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!