Coders at Work

Reflections on the Craft of Programming

Peter Seibel

Apress®

Coders at Work

ISBN-13 (pbk): 978-1-4302-1948-4
ISBN-13 (electronic): 978-1-4302-1949-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jeffrey Pepper
Technical Reviewer: John Vacca

Editorial Board: Clay Andres, Steve Anglin, Mark Beckner, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh

Coordinating Editor: Anita Castro
Copy Editor: Candace English
Production Manager: Frank McGuckin
Cover Designer: Anna Ishschenko
Manufacturing Managers: Tom Debolsky; Michael Short

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact us by e-mail at info@apress.com, or visit `http://www.apress.com`.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at `http://www.apress.com/info/bulksales`.

For Amelia

Contents

About the Author

Peter Seibel is either a writer turned programmer or programmer turned writer. After picking up an undergraduate degree in English and working briefly as a journalist, he was seduced by the web. In the early '90s he hacked Perl for *Mother Jones Magazine* and Organic Online. He participated in the Java revolution as an early employee at WebLogic and later taught Java programming at UC Berkeley Extension. In 2003 he quit his job as the architect of a Java-based transactional messaging system, planning to hack Lisp for a year. Instead he ended up spending two years writing the Jolt Productivity Award–winning *Practical Common Lisp*. Since then he's been working as chief monkey at Gigamonkeys Consulting, learning to train chickens, practicing Tai Chi, and being a dad. He lives in Berkeley, California, with his wife Lily, daughter Amelia, and dog Mahlanie.

Acknowledgments

First of all I want to thank my subjects who gave generously of their time and without whom this book would be nothing but a small pamphlet of unanswered questions. Additional thanks go to Joe Armstrong and Bernie Cosell, and their families, for giving me a place to stay in Stockholm and Virginia. Extra thanks also go to Peter Norvig and Jamie Zawinski who, in addition to taking their own turns speaking into my recorders, helped me get in touch with other folks who became my subjects.

As I traveled around the world conducting interviews several other families also welcomed me into their homes: thanks for their hospitality go to Dan Weinreb and Cheryl Moreau in Boston, to Gareth and Emma McCaughan in Cambridge, England, and to my own parents who provided a great base of operations in New York city. Christophe Rhodes helped me fill some free time between interviews with a tour of Cambridge University and he and Dave Fox rounded out the evening with dinner and a tour of Cantabrigian pubs.

Dan Weinreb, in addition to being my Boston host, has been my most diligent reviewer of all aspects of the the book since the days when I was still gathering names of potential subjects. Zach Beane, Luke Gorrie, Dave Walden and my mom also all read chapters and provided well-timed encouragement. Zach additionally—as is now traditional with my books—provided some words to go on the cover; this time the book's subtitle. Alan Kay made the excellent suggestion to include Dan Ingalls and L Peter Deutsch. Scott Fahlman gave me some useful background on Jamie Zawinski's early career and Dave Walden sent historical materials on Bolt Beranek and Newman to help me prepare for my interview with Bernie Cosell. To anyone I have forgotten, you still have my thanks and also my apologies.

Thanks to the folks at Apress, especially Gary Cornell who first suggested I do this book, John Vacca and Michael Banks for their suggestions, and my copy editor Candace English who fixed innumerable errors.

Finally, deepest thanks to my family, extended and nuclear. Both of my moms, biological and in-law, came on visits to watch the the kid and let me get some extra work done; my parents gave my wife and kid a place to escape for a week so I could make another big push. And most of all, thanks to the wife and kid themselves: Lily and Amelia, while I may occasionally need some time to myself to do the work, without you guys in my life, it wouldn't be worth doing. I love you.

Introduction

Leaving aside the work of Ada Lovelace—the 19th century countess who devised algorithms for Charles Babbage's never-completed Analytical Engine—computer programming has existed as a human endeavor for less than one human lifetime: it has been only 68 years since Konrad Zuse unveiled his Z3 electro-mechanical computer in 1941, the first working general-purpose computer. And it's been only 64 years since six women—Kay Antonelli, Jean Bartik, Betty Holberton, Marlyn Meltzer, Frances Spence, and Ruth Teitelbaum—were pulled from the ranks of the U.S. Army's "computer corps", the women who computed ballistics tables by hand, to become the first programmers of ENIAC, the first general-purpose electronic computer. There are many people alive today—the leading edge of the Baby Boom generation and all of the Boomers' parents—who were born into a world without computer programmers.

No more, of course. Now the world is awash in programmers. According to the Bureau of Labor Statistics, in the United States in 2008 approximately one in every 106 workers—over 1.25 million people—was a computer programmer or software engineer. And that doesn't count professional programmers outside the U.S. nor the many student and hobbyist programmers and people whose official job is something else but who spend some or even a lot of their time trying to bend a computer to their will. Yet despite the millions of people who have written code, and the billions, if not trillions of lines of code written since the field began, it still often feels like we're still making it up as we go along. People still argue about what programming is: mathematics or engineering? Craft, art, or science? We certainly argue—often with great vehemence—about the best way to do it: the Internet overflows with blog articles and forum postings about this or that way of writing code. And bookstores are chock-a-block with books about new programming languages, new methodologies, new ways of thinking about the task of programming.

