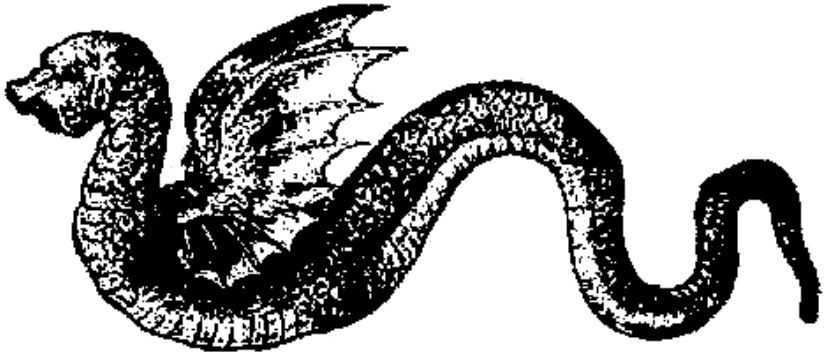
Software Exorcism: A Handbook for Debugging and Optimizing Legacy Code

BILL BLUNDEN



Apress™

ISBN: 1-59059-234-4

Printed and bound in the United States of America 10987654321

Technical Reviewer: Doug Holland

Editorial Board: Dan Appleman, Craig Berry, Gary Cornell, Tony Davis, Steven Rycroft, Julian Skinner, Martin Streicher, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Copy Editor: Ami Knox

Production Manager: Kari Brooks

Compositor and Artist: Kinetic Publishing Services, LLC

Proofreader: Thistle Hill Publishing Services

Indexer: Carol Burbo

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section.

The medieval illustrations that appear on the cover and throughout this book were taken from *Devils, Demons, and Witchcraft: 244 Illustrations for Artists* by Ernst and Johanna Lehner (Dover Publications, ISBN: 0486227510) and were reprinted with permission.

CHAPTER 7

Final Words of Advice

*For the listener who's enjoying my stuff, I would hope that they just take everything they're told by authority figures less seriously—not believe what their parents say, what their teachers say, not believe clergymen, law enforcement people, legislators, business leaders. Because they're being bullsh**ted at every corner.*

—George Carlin, *Inside Borders* interview

Back in 1990, I took an introductory political science course taught by Theodore Lowi and Ben Ginsberg. It was one of those classes where they crammed 500 students into an auditorium. Class felt more like a circus than a lecture. I distinctly remember the last day of class. Ginsberg walked up to the lectern and announced, *"Up until now, there probably hasn't been anyone who has been out to get you. This will change the minute you graduate and go out into the real world."*

As far as I can tell, this was the only useful thing that I learned at Cornell (contrary to popular belief, knowing all about quantum mechanics is not horribly practical). People who are paranoid have enemies that are imaginary. Victims have enemies that they think are imaginary. Both groups of people suffer from their delusions. The only meaningful distinction is that victims tend to suffer more from their mistakes than paranoids.

For those of you about to enter the corporate landscape for the first time, college graduates in particular, I would urge you to consider what Ginsberg had to say. It may sound paranoid, but it's not. Do not be fooled by the cotton-candy fluff that the human resources people feed you, or the glossy brochures that they pass out. There will be people who see you as a threat to their jobs, managers who want to treat you like a disposable diaper, and disgruntled workers who want to vent their frustration on you. In other words, there will be people out to get you.

One of the themes that this book examines is the impact of human behavior. While I have spent much of the book discussing technical issues, I have also tried to address some of the social and environmental forces that can influence the outcome of a software project. Most of the projects that I have seen fail did not fail due to technical challenges. They failed because of behavioral challenges: politics, infighting, witch-hunts, nepotism, backstabbing, and sabotage, just to name a few.

One of my primary motivations for writing this book has been to alert newcomers so that they can learn to spot trouble before it ambushes them. If I can prevent just one person from being victimized, then I will have accomplished my mission. Having said that, I would like to end this book with a few words of advice—advice that I wish someone had given to me back in 1988.

7.1 Other Threats to Source Code Integrity

One of the greatest and least talked about threats to the stability of your source code is fashionable technology. Revamping a code base to cater to the latest flavor of the month will waste more man-hours than any memory leak or race condition. Once more, the return on investment is awful, because six months later something new will come out. Engineers who chase after the next big thing are constantly playing a game of catch-up, and it is a game they can never win.

The most dangerous thing about adopting a fashionable technology is that it tends to constrain your options. When some commercial software vendor is marketing a new development technology, it's in their best interest to sell you something that anchors you to their platform. Anyone who's worked at a movie theater knows that it's easier to gouge a customer for candy when you have a captive audience. By putting the family heirloom (i.e., your code base) in the hands of a third party, you are surrendering control of important, long-term features like portability and flexibility.

