و خوب تبدیل کند.

لینوکس و **fetchmail**، هر دو با طرح‌های پایه‌ای قدرتمند و جذاب، با به جامعه گذاشتند. افراد بسیاری که پس از ارائه این مقاله به مدل بازار می‌اندیشند، به این نکته مهم پی برده‌اند؛ سپس از کل مطالب به این نتیجه میرسند که سطح بالای بینش طراحی نرم‌افزار و هوشمندی در سرپرستی پروژه، از هم تفکیک‌ناپذیرند.

اما لینوس، طرح برنامه خود را از یونیکس گرفت. من طرح خودم را از نسخه‌های 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/همه‌انگ کننده یک پروژه در سبک بازار واقعاً نیازمند داشتن استعداد استثنائی طراحی نرم‌افزار است، یا اینکه او میتواند از قدرت استعداد طراحی دیگران بهره‌بردار؟

فکر میکنم، زیاد مهم نیست که همه‌انگ کننده توانائی خلق طرح‌هایی با نبوغ استثنائی را داشته باشد، بلکه آنچه که مهم است این است که همه‌انگ کننده پروژه قادر باشد تا ایده‌های خوب را از دیگر ایده‌ها تشخیص بدهد.

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 فوق‌العاده‌ای را از خود برای تشخیص طرح‌های خوب و یکپارچه‌کردنشان با هسته لینوکس نشان داد. و من نیز پیش از این، به شرح چگونگی شکل‌گیری 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 پروژه‌های تحت بازار راحت جلوه داده‌ام، 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 برنامه‌نویسی و یا عیب‌یابی) مطمئناً قویترین مهارت من است.

اما مشکل زیرک و مبتکر بودن در طراحی نرم‌افزار 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. پروژه‌هایی را بیاد می‌آورم که در آنها به مشکل برخوردیم، چرا که مرتکب این اشتباه شدم؛ اما حواسم بود که این اشتباه را در **fetchmail** مرتکب نشوم.

بنابراین، به این باور رسیدم که پروژه **fetchmail** تاحدی موفقیت‌آمیز بود، چرا که زیرکی‌ام را کنترل کردم؛ این استدلال (حداقل) در تضاد با ابتکار در طراحی، برای پروژه‌هایی که سبک بازار اجرا میشوند، لازم و ضروری است. و لینوکس را در نظر بگیرید. فرض کنید لینوس توروالدز سعی میکرد نوآوری‌های بنیادین در طرح سیستم‌عامل را در طی توسعه پروژه‌اش 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 پایه‌ای مشخصی از توانائی طراحی و برنامه‌نویسی ضروری است، اما من معتقدم تقریباً هر کسی که بطور جدی درباره اجرای کاری در سبک بازار می‌اندیشد، از آن سطح مینیمم گفته شده بالاتر است. بازار داخلی جامعه باز متن، فشار زیرکانه‌ای را بر روی افراد اعمال میکند تا به دنبال کارهایی در برنامه‌نویسی 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 نرم‌افزاری 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 مهارت‌های ارتباطی خوب در برخورد با افراد، برخوردار باشد.

این مسئله بدیهی است. برای ساخت یک انجمن توسعه نرم‌افزار، 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 برون‌گرای فعال هستم که از کار کردن با دیگران لذت میبرم و مهارت زیادی در طنز دارم. برای اینکه مدل بازار با برجا بماند، اندک توانائی شما در مجذوب ساختن افراد میتواند بسیار موثر واقع شود.

زمینه اجتماعی نرم‌افزار بازمتن

حقیقت دارد که: بهترین هک‌ها از راه حل‌های شخصی برای مشکلات هر روزه مبتکران شروع میشود؛ و سپس به طور گسترده مورد استفاده قرار میگیرد، چرا که مشکل فوق، میتواند مشکل گروه گسترده‌ای از کاربران باشد. این ما را به دلیل قانون ۱ باز میگرداند، که اگر بخواهیم آن را به شیوه‌ای بهتر بیان کنیم، میتوانیم بگوئیم:

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

برای من، کارل هریس و ورژن‌های اولیه **popclient** جرقه‌های اولیه شروع بود، که نتیجه اش برنامه **fetchmail** شد. اما درک این نکته نیازمند گذشت زمان نسبتاً زیادی است. نکته جالب توجه، این است که بجا بو. نکته جادان انتخاب سیر حرکت لینوکس و **fetchmail** برای بررسی، ما را به مرحله بعدی هدایت میکند - سیر تکاملی نرم‌افزار در بین انجمن‌های زیاد و فعالی از کاربران و برنامه نویسان.

