Alex Groce (agroce@gmail.com), Oregon State University

David Agans' *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems* is only worth reading if anything has ever gone wrong for you, in any kind of way, at some point in your life, and you would like to have done a better job fixing it. Perhaps that's an exaggeration -- Agans is not going to help you with a bad breakup or a major illness (almost certainly on the first, and most likely on the second). *Debugging* is intended to help with technical problems, and in that domain with ones where the key (usually) to fixing the problem is simply figuring out what is causing the problem. That covers a large portion of what I do with my life, both at work and sometimes outside it, and it probably does for you, too, if you're reading this column.

Debugging is the indispensable book on debugging that everyone who might need to debug a complex system needs to read. That's not to say *Debugging* is the *only* good book on debugging. Andreas Zeller's *Why Programs Fail: A Guide to Systematic Debugging* is also a great book. However, it's nearly three times as long as *Debugging* and more focused on the introducing you to the academic formalisms and many current technologies and tools for debugging. Zeller's book is something anyone really serious about debugging should read after reading *Debugging*, but what if you're not really serious? It's summer, and a 192 page book is easier to read on the beach than a 544 page book. Also, Zeller's book is ineligible for this column, as it isn't quite 10 years old, and Agans' book dates from 2002.

Debugging is organized around the 9 indispensable rules that are promised in the subtitle. What are those rules? Agans has free posters of the rules on the book's website, listing them, and they are worth stating in full for the prospective reader:

1. Understand the system.
2. Make it fail.
3. Quit thinking and look.
4. Divide and conquer.
5. Change one thing at a time.
6. Keep an audit trail.
7. Check the plug.
8. Get a fresh view.
9. If you didn't fix it, it ain't fixed.

The 9 rules themselves are useful, but they don't, on their own, help a novice debugger become an expert. They are too succinct to work without explanation. That's why you need to read the book, not just print the poster.

In *Debugging* David Agans takes these basic principle and, in a dedicated chapter (Chapters 3-11) for each rule, explains what they mean, why they matter, and, most importantly, gives numerous memorable anecdotes (mostly from his own career) to help build an intuition for when

and how to apply the rules. The anecdotes are so well chosen, and the principles so nicely explained that in some ways reading this book attentively works like a condensed version of having years of experience debugging complex systems, but with the advantage that the experience is pre-digested and organized, so the ideas don't just rattle around in your head, but become guides for action in novel situations. For instance, what does "understand the system" entail? It involves understanding the software or hardware system in question, it involves understanding the operating system, programming language, and environment of the system being debugged. Critically, it involves understanding the code in question. The anecdote for this last point involves searching for "bug" in source code for a system that seemed to be doing things in the wrong order. Sure enough, in the code there was a comment saying "Bug here? Maybe should call these in reverse order?" and that was in fact the problem. So, don't just "read the manual." Read all the manuals. Read the specs. Read the source.

After going through all the rules separately, Agans has a single story (Chapter 12) that brings in all the debugging rules and makes it clear how they fit together. Chapter 13 has some light exercises for the reader, and Chapter 14 (more useful than you might think for SEN readers) has "The View from the Help Desk" with the constraints of the murky and remote view sometimes needed when you help someone else debug in mind.

Is Agans really the debugging book of choice? I can't prove that everyone agrees with this claim, but the only two conversations I am aware of on the topic (http://blog.regehr.org/archives/849 and http://programmers.stackexchange.com/questions/160293/are-there-any-theories-or-books-about-how-to-debug-in-general) seem to reach that conclusion. As John Regehr says, "I wouldn't exactly say that following these rules makes debugging easy, but it's definitely the case that ignoring them will make debugging a lot harder."

A book that makes the most difficult part of programming easier, and is fun to read? That's a classic, from the day it was written.