Fashionable Technology: A Case Study

Let's take a look at the evolution of Microsoft's development technology. Back in 1987, Windows 2.0 supported Dynamic Data Exchange (DDE), which was basically an interprocess communication (IPC) mechanism that allowed applications to share data. The object linking and embedding (OLE) framework replaced DDE. OLE was geared towards supporting document components that could be cut and pasted between applications. OLE-related tools were first made available to developers in 1991. Two years later, in 1993, OLE 2.0 was released. OLE 2.0 objects were based on a core infrastructure known as the Component Object Model (COM). As time passed, COM became a buzzword in its own right, serving as a foundation for implementing software components in general (not just those related to application documents).

In 1996, Microsoft introduced two new terms: DCOM and ActiveX. ActiveX was a branding name used to describe COM components that provided interactive Web content. ActiveX was Microsoft's response to Java applets. DCOM was Distributed COM, a framework that allowed COM objects on different machines to interact. DCOM was Microsoft's response to CORBA, through which Windows NT 4.0 supplied ORB-like services. DCOM had a couple of serious shortcomings that prevented it from being a serious threat to CORBA, like the inability to support distributed transactions. Microsoft went back to the drawing board and, in late 1997, announced the creation of COM+, which was a merger of COM technology and the Microsoft Transaction Server (MTS). With COM+, Microsoft seemed to be moving away from the client-server topology of DCOM towards a Web-based, server-centric model.

In every case, going with Microsoft meant handcuffing your code to Windows. Sure, there were attempts to provide support on Unix platforms, but they were nothing more than token gestures. Microsoft proponents may claim that the recent .NET initiative, with its virtual machine approach, offers

more alternatives. After all, the nature of a virtual machine is that it can be implemented anywhere, using any set of tools, just as long as the implementation obeys the virtual machine's specification. I suspect, however, that Microsoft's *common language runtime* (CLR) is just a thinly veiled attempt to counter the rising popularity of Java, which has done an admirable job of offering *true* cross-platform support. In my opinion, Microsoft's marketing hype is paying lip service to portability, while at the same time quietly conveying the notion that .NET applications "run best on Windows."

You'd pay to know what you really think.

—J.R. "Bob" Dobbs

Brainwashing 101

In the end, fashionable technology is a ruse, an excuse for you to spend money. The big corporations want your cash, and they will tell you damn near anything to get you to part with it. Everything that they say is tainted with this ulterior motive.

Marketing hype can be very seductive. Even worse, it's everywhere. Half of the technical magazines that you see at the newsstand are nothing more than oversized brochures. On a superficial level, the technical articles that you read may seem like they are trying to "educate" you. The actual agenda is not so philanthropic. This propaganda is intended to subliminally give you the impression of what is "current."

By bombarding you with the same acronym enough times, the media is hoping to encourage the notion that "everyone" is moving to technology XYZ. Their ability to convince people of this is what allows them to charge millions of dollars for advertising space. In so many words, a technology is "current" only because the corporate sponsors are paying them to make it look that way.

As a junior engineer in the early 1990s, I was like a kid in a candy store. The release of Windows 3.1 was accompanied by a rash of slick, sexy-sounding engineering technologies. There wasn't a single new toolkit that I didn't love. At 20 years of age, I was very impressionable. I can recall looking down on all of the veteran engineers and their suspicious attitude. They seemed like crotchety old men who had fallen out of touch with the world. In reality, I was the one who was out of touch.

The Real Issue

The salient issue is not "which solution is current;" this is just a trick that the marketing people use to distract you. The real issue is about *return on investment* (ROI). It's not about being trendy; it's about getting the most bangs per buck. Other than the research firms, like Gartner, Inc. or Forrester Research,

Inc., none of the periodicals seems to pay homage to this topic. Why? The reason that the software industry periodicals shy away from ROI is that their corporate sponsors have expensive products that they want to sell you.

When a CIO decides to roll out a new platform, a venture that can make or break some businesses, the last thing they are worried about is being in fashion. Instead, they have their eyes on long-term financial repercussions. They're focused on satisfying business requirements, minimizing total cost of ownership, availability, compatibility, and safeguarding against vendor lock-in. As John Schindler, CIO of Kichler Lighting, put it, *"If I caught a CIO reading* Byte *magazine, I'd fire him."*

Your goal, as an engineer, should be to adopt this mindset and apply it to software development. Don't be a victim of marketing hype. Renovating significant portions of your code base just to be fashionable is an expensive waste of resources. Ask yourself: "What is this technology really going to buy me? Am I going to get tangible benefits from this new technology, or am I just following the rest of the herd?"

