**When Is a Rose Not a Rose?**

**장미는 언제 장미가 아닌가?**

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

리누스의 행동을 연구하고 왜 그것이 성공했는지에 대한 이론을 세우면서 필자는 (훨씬 덜 복잡하고 야심적인) 새 프로젝트에서 이 이론을 시험하기 위한 의식적인 결정을 내렸다.

But the first thing I did was reorganize and simplify popclient a lot.

그러나 내가 한 첫 번째 일은 popclient를 재구성하고 단순화하는 것 이었습니다.

Carl Harris's implementation was very sound, but exhibited a kind of unnecessary complexity common to many C programmers.

Carl Harris의 구현은 매우 건전하지만 많은 C 프로그래머에게 흔히 나타나는 불필요한 복잡성을 보인다.

He treated the code as central and the data structures as support for the code.

그는 코드를 중심으로 취급하고 자료 구조를 코드 지원으로 취급했습니다.

As a result, the code was beautiful but the data structure design ad-hoc and rather ugly (at least by the high standards of this veteran LISP hacker).

결과적으로 코드는 아름답지만 자료 구조 디자인은 임시적이고 오히려 보기 좋지 않았다 (적어도 이 베테랑 LISP 해커의 높은 기준으로는).

I had another purpose for rewriting besides improving the code and the data structure design, however.

그러나 코드 및 자료 구조 디자인을 개선하는 것 외에도 재 작성을 위한 또 다른 목적이 있었습니다.

That was to evolve it into something I understood completely.

그것은 popclient를 완전히 이해 한 무언가로 진화 시키는 것이었습니다.

It's no fun to be responsible for fixing bugs in a program you don't understand.

당신이 이해하지 못하는 프로그램에서 버그를 고치는 것은 책임이 없습니다.

For the first month or so, then, I was simply following out the implications of Carl's basic design.

처음 몇 달 동안 Carl의 기본 디자인의 의미를 간단히 따랐습니다.

The first serious change I made was to add IMAP support.

내가 처음 만든 심각한 변화는 IMAP 지원을 추가하는 것이 었습니다.

I did this by reorganizing the protocol machines into a generic driver and three method tables (for POP2, POP3, and IMAP).

필자는 프로토콜 컴퓨터를 일반 드라이버와 세 가지 방법 표 (POP2, POP3 및 IMAP)로 재구성하여이 작업을 수행했습니다.

This and the previous changes illustrate a general principle that's good for programmers to keep in mind, especially in languages like C that don't naturally do dynamic typing:

이 변경과 이전 변경은 프로그래머가 특히 동적으로 타이핑하지 않는 C와 같은 언어를 염두에 두는 데 좋은 일반 원칙을 보여줍니다.

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

9. 똑똑한 데이터 구조와 멍청한 코드가 다른 방법보다 훨씬 효과적입니다.

Brooks, Chapter 9: ``Show me your flowchart and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowchart; it'll be obvious.'' Allowing for thirty years of terminological/cultural shift, it's the same point.

Brooks, Chapter 9 :``당신의 순서도를 보여주고 당신의 식탁을 감추고, 나는 계속 신비화 될 것입니다. 나에게 테이블을 보여 주면, 나는 보통 당신의 순서도가 필요 없다. 그것은 명백 할 것입니다. "30 년 동안의 용어 학적 / 문화적 변화를 허용하면서, 그것은 같은 시점입니다.

At this point (early September 1996, about six weeks from zero) I started thinking that a name change might be in order—after all, it wasn't just a POP client any more.

이 시점에서 (1996 년 9 월 초, 0에서 약 6 주) 나는 이름 변경이 필요하다고 생각하기 시작했습니다. 결국 POP 클라이언트가 아닙니다.

But I hesitated, because there was as yet nothing genuinely new in the design.

그러나 디자인에 진실로 새로운 것은 없었기 때문에 나는 주저했다.

My version of popclient had yet to develop an identity of its own.

내 버전의 popclient는 아직 자신의 정체성을 개발하지 못했습니다.

That changed, radically, when popclient learned how to forward fetched mail to the SMTP port.

popclient는 가져온 메일을 SMTP 포트로 전달하는 방법을 배웠을 때 근본적으로 바뀌 었습니다.

I'll get to that in a moment.

나는 그 순간에 곧 갈 것이다.

But first: I said earlier that I'd decided to use this project to test my theory about what Linus Torvalds had done right.

하지만 먼저 : 나는 Linus Torvalds가했던 일에 대한 나의 이론을 테스트하기 위해이 프로젝트를 사용하기로 결정했다.

How (you may well ask) did I do that? In these ways:

어떻게 했습니까 (물을지도 모른다)? 다음과 같은 방법으로 :

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

• 나는 일찍 그리고 자주 풀려났다. (격렬한 발달 기간 동안에는 하루에 한 번씩, 매 10 일마다 거의 거의 빈번히 발병하지 않았다.)

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

• 나는 베타 목록에 fetchmail에 관해 나에게 연락 한 모든 사람을 추가하여 성장시켰다.

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

• 발표 할 때마다 베타 목록에 수다스러운 발표를 보내 사람들의 참여를 독려했습니다.

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

• 그리고 베타 테스터의 말을 듣고 설계 결정에 대해 폴링하고 패치 및 피드백을 보낼 때마다 쓰다듬습니다.

The payoff from these simple measures was immediate.

이 간단한 조치의 결과는 즉각적이었습니다.

From the beginning of the project, I got bug reports of a quality most developers would kill for, often with good fixes attached.

프로젝트가 시작될 때부터 대부분의 개발자가 죽일만한 품질의 버그 보고서를 받았으며 종종 좋은 수정 사항이 첨부되었습니다.

I got thoughtful criticism, I got fan mail, I got intelligent feature suggestions. Which leads to:

나는 사려 깊은 비판을 받았고 팬 메일을 받았고 지능형 기능 제안을 받았습니다. 어떤 결과가 나옵니까?

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

10. 베타 테스터가 가장 귀중한 리소스 인 것처럼 베타 테스터를 대하면 가장 귀중한 리소스가 될 것입니다.

One interesting measure of fetchmail's success is the sheer size of the project beta list, fetchmail-friends.

fetchmail의 성공에 대한 한 가지 재미있는 척도는 프로젝트 베타 목록 인 fetchmail-friends의 규모입니다.

At the time of latest revision of this paper (November 2000) it has 287 members and is adding two or three a week.

이 백서의 최신 개정판 (2000 년 11 월)에는 287 명의 회원이 있으며 일주일에 2 ~ 3 명이 추가됩니다.

Actually, when I revised in late May 1997 I found the list was beginning to lose members from its high of close to 300 for an interesting reason.

사실, 1997 년 5 월 말에 개정했을 때 흥미로운 이유 때문에 300 명에 가까운 회원 수를 상회하는 회원이되었다는 것을 알았습니다.

Several people have asked me to unsubscribe them because fetchmail is working so well for them that they no longer need to see the list traffic!

fetchmail이 더 이상 목록 트래픽을 볼 필요가 없어서 여러 사람들이 나에게 구독 취소를 요청했습니다!

Perhaps this is part of the normal life-cycle of a mature bazaar-style project.

아마도 이것은 성숙한 바자 스타일 프로젝트의 정상적인 라이프 사이클의 일부입니다.