فرد بروکس در کتابش، نشان داد که زمان برنامه‌نویس قابل تعویض نیست، و اضافه کردن برنامه نویسی جهت تسریع در پروژه‌ای 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 برنامه‌نویسان آن، هنگامی که نمودار پیشرفت پروژه خطی است، نسبت مستقیم دارد. این ادعا "قانون بروکس" نام گرفت و به عنوان قانونی بدیهی پذیرفته شد. اما اگر قانون بروکس همه ماجرا بود، در این صورت شکل‌گیری لینوکس غیر ممکن میشد.

تنها چند سال بعد، جرال وینبرگ در کتاب روانشناسی برنامه‌نویسی کامپیوتر، بیان کرد که قانون بروکس باید به طور اساسی تصحیح شود. در بحث "برنامه‌نویسی با دیگران (egoless programming)"، وینبرگ نشان داد که در حالاتی که برنامه‌نویسان 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 نمونه‌اش را نمیتوان در هیچ مدل دیگری یافت.

میتوان گفت که واژه‌های انتخابی وینبرگ، تحلیلش را از پذیرشی که شایسته و سزاوار آن بود، بر حذر داشت - مثلاً بکار بردن واژه "egoless" برای هکرهای اینترنتی، برای برخی خنده‌دار بود. اما من فکر میکنم که استدلال او، امروزه بیش از زمان دیگری به واقعیت نزدیک است.

تاریخچه یونیکس باید قاعدتاً 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 عمدی از روش لینوس، شرح دادم). یعنی، زمانی که برنامه‌نویسی 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. برنامه‌نویسی که تنها از قدرت فکر خود در یک پروژه بسته استفاده میکند، در قیاس با برنامه‌نویسی که به خوبی میداند که چطور زمینه یک پروژه باز و تکاملی را که عیب یابی و پیشرفت آن به کمک صداها برنامه نویسی 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 محدودیت‌های مجوزهای گوناگون، اسرار تجاری، و سودهای 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" وار وینبرگ به فعالیت میپرداختند، و یک برنامه نویس به راحتی میتوانست نظر بسیاری از افراد حرفه‌ای و برنامه نویسان دیگر را جلب کند. آزمایشگاه Bell، MIT AI، و UC Berkeley، به مهد نوآوری‌های افسانه‌ای و بزرگ تبدیل شده بودند.

لینوکس اولین پروژه‌ای 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 ۱۹۹۳-۱۹۹۴ همزمان با گسترش صنعت سرویس‌دهندگان اینترنت و انفجار تقاضا برای اینترنت سپری کرد. لینوکس 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 نتیجه‌ای که امروز شاهدش هستیم، در بر نخواهد داشت. وینبرگ از کتاب زندگینامه پایتر الکسیویچ کروپتکین، آنارشویست قرن نوزدهم روسیه، با عنوان "خاطرات یک انقلابی"، مطالبی را در تائید این موضوع نقل قول کرده است:

"پس از گذراندن دوران کودکی در خانواده‌ای فقیر، مثل هر مرد جوان دیگری در آن دوران، وارد زندگی واقعی شدم؛ با باور لزوم حرف شنوی، فرمانپذیری، 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 رژه‌های 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 پروژه‌های همانند لینوکس به چه چیزی نیاز دارد - و "قانون حرف شنوی" را به هیچ وجه نمیتوان در میان داوطلبانی که در بهشت آنارشویست‌ها!! که ما آن را اینترنت مینامیم، بسر میبرند، بکار برد. برای اینکه خوب به کار و رقابت بپردازید، هکرها می‌خواهند پروژه‌های گروهی را رهبری کنند، باید چگونگی جذب افراد و تزریق انرژی و شوق به کار را در انجمن‌های 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 لینوکس استفاده کنند.