7.2 Maintaining a Paper Trail

When the proverbial crap hits the fan, the best way to defend yourself against fallout, as I have stressed before, is to maintain a well-documented paper trail. Concrete documentation can be used to assign responsibility, and responsibility transforms into blame if a project heads south.

Some managers have been known to save their own skin by blaming things on the other guy. A manager who was rooting and hollering for your project last week, in a staff meeting, can suddenly turn around and stick a shank in your back: "I knew those guys would screw it up, they wouldn't listen to me. I told them it wouldn't work."

Quietly Keep Records

If you are going to maintain a paper trail, do so as inconspicuously as possible. This kind of record keeping can be serious business. People can get fired. You don't want to make your allies nervous, and at the same time you don't want to alert your enemies. Do all of your strategic thinking and analysis at home. Granted, you will still have to collect and record information at work, but there are steps you can take to minimize your footprints when you do.

If you need to huddle with team members to discuss a sensitive issue, go to a bar or a restaurant or any other place in a remote area of town, and do it there. Eavesdropping has been honed to a fine art among cube denizens (which is one reason why managers have offices). Not to mention that a number of financial institutions are required to record telephone conversations to guard against insider trading and disclosure violations.

The Myth of Privacy

When I walked into my first software gig, there was a grizzled old-timer sitting in the cube next to me. He had been with the company for almost 15 years. He had some impressive hardware humming away in his cube, including a fiber optic network connection. Yet, there he was, reading a cheap paperback novel when he could have been surfing the Internet. I was just a little confused at how this technologically savvy early adopter could resist playing with his toys. As I was to find out, he had his reasons . . .

This may sound a little too cloak-and-dagger, but the growing threat of industrial espionage has led many software companies to closely monitor their employees. The idea of privacy in the workplace is a myth. Everything that your employer supplies you with (e.g., a computer, a network connection, a chair, a desk, office supplies) is their property and they can do whatever they want to with it. If they want to, your employer can install a keyboard logger on your workstation to see what you're typing in. They can also legally intercept traffic that you send over the network. This includes e-mails, Web browser downloads, and chat messages. In extreme cases, they can use remote desktop software or TEMPEST equipment to observe everything that you do in real time.

> **WARNING** *The prevalence of IP snooping is one reason why e-mail can be particularly dangerous. Don't EVER e-mail anything at work unless you feel comfortable with the whole world reading it.*

If you want privacy at work, you'll need to buy a laptop and bring it with you. This laptop is your property, not theirs. They have no legal right to install logging software on it. If you suspect that network traffic is being monitored, for $150 you can buy a firewall appliance and set up a VPN tunnel between your laptop and your home computer (assuming you have a home computer). Depending on how your home LAN is set up, you can then reroute incoming traffic from your laptop back out onto the Internet through your home connection, and achieve a modicum of privacy.

> **NOTE** *I'm not sure what to tell you when it comes to TEMPEST equipment. Most companies that manufacture EMF shielding products sell only to the government.*

The bottom line is this: if you are going to keep records so that you can defend yourself later on, collect information unobtrusively and then process it away from prying eyes.

7.3 History Repeats Itself

> *Silicon Valley is like Hollywood; everyone wants to be a movie star. Yet for every company that makes it to an IPO, a thousand go down in flames.*
>
> —Howie Ernesti

In the aftermath of the dot-com bust, the outlook for the software industry doesn't seem very good. Some people are even claiming that the recent collapse of the software industry is an omen of more far-reaching changes. For example, in April 2003, Larry Ellison announced, *"What's going on . . . is the end of Silicon Valley as we know it."*[1]

Ellison predicts that the software industry will mature, in the same way that our steel industry did decades ago. The growing standardization of products will result in thinner profit margins, as larger companies rely on economies of scale to squeeze out the smaller companies and gain dominant market positions. To boost efficiency, the survivors will move operations overseas to take advantage of cheaper labor.

There are those who agree, in part, with Ellison. In November 2002, researchers at Gartner[2] predicted that by the end of 2004, half of the world's software vendors will be acquired or be put out of business. IBM has already embraced the idea of "grid computing," where products are standardized to the extent that they are more like utilities. In October 2002, Sam Palmisano stated that IBM would be committing $10 billion towards "on-demand computing," which aims to turn software services into a commodity.[3]

Then again, Larry has been wrong in the past. In the previous chapter I mentioned his failed Raw Iron initiative, which attempted to replace monolithic database servers with appliances that used a microkernel OS. In the mid-1990s, Ellison also backed two companies to build a "network computer," which would execute all of its programs on a remote service (i.e., essentially the 1990s' equivalent of the serial terminal). Neither of the two companies succeeded.