This book takes a different approach to getting at what programming is, following in the tradition established when the literary journal *The Paris*

Review sent two professors to interview the novelist E.M. Forster, the first in a series of Q&A interviews later collected in the book *Writers at Work*.

I sat down with fifteen highly accomplished programmers with a wide range of experiences in the field—heads down systems hackers such as Ken Thompson, inventor of Unix, and Bernie Cosell, one of the original implementers of the ARPANET; programmers who combine strong academic credentials with hacker cred such as Donald Knuth, Guy Steele, and Simon Peyton Jones; industrial researchers such as Fran Allen of IBM, Joe Armstrong of Ericsson, and Peter Norvig at Google; Xerox PARC alumni Dan Ingalls and L Peter Deutsch; early Netscape implementers Jamie Zawinski and Brendan Eich; folks involved in the design and implementation of the languages the present-day web, Eich again as well as Douglas Crockford and Joshua Bloch; and Brad Fitzpatrick, inventor of Live Journal, and an able representative of the generation of programmers who came of age with the Web.

I asked these folks about programming: how they learned to do it, what they've discovered along the way, and what they think about its future. More particularly, I tried to get them to talk about the issues that programmers wrestle with all the time: How should we design software? What role do programming languages play in helping us be productive or avoid errors? Are there ways we can make it easier to track down hard-to-find bugs?

As these are far from settled questions, it's perhaps unsurprising that my subjects sometimes had quite varied opinions. Jamie Zawinski and Dan Ingalls emphasized the importance of getting code up and running right away while Joshua Bloch described how he designs APIs and tests whether they can support the code he wants to write against them before he does any implementation and Donald Knuth described how he wrote a complete version of his typesetting software TeX in pencil before he started typing in any code. And while Fran Allen lay much of the blame for the decline in interest in computer science in recent decades at the feet of C and Bernie Cosell called it the "biggest security problem to befall modern computers", Ken Thompson argued that security problems are caused by programmers, not their programming languages and Donald Knuth described C's use of pointers as one of the "most amazing improvements in notation" he's seen. Some of my subjects scoffed at the notion that formal proofs could be useful

in improving the quality of software, but Guy Steele gave a very nice illustration of both their power and their limitations.

There were, however, some common themes: almost everybody emphasized the importance of writing readable code; most of my subjects have found that the hardest bugs to track down are in concurrent code; and nobody seemed to think programming is a solved problem: most are still looking for a better way to write software, whether by finding ways to automatically analyze code, coming up with better ways for programmers to work together, or finding (or designing) better programming languages. And almost everyone seemed to think that ubiquitous multi-core CPUs are going to force some serious changes in the way software is written.
These conversations took place at a particular moment in our field's history, so no doubt some of the topics discussed in this book will fade from urgent present-day issues to historical curiosities. But even in a field as young as programming, history can hold lessons for us. Beyond that, I suspect that my subjects have shared some insights into what programming is and how we could do it better that will be useful to programmers today *and* to programmers several generations from now.

Finally, a note on the title: we chose *Coders at Work* for its resonance with the previously mentioned *Paris Review's Writers at Work* series as well as Apress's book *Founders at Work*, which does for starting a technology company what this book tries to do for computer programming. I realize that "coding" could be taken to refer to only one rather narrow part of the larger activity of programming. Personally I have never believed that it is possible to be a good coder without being a good programmer nor a good programmer without being a good designer, communicator, and thinker. Certainly all of my subjects are all of those and much more and I believe the conversations you are about to read reflect that. Enjoy!

CHAPTER

1

Jamie Zawinski

Lisp hacker, early Netscape developer, and nightclub owner Jamie Zawinski, a.k.a. jwz, is a member of the select group of hackers who are as well known by their three-letter initials as by their full names.

Zawinski started working as a programmer as a teenager when he was hired to hack Lisp at a Carnegie Mellon artificial intelligence lab. After attending college just long enough to discover that he hated it, he worked in the Lisp and AI world for nearly a decade, getting a strange immersion in a fading hacker subculture when other programmers his age were growing up with microcomputers.

He worked at UC Berkeley for Peter Norvig, who has described him as "one of the of the best programmers I ever hired," and later at Lucid, the Lisp company, where he ended up leading the development of Lucid Emacs, later renamed XEmacs, which eventually led to the great Emacs schism, one of the most famous open source forks.

In 1994 he finally left Lucid and the Lisp world to join Netscape, then a fledgling start-up, where he was one of the original developers of the Unix version of the Netscape browser and later of the Netscape mail reader.

In 1998 Zawinski was one of the prime movers, along with Brendan Eich, behind mozilla.org, the organization that took the Netscape browser open source. A year later, discouraged by the lack of progress toward a release, he quit the project and bought a San Francisco nightclub, the DNA Lounge, which he now runs. He is currently devoting his energies to battling the California Department of Alcoholic Beverage Control in an attempt to convert the club to an all-ages venue for live music.