پیشتر، از "اثر دلفی" به عنوان تفسیری قابل قبول برای قانون لینوکس، یاد کردم. اما قیاس‌های قدرتمند دیگری نسبت به سیستم‌های تطبیقی در زیست‌شناسی 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 شده‌ای خواهد انجامید که بسیار کارا تر از هر پروژه متمرکزی است. اینجاست که باید به دنبال نتایج "اصل توافق" بگردیم.

منظور از تلاش برای "عملکرد بر اساس منفعت" در هکرها لینوکسی، تنها به معنی کلاسیک اقتصادی آن نیست، بلکه به معنی حس معنوی حاصل از رضایتمندی درونی آنها و شهرتی است که در میان هکرها دیگر بدست می‌آورند. (ممکن است این انگیزه، نوعدوستانه عنوان شود، اما نمیتوان انکار کرد که نوع دوستی نیز به خودی خود، نوعی از رضایتمندی درونی است که به فرد نوعدوست دست میدهد). فرهنگ‌های اختیارگرا که از این روش استفاده میکنند، کم نیستند. یکی از آنها که من مدت زیادی است در آن مشارکت دارم، گروه هواداران داستان‌های علمی-تخیلی است،

برخلاف آنچه که هکرها آنرا صریحا به " egoboo " افزایش شهرت کسی در میان دیگر هواداران (میشناسد، محرک اصلی فعالیت های داوطلبانه آن است.

لینوس، با درست قرار دادن خویش به عنوان دروازه بان پروژه ای که توسعه اش اکثرا 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 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 اصولا 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، کاملا توسط (تنها برای اینکه مثالی ذکر کنم) تنوع گیج کننده، کیفیت و عمق مستندات لینوکس مردود است. اگر برنامه نویسان از مستندسازی متنفرند، پس چگونه است که هکرهای 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 مستندسازی 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 هکرهای 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 بکشاند. بنابراین، با توجه به قانون بروکس ، من میگویم:

اگر هماهنگ کننده پروژه، از رسانه های همچون اینترنت استفاده کند، و بداند که چگونه بدون استفاده از جبر به رهبری بپردازد، مطمئنا 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 افزارهای 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.

احتمالا، 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.

پس گفتار : نتاسکیپ به بازار میپیوندد

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

در ۲۲ ژانویه سال ۱۹۹۸، حدوداً هفت ماه پس از اولین انتشار این مقاله، شرکت نتاسکیپ، اعلام کرد که قصد دارد کد منبع برنامه netscape communicator را انتشار دهد. من از این ماجرا تا روز انتشار آن خبری نداشتم.

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

در هفته‌های بعد، من مرتباً به دعوت نتاسکیپ به silicon valley دعوت میشدم تا در کنفرانس‌های استراتژیک با برخی از مدیران بلند پایه و مهندسان فنی آنها شرکت کنم. ما به تدوین استراتژی انتشار کد منبع نتاسکیپ و مجوز مربوطه اش پرداختیم و در مورد طرح‌های بلند مدت و اثرات آن در جامعه باز-متن نیز گفتگو کردیم. الان که من در حال نوشتن این پس‌گفتار هستم، کمی زود است که بخواهم به شرح آن بپردازم، اما جزئیات آن به زودی اعلام خواهد شد.

از سوی دیگر، این یک فرصت استثنائی است. واکنش‌های اولیه از Wall Street و نقاط دیگر کاملاً مثبت بوده است. فرصتی برای پیشرفت برایمان فراهم شده است. اگر نتاسکیپ 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.

منبع: <http://www.catb.org/~esr/writings/cathedral-bazaar>

منبع ترجمه: مجله الکترونیکی IranTUX <http://www.iranlux.org> مترجم: نیما ابوطالبی nima717a@yahoo.com