---

1. Mylene Mangalindan, "Larry Ellison's Sober Vision," *The Wall Street Journal*, April 8, 2003
2. Thomas Topolinski and Joanne Correia, "Prediction 2003: Continued Challenges for Software Industry," Gartner Research, AV-18-8042, November 20, 2002
3. Ludwig Siegele, "At Your Service: Despite Early Failures, Computing Will Eventually Become a Utility," *The Economist*, May 16, 2003

The "New Economy" Hits Home

Ed Yourdon once predicted the demise of the American programmer.[4] While the fate of the entire industry has yet to be seen, I think that Ed has hit the bull's-eye (even if he was a little premature). The sad fact is that software engineering in the U.S., as a career path, has become a quaint anachronism.

Look around you—how many 55-year-old software architects do you see? If anything, software is a young man's game. This is due to two reasons:

- The constantly shifting skill set
- The availability of cheaper substitutes

The skill set that you learn today can be completely supplanted within a year. I remember taking the better part of six months to become comfortable with DCOM, and then "poof," it vanished off the radar as soon as the next fashionable technology (i.e., COM+) appeared. This makes taking the time to understand the latest tools a poor investment in the long run.

I know some older engineers who try to counter this by claiming, "Well, I like to think that my years of experience give me a leg up when it comes to the bigger picture of implementing a solid design and getting a stable product out the door." This may be true, but only as long as you understand the technology being used. All it takes is a year or two to get out of touch with the industry, and that "big picture" explanation sounds more like an excuse for not having to stay up to date. Before you know it, the junior engineers are sneering at you as if you were a COBOL programmer. Hence, even the "big picture" guys will have to relearn everything periodically to keep from looking outmoded.

The short-lived utility of software engineering job skills is compounded by the availability of cheaper substitutes. Every year a whole new batch of kiddies enter the workforce knowing the latest thing. My guess is that they would be willing to do your job for a fraction of what you make. They don't have a mortgage, they don't have children, and they're too naive to be bothered by working 15 hours a day. Most college graduates see pulling a 15-hour workday as heroic, just like charging a machine gun nest (now you know why they draft 18-year-olds).

Thus, software engineering is a great field to go into if you're young. Harried managers are always looking for new blood they can put to work (ahem, exploit). However, after you have spent a few years building up your market value, you will discover that you have morphed into a target for the efficiency experts. If you decide to enter the workforce as a programmer, you should do so with your eyes open. Like professional football, programming is

4. Ed Yourdon, *Decline and Fall of the American Programmer* (Prentice Hall, 1992. ISBN: 0-132-03670-3)

strictly a short-term occupation. Have an exit strategy in place so that your transition out of programming is as smooth as your entrance.

The really big threat to the future of software engineering in the U.S., however, is not here at home. It's overseas. The GNP per capita in India and China is a fraction of what it is in the U.S. Not only that, but the emphasis placed on education in these countries has produced an army of highly trained engineers. This glut of relatively cheap labor is an awful temptation for software vendors. With the availability of gigabit networking equipment, teleconferencing, instant messaging, and e-mail, a branch office in another country can seem like it's right next door.

In 2002, Bank of America laid off 3,700 of its 25,000 IT and back-office employees. That's about 15 percent. In 2003, Bank of America will outsource 1,100 jobs to India.[5] This is by no means an isolated incident. Hardware manufacturers like Intel have been frantically hiring Chinese and Indian engineers, with advanced degrees, to design new processors. Hewlett-Packard, for instance, has 3,300 software engineers in India. Then there's Microsoft. Over the next three years, Microsoft will invest $400 million in India and $750 million in China. Over a billion dollars of capital; think about that.

NOTE *To give you an idea of how bleak things look: John C. McCarthy, an analyst for Forrester Research, predicts that more than 3.3 million white-collar jobs will leave the U.S. for other countries by 2015.*[6]

We've seen this before. Decades ago, this happened to the U.S. steel industry when the means of production moved overseas. History is repeating itself. Only this time it's the white-collar workers who are getting sold out. Back then, political and business leaders proclaimed that the blue-collar workers who lost their jobs would be retrained and redeployed in the IT industry. As we all know, this never happened. What are they going to tell us this time? If I may, I would suggest that you revisit the George Carlin quote that I began the chapter with.

---

5. Peter Engardio et al., "The New Global Job Shift," *Business Week*, February 3, 2003
6. Ibid.