In this interview we talked about, among other things, why C++ is an abomination, the joy of having millions of people use your software, and the importance of tinkering for budding programmers.

Seibel: How did you learn to program?

Zawinski: Wow, it was so long ago I can barely remember it. The first time I really used a computer in a programming context was probably like eighth grade, I think. We had some TRS-80s and we got to goof around with BASIC a little bit. I'm not sure there was even a class—I think it was just like an after-school thing. I remember there was no way to save programs so you'd just type them in from magazines and stuff like that. Then I guess I read a bunch of books. I remember reading books about languages that I had no way to run and writing programs on paper for languages that I'd only read about.

Seibel: What languages would that have been?

Zawinski: APL, I remember, was one of them. I read an article about it and thought it was really neat.

Seibel: Well, it saves having to have the fancy keyboard. When you were in high school did you have any classes on computers?

Zawinski: In high school I learned Fortran. That's about it.

Seibel: And somehow you got exposed to Lisp.

Zawinski: I read a lot of science fiction. I thought AI was really neat; the computers are going to take over the world. So I learned a little bit about that. I had a friend in high school, Dan Zigmond, and we were trading books, so we both learned Lisp. One day he went to the Apple Users Group meeting at Carnegie Mellon—which was really just a software-trading situation—because he wanted to get free stuff. And he's talking to some college student there who's like, "Oh, here's this 15-year-old who knows Lisp; that's novel; you should go ask Scott Fahlman for a job." So Dan did. And Fahlman gave him one. And then Dan said, "Oh, you should hire my friend too," and that was me. So Fahlman hired us. I think his motivation had to be something along the lines of, Wow, here are two high school kids who are actually interested in this stuff; it doesn't really do me much harm to let them hang out in the lab." So we had basic grunt work—this set of stuff needs to be recompiled because there's a new version of the compiler; go figure out how to do that. Which was pretty awesome. So there are the two of us—these two little kids—surrounded by all these grad students doing language and AI research.

Seibel: Was that the first chance you actually had to run Lisp, there at CMU.

Zawinski: I think so. I know at one point we were goofing around with XLISP, which ran on Macintoshes. But I think that was later. I learned how to program for real there using these PERQ workstations which were part of the Spice project, using Spice Lisp which became CMU Common Lisp. It was such an odd environment. We'd go to weekly meetings, learning how software development works just by listening in. But there were some really entertaining characters in that group. Like the guy who was sort of our manager—the one keeping an eye on us—Skef Wholey, was this giant blond-haired, barbarian-looking guy. Very intimidating-looking. And he didn't talk much. I remember a lot of times I'd be sitting there—it was kind of an open-plan cubicle kind of thing—working, doing something, writing some Lisp program. And he'd come shuffling in with his ceramic mug of beer, bare feet, and he'd just stand behind me. I'd say hi. And he'd grunt or say nothing. He'd just stand there watching me type. At some

point I'd do something and he'd go, "Ptthh, wrong!" and he'd walk away. So that was kind of getting thrown in the deep end. It was like the Zen approach—the master hit me with a stick, now I must meditate.

Seibel: I emailed Fahlman and he said that you were talented and learned very fast. But he also mentioned that you were kind of undisciplined. As he put it, "We tried gently to teach him about working in a group with others and about writing code that you, or someone else, could understand a month from now." Do you remember any of those lessons?

Zawinski: Not the learning of them, I guess. Certainly one of the most important things is writing code you can come back to later. But I'm about to be 39 and I was 15 at the time, so it's all a little fuzzy.

Seibel: What year did that start?

Zawinski: That must have been '84 or '85. I think I started in the summer between 10th and 11th grade. After high school, at 4:00 or so I'd head over there and stay until eight or nine. I don't think I did that every day but I was there a fair amount.

Seibel: And you very briefly went to CMU after you finished high school.

Zawinski: Yeah. What happened was, I hated high school. It was the worst time of my life. And when I was about to graduate I asked Fahlman if he'd hire me full-time and he said, "No, but I've got these friends who've got a startup; go talk to them." Which was Expert Technologies—ETI. I guess he was on their board. They were making this expert system to automatically paginate the yellow pages. They were using Lisp and I knew a couple of the people already who had been in Fahlman's group. They hired me and that was all going fine, and then about a year later I panicked: Oh my god, I completely lucked into both of these jobs; this is never going to happen again. Once I no longer work here I'm going to be flipping burgers if I don't have a college degree, so what I ought to do is go get one of those.

The plan was that I'd be working part-time at ETI and then I'd be going to school part time. That turned into working full-time and going to school full-time and that lasted, I think, six weeks. Maybe it was nine weeks. I know it lasted long enough that I'd missed the add/drop period, so I didn't get any of my money back. But not long enough that I actually got any grades. So it's questionable whether I actually went.

It was just awful. When you're in high school, everyone tells you, "There's a lot of repetitive bullshit and standardized tests; it'll all be better once you're in college." And then you get to your first year of college and they're like, "Oh, no—it gets better when you're in grad school." So it's just same shit, different day—I couldn't take it. Getting up at eight in the morning, memorizing things. They wouldn't let me opt out of this class called Introduction to Facilities where they teach you how to use a mouse. I was like, "I've been working at this university for a year and a half—I know how to use a mouse." No way out of it—"It's policy." All kinds of stuff like that. I couldn't take it. So I dropped out. And I'm glad I did.

Then I worked at ETI for four years or so until the company started evaporating. We were using TI Explorer Lisp machines at ETI so I spent a lot of my time, besides actually working on the expert system, just sort of messing around with user-interface stuff and learning how those machines worked from the bottom up. I loved them—I loved digging around in the operating system and just figuring out how it all fit together.

I'd written a bunch of code and there was some newsgroup where I posted that I was looking for a job and, oh, by the way, here's a bunch of code. Peter Norvig saw it and scheduled an interview. My girlfriend at the time had moved out here to go to UC Berkeley, so I followed her out.

Seibel: Norvig was at Berkeley then?

Zawinski: Yeah. That was a very strange job. They had a whole bunch of grad students who'd been doing research on natural language understanding; they were basically linguists who did some programming. So they wanted someone to take these bits and pieces

of code they'd left behind and integrate them into one thing that actually worked.

That was incredibly difficult because I didn't have the background to understand what in the world they were doing. So this would happen a lot: I'd be looking at something; I'd be completely stuck. I have no idea what this means, where do I go from here, what do I have to read to understand this. So I'd ask Peter. He'd be nice about it—he'd say, "It totally makes sense that you don't understand that yet. I'll sit down and explain it to you Tuesday." So now I've got nothing to do. So I spent a lot of time working on windows system stuff and poking around with screen savers and just the kind of UI stuff that I'd been doing for fun before.

After six or eight months of that it just felt like, wow, I'm really just wasting my time. I'm not doing anything for them, and I just felt like I was on vacation. There have been times when I was working really a lot when I'd look back at that and I'm like, "Why did you quit the vacation job? What is wrong with you? They were paying you to write screen savers!"

So I ended up going to work for Lucid, which was one of the two remaining Lisp-environment developers. The thing that really made me decide to leave was just this feeling that I wasn't accomplishing anything. And I was surrounded by people who weren't programmers. I'm still friends with some of them; they're good folks, but they were linguists. They were much more interested in abstract things than solving problems. I wanted to be doing something that I could point to and say, "Look, I made this neat thing."

Seibel: Your work at Lucid eventually gave rise to XEmacs, but when you went there originally were you working on Lisp stuff?

Zawinski: Yeah, one of the first projects I worked on was—I can't even remember what the machine was, but it was this 16-processor parallel computer and we had this variant of Lucid Common Lisp with some control structures that would let you fork things out to different processors.

I worked a little bit on the back end of that to make the overhead of spawning a thread lower so you could do something like a parallel implementation of Fibonacci that wasn't just completely swamped by the overhead of creating a new stack group for each thread. I really enjoyed that. It was the first time I'd gotten to use a fairly bizarre machine like that.

Before that I was bringing up Lisp on new machines. Which means basically someone's already written the compiler back end for the new architecture and then they've compiled the bootstrap piece of code. So I've got this file full of binary, supposedly executable code for this other machine and now I've got to decipher their loader format so that I can write a little C program that will load that in, make the page executable, and jump to it. Then, hopefully, you get a Lisp prompt and at that point you can start loading things in by hand.

Which for every architecture was bizarre, because it's never documented right. So it's a matter of compiling a C program and then looking at it byte by byte—byte-editing it in Emacs. Let's see what happens if I change this to a zero; does it stop running?

Seibel: When you say it wasn't documented right, was it that it wasn't documented correctly, or it wasn't documented at all?

Zawinski: It was usually documented and it was usually wrong. Or maybe it was just three revisions behind—who knows? But at some point you tweak a bit and then it would no longer believe this was an executable file and you had to figure out what was going on there.

Seibel: So that's something that comes up all the time, from the lowest-level systems programming to high-level APIs, where things just don't work the way you expect or the way they are documented. How do you deal with that?

Zawinski: Well, you just come to expect it. The sooner you realize that your map is wrong, the sooner you'll be able to figure out where it went wrong. In my case, I'm trying to produce an executable file. Well, I know the C compiler will produce one. Take the good one and

start converting it into the bad one until it stops working. That's primary tool of reverse engineering.

The hardest bug I've ever fixed, I think, was probably during that period at Lucid. I'd gotten to the point where it's running the executable and it's trying to bootstrap Lisp and it gets 500 instructions in and crashes. So there I am leaning on the S key, stepping through trying to figure out where it crashes. And it seems to be crashing at a different place each time. And it doesn't make any sense. I'm reading the assembly output of this architecture I only barely understand. Finally I realize, "Oh my god, it's doing something different when I step; maybe it's timing-based." Eventually I figure out that what's going on is this is one of the early machines that did speculative execution. It would execute both sides of the branch. And GDB would always take the branch if you single-stepped past a branch instruction. There was a bug in GDB!

Seibel: Nice.

Zawinski: Right. So then that takes me down into, "Oh my god; now I'm trying to debug GDB, which I've never looked at before." The way to get around that is you're coming up to a branch instruction and you stop before the branch, set a break point on both sides, and continue. So that was how I proved that really was what was going on. Spent like a week trying to fix GDB; couldn't figure it out. I assume a register was getting stomped somewhere, so it always thought there was a positive value in the branch check or something like that.

So I changed the step-by-instruction command to recognize when it was coming up on a branch instruction and just say, "No, don't do that." Then I can just lean on the S key and it would eventually stop and I'd set the break point by hand and continue. When you're debugging something and then you figure out that not only is the map wrong but the tools are broken—that's a good time.

Working on Lisp systems was especially weird because GDB was completely nonfunctional on Lisp code because it doesn't have any debug info—it's written by a compiler GDB has never heard of. I think on some platforms it laid out the stack frames in a way GDB didn't

understand. So GDB was pretty much an assembly stepper at that point. So you wanted to get out of the GDB world just as quickly as you could.

Seibel: And then you'd have a Lisp debugger and you'd be all set.

Zawinski: Right, yeah.

Seibel: So somewhere in there Lucid switched directions and said, "We're going to make a C++ IDE".

Zawinski: That had been begun before I started working there—it was in progress. And people started shifting over from the Lisp side to the Energize side, which is what the development environment was called. It was a really good product but it was two or three years too early. Nobody, at least on the Unix side, had any idea they wanted it yet. Everyone uses them now but we had to spend a lot of time explaining to people why this was better than vi and GCC. Anyway, I'd done a bit of Emacs stuff. I guess by that point I'd already rewritten the Emacs byte compiler because—why did I do that? Right, I'd written this Rolodex phone/address-book thing.

Seibel: Big Brother Database?

Zawinski: Yeah. And it was slow so I started digging into why it was slow and I realized, oh, it's slow because the compiler sucks. So I rewrote the compiler, which was my first run-in with the intransigence of Stallman. So I knew a lot about Emacs.

Seibel: So the change to the byte compiler, did it change the byte-code format or did it just change the compiler?

Zawinski: It actually had a few options—I made some changes at the C layer, the byte-code interpreter, added a few new instructions that sped things up. But the compiler could be configured to emit either old-style byte-code or ones that took advantage of the new codes.

So I write a new compiler and Stallman's response is, "I see no need for this change." And I'm like, "What are you talking about? It generates way faster code." Then his next response is, "Okay, uh,

send me a diff and explain each line you changed." "Well, I didn't do that—I rewrote it because the old one was crap." That was not OK. The only reason it ever got folded in was because I released it and thousands of people started using it and they loved it and they nagged him for two years and finally he put it in because he was tired of being nagged about it.

Seibel: Did you sign the papers assigning the copyright to the Free Software Foundation?

Zawinski: Oh yeah, I did that right away. I think that was probably the first thing in the email. It was like, send me a diff for each line and sign this. So I signed and said, "I can't do the rest; can't send you a diff; that's ridiculous. It's well documented; go take a look." I don't think he ever did.

There's this myth that there was some legal issue between Lucid and FSF and that's absolutely not true—we assigned copyrights for everything we did to them. It was convenient for them to pretend we hadn't at certain times. Like, we actually submitted the paperwork multiple times because they'd be like, "Oh, oh, we seem to have lost it." I think there was some kind of brouhaha with assignments and XEmacs much later, but that was way after my time.

Seibel: So you started with Lisp. But you obviously didn't stick with it for your whole career. What came next?

Zawinski: Well, the next language I did any serious programming in after Lisp was C, which was kind of like going back to the assembly I programmed on an Apple II. It's the PDP-11 assembler that thinks it's a language. Which was, you know, unpleasant. I'd tried to avoid it for as long as possible. And C++ is just an abomination. Everything is wrong with it in every way. So I really tried to avoid using that as much as I could and do everything in C at Netscape. Which was pretty easy because we were targeting pretty small machines that didn't run C++ programs well because C++ tends to bloat like crazy as soon as you start using any libraries. Plus the C++ compilers were all in flux—there were lots of incompatibility problems. So we just settled on ANSI C from the beginning and that served us pretty well. After that

Java felt like going back to Lisp a bit in that there were concepts that the language wasn't bending over backwards trying to make you avoid—that were comfortable again.

Seibel: Like what?

Zawinski: Memory management. That functions felt more like functions than subroutines. There was much more enforced modularity to it. It's always tempting to throw in a goto in C code just because it's easy.

Seibel: So these days it seems like you're mostly doing C and Perl.

Zawinski: Well, I don't really program very much anymore. Mostly I write stupid little Perl scripts to keep my servers running. I end up writing a lot of goofy things for getting album art for MP3s I have—that kind of thing. Just tiny brute-force throw-away programs.

Seibel: Do you like Perl or is it just handy?

Zawinski: Oh, I despise it. It's a horrible language. But it is installed absolutely everywhere. Any computer you sit down on, you're never going to have to talk someone through installing Perl to run your script. Perl is there already. That's really the one and only thing that recommends it.

It has an OK collection of libraries. There's often a library for doing the thing you want. And often it doesn't work very well, but at least there's something. The experience of writing something in Java and then trying to figure out—I myself have trouble installing Java on my computer—it's horrible. I think Perl is a despicable language. If you use little enough of it, you can make it kind of look like C—or I guess more like JavaScript than like C. Its syntax is crazy, if you use it. Its data structures are a mess. There's not a lot good about it.

Seibel: But not as bad as C++.

Zawinski: No, absolutely not. It's for different things. There's stuff that would be so much easier to write in Perl or any language like Perl than in C just because they're text-oriented—all these so-called

"scripting languages". Which is a distinction I don't really buy—"programming" versus "scripting". I think that's nonsense. But if what you're doing is fundamentally manipulating text or launching programs, like running wget and pulling some HTML out and pattern-matching it, it's going to be easier to do that in Perl than even Emacs Lisp.

Seibel: To say nothing of, Emacs Lisp is not going to be very suitable for command-line utilities.

Zawinski: Yeah, though I used to write just random little utilities in Emacs all the time. There was actually a point, early on in Netscape, where part of our build process involved running "emacs -batch" to manipulate some file. No one really appreciated that.

Seibel: No. I imagine they wouldn't. What about XScreenSaver—do you still work on that?

Zawinski: I still write new screen savers every now and then just for kicks, and that's all C.

Seibel: Do you use some kind of IDE for that?

Zawinski: I just use Emacs, mostly. Though recently, I ported XScreenSaver to OS X. The way I did that was I reimplemented Xlib in terms of Cocoa, the Mac graphics substrate, so I wouldn't have to change the source code of all the screen savers. They're still making X calls but I implemented the back end for each of those. And that was in Objective C, which actually is a pretty nice language. I enjoyed doing that. It definitely feels Java-like in the good ways but it also feels like C. Because it's essentially C, you can still link directly with C code and just call the functions and not have to bend over backwards.

Seibel: At Lucid, leaving aside the politics of Emacs development, what technical stuff did you learn?

Zawinski: I definitely became a better programmer while I was there. Largely because that was really the smartest group of people I've been around. Everyone who worked there was brilliant. And it was just nice to be in that kind of environment where when someone says, "That's

nonsense," or "We should do it this way," you can just take their word for it, believe that they know what they were talking about. That was really nice. Not that I hadn't been around smart people before. But it was just such a high-quality group of people there, consistently.

Seibel: And how big was the development team?

Zawinski: I think there were like 70 people at the company so probably; I don't know, 40 or so on the development team. The Energize team was maybe 25 people, 20. It was divided up into pretty distinct areas. There were the folks working on the compiler side of things and the back-end database side of things. The GUI stuff that wasn't Emacs. And then there was, at one point, me and two or three other people working on integrating Emacs with the environment. That eventually turned into mostly me working on mostly Emacs stuff, trying to make our Emacs 19 be usable, be an editor that doesn't crash all the time, and actually runs all the Emacs packages that you expect it to run.

Seibel: So you wanted the Emacs included in your product to be a fully capable version of Emacs.

Zawinski: The original plan was that we wouldn't include Emacs with our product. You have Emacs on your machine already and you have our product and they work together. And you had GCC on your machine already and our product, and they work together. I think one of the early code names for our product was something like Hitchhiker because the idea was that it would take all the tools that you already have and integrate them—make them talk to each other by providing this communication layer between them.

That didn't work out at all. We ended up shipping our version of GCC and GDB because we couldn't get the changes upstream fast enough, or at all in some cases. And same thing with Emacs. So we ended up shipping the whole thing. We ended up going down the path of, "Well, we're replacing Emacs. Shit. I guess we have to do that so we better make it work." One thing I spent a bunch of time on was making the vi emulation mode work.

Seibel: And that's several weeks of your life you're never going to get back.

Zawinski: That's true, yeah. It was challenging. I think it ended up working OK. The real problem with that wasn't so much that it was emulating vi wrong as that vi users quit and restart vi all the time. And no amount of coding on my part is going to get them out of that mindset. So they're like, "I expected this to launch in half a second and it's taking 14 seconds. That's ridiculous. I can't use this."

Seibel: Why did you leave Lucid?

Zawinski: Lucid was done. There'd been a bunch of layoffs. I sent mail to a bunch of people I know saying, "Hey, looks like I'm going to need a new job soon" and one of those people was Marc Andreessen and he said, "Oh, funny you should mention that, because we just started a company last week." And that was that.

Seibel: So you went to Netscape. What did you work on there?

Zawinski: I pretty much started right away doing the Unix side of the browser. There had been maybe a few days' worth of code written on it so far. A little bit more of the Windows and Mac sides had been started. The model was a big pile of back-end code and then as small as possible a piece of front-end code for each of the three platforms.

Seibel: And was this all new code?

Zawinski: It was all new code. Most of the Netscape founders had been NCSA/Mosaic developers so they had written the various versions of NCSA/Mosaic, which was actually three different programs. And all six of those people were at Netscape. They weren't reusing any code but they had written this program before.

Seibel: So they started with an empty disk and started typing?

Zawinski: Exactly. I never looked at the Mosaic code; still haven't. We actually were sued over that at one point; the university claimed that we were reusing their code and I guess that was settled one way

or the other. There's always been that rumor that we started that way, but we didn't.

And really, why would we? Everyone wants to write version two, right? You were figuring it out while you wrote it and now you've got a chance to throw that away and start over—of course you're going to start over. It's going to be better this time. And it was. With the design that the other ones had, there was basically no way to load images in parallel, things like that. And that was really important. So we had a better design for the back end.

Seibel: Yet that's also a classic opportunity to fall into the second-system syndrome.

Zawinski: It is, it is.

Seibel: How did you guys avoid that?

Zawinski: We were so focused on deadline it was like religion. We were shipping a finished product in six months or we were going to die trying.

Seibel: How did you come up with that deadline?

Zawinski: Well, we looked around at the rest of the world and decided, if we're not done in six months, someone's going to beat us to it so we're going to be done in six months.

Seibel: Given that you picked the date first, you had to rein in scope or quality. How did that work?

Zawinski: We spent a long time talking about features. Well, not a long time, but it seemed like a long time because we were living a week every day. We stripped features, definitely. We had a whiteboard; we scribbled ideas; we crossed them out. This was a group of like six or seven people. I don't remember exactly the number. A bunch of smart, egotistical people sitting in a room yelling at each other for a week or so.

Seibel: Six or seven being the whole Netscape development team or the Unix development team?

Zawinski: That was the whole client team. There were also the server folks who were implementing their fork of Apache, basically. We didn't talk to them much because we were busy. We had lunch with them, but that was it. So we figured out what we wanted to be in the thing and we divided up the work so that there were, I guess, no more than two people working on any part of the project. I was doing the Unix side and Lou Montulli did most of back-end network stuff. And Eric Bina was doing layout and Jon Mittelhauser and Chris Houck were doing the Windows front end and Aleks Totić and Mark Lanett were doing the Mac front end for the pre–version 1.0 team. Those teams grew a little bit after that. But we'd have our meetings and then go back to our cubicles and be heads-down for 16 hours trying to make something work.

It was really a great environment. I really enjoyed it. Because everyone was so sure they were right, we fought constantly but it allowed us to communicate fast. Someone would lean over your cubicle and say, "What the fuck did you check in; that's complete bullshit—you can't do it that way. You're an idiot." And you'd say, "Fuck off!" and go look at it and fix it and check it in. We were very abrasive but we communicated fast because you didn't have to go blow sunshine up someone's ass and explain to them what you thought was wrong—you could say, "Hey, that's a load of shit! I can't use that." And you'd hash it out very quickly. It was stressful but we got it done pretty quickly.

Seibel: Are the long hours and the intensity required to produce software quickly?

Zawinski: It's certainly not healthy. I know we did it that way and it worked. So the way to answer that question is, is there another example of someone delivering a big piece of software that fast that's of reasonable quality where they actually had dinner at home and slept during the night? Has that ever happened? I don't actually know. Maybe it has.

But it's not always about getting it done as quickly as possible. It also would be nice to not burn out after two years and be able to continue doing your job for ten. Which is not going to happen if you're working 80-plus hours a week.

Seibel: What is the thing that you worked on that you were most proud of.

Zawinski: Really just the fact that we shipped it. The whole thing. I was very focused on my part, which was the user interface of the Unix front end. But really just that we shipped the thing at all and that people liked it. People converted immediately from NCSA Mosaic and were like, "Wow, this is the greatest thing ever." We had the button for the What's Cool page up in the toolbar and got to show the world these crazy web sites people had put up already. I mean, there were probably almost 200 of them! It's not so much that I was proud of the code; just that it was done. In a lot of ways the code wasn't very good because it was done very fast. But it got the job done. We shipped—that was the bottom line.

That first night when we put up the .96 beta, we were all sitting around the room watching the downloads with sound triggers hooked up to it—that was amazing. A month later two million people were running software I'd written. It was unbelievable. That definitely made it all worthwhile—that we'd had an impact on people's lives; that their day was more fun or more pleasant or easier because of the work we'd done.

Seibel: After this relentless pace, at some point that has to start to catch up with you in terms of the quality of the code. How did you guys deal with that?

Zawinski: Well, the way we dealt with that was badly. There's never a time to start over and rewrite it. And it's never a good idea to start over and rewrite it.

Seibel: At some point you also worked on the mail reader, right?

Zawinski: In 2.0 Marc comes into my cubicle and says, "We need a mail reader." And I'm like, "OK, that sounds cool. I've worked on mail readers before." I was living in Berkeley and basically I didn't come into the office for a couple weeks. I was spending the whole time sitting in cafes doodling, trying to figure out what I wanted in a mail reader. Making lists, crossing it off, trying to decide how long it would take me. What should the UI look like?

Then I came back and started coding. And then Marc comes in again and says, "Oh, so we hired this other guy who's done mail stuff before. You guys should work together." It's this guy Terry Weissman, who was just fantastic—we worked together so well. And it was a completely different dynamic than it had been in the early days with the rest of the browser team.

We didn't yell at each other at all. And the way we divided up labor, I can't imagine how it possibly worked or could ever work for anyone. I had the basic design done and I'd started doing a little coding and every day or every couple of days we'd look at the list of features and I'd go, "Uhhh, maybe I'll work on that," and he'd go, "OK, I'll work on that," and then we'd go away.

Check-ins would happen and then we'd come back and he'd say, "Alright, I'm done with that, what are you doing?" "Uh, I'm working on this." "OK, well, I'll start on that then." And we just sort of divided up the pieces. It worked out really well.

We had disagreements—I thought we had to toss filtering into folders because we just didn't have time to do it right. And he was like, "No, no, I really think we ought to do that." And I was like, "We don't have time!" So he wrote it that night.

The other thing was, Terry and I rarely saw each other because he lived in Santa Cruz and I lived in Berkeley. We were about the same distance from work in opposite directions and because the two of us were the only two who ever needed to communicate, we were just like, "I won't make you come in if you don't make me come in." "Deal!"

Seibel: Did you guys email a lot?

Zawinski: Yeah, constant email. This was before instant messaging—these days it probably all would have been IM because we were sending one-liner emails constantly. And we talked on the phone.

So we shipped 2.0 with the mail reader and it was well-received. Then we're working on 2.1, which is the version of the mail reader that I'm starting to consider done—this is the one with all the stuff that we couldn't ship the first time around. Terry and I are halfway through doing that and Marc comes in and says, "So we're buying this company. And they make a mail-reader thing that's kind of like what you guys did." I'm like, "Oh. OK. Well, we have one of those." And he says, "Well, yeah, but we're growing really fast and it's really hard to hire good people and sometimes the way you hire good people is you just acquire another company because then they've already been vetted for you." "OK. What are these people going to be working on?" "They're going to be working on your project." "OK, that kind of sucks—I'm going to go work on something else."

So basically they acquired this company, Collabra, and hired this whole management structure above me and Terry. Collabra has a product that they had shipped that was similar to what we had done in a lot of ways except it was Windows-only and it had utterly failed in the marketplace.

Then they won the start-up lottery and they got acquired by Netscape. And, basically, Netscape turned over the reins of the company to this company. So rather than just taking over the mail reader they ended up taking over the entire client division. Terry and I had been working on Netscape 2.1 when the Collabra acquisition happened and then the rewrite started. Then clearly their Netscape 3.0 was going to be extremely late and our 2.1 turned into 3.0 because it was time to ship something and we needed it to be a major version.

So the 3.0 that they had begun working on became 4.0 which, as you know, is one of the biggest software disasters there has ever been. It basically killed the company. It took a long time to die, but that was it: the rewrite helmed by this company we'd acquired, who'd never

accomplished much of anything, who disregarded all of our work and all of our success, went straight into second-system syndrome and brought us down.

They thought just by virtue of being here, they were bound for glory doing it their way. But when they were doing it their way, at their company, they failed. So when the people who had been successful said to them, "Look, really, don't use C++; don't use threads," they said, "What are you talking about? You don't know anything."

Well, it was decisions like not using C++ and not using threads that made us ship the product on time. The other big thing was we always shipped all platforms simultaneously; that was another thing they thought was just stupid. "Oh, 90 percent of people are using Windows, so we'll focus on the Windows side of things and then we'll port it later." Which is what many other failed companies have done. If you're trying to ship a cross-platform product, history really shows that's how you don't do it. If you want it to really be cross-platform, you have to do them simultaneously. The porting thing results in a crappy product on the second platform.

Seibel: Was the 4.0 rewrite from scratch?

Zawinski: They didn't start from scratch with a blank disk but they eventually replaced every line of code. And they used C++ from the beginning. Which I fought against so hard and, dammit, I was right. It bloated everything; it introduced all these compatibility problems because when you're programming C++ no one can ever agree on which ten percent of the language is safe to use. There's going to be one guy who decides, "I have to used templates." And then you discover that there are no two compilers that implement templates the same way.

And when your background, your entire background, is writing code where multiplatform means both Windows 3.1 and Windows 95, you have no concept how big a deal that is. So it made the Unix side of things—which thankfully was no longer my problem—a disaster. It made the Mac side of things a disaster. It meant it was no longer possible to ship on low-end Windows boxes like Win16. We had to