Raymond's reading list: The Mythical Man-Month, The Design of Everyday Things, and Systemantics

devblogs.microsoft.com/oldnewthing/20080416-00

April 16, 2008

Raymond Chen

The first two of these books are probably on everybody else's reading list, but I'm going to mention them anyway since I consider them required reading for managers and designers. (Note: Links may be sponsored.)

The Mythical Man-Month is over 30 years old, but the lessons contained therein are as true now as they were back in 1975, such as what is now known as Brooks' law: *Adding manpower to a late software product makes it later*.

I much preferred the original title for *The Design of Everyday Things*, namely, *The Psychology of Everyday Things*, but I'm told that booksellers ended up mistakenly filing the book in the psychology section. Once you've read this book, you will never look at a door the same way again. And you'll understand the inside joke when I say, "I bet it won an award."

The third book is the less well-known *Systemantics: How Systems Work and Especially How They Fail*. The book was originally published in 1978, then reissued under the slightly less catchy title, *Systemantics: The Underground Text of Systems Lore*, and re-reissued under the completely soul-sucking title *The Systems Bible*. I reject all the retitling and continue to refer to the book as *Systemantics*.

Systemantics is very much like *The Mythical Man-Month*, but with a lot more attitude. The most important lessons I learned are a reinterpretation of Le Chatelier's Principle for complex systems ("Every complex system resists its proper functioning") and the Fundamental Failure-Mode Theorem ("Every complex system is operating in an error mode").

You've all experienced the Fundamental Failure-Mode Theorem: You're investigating a problem and along the way you find some function that never worked. A cache has a bug that results in cache misses when there should be hits. A request for an object that should be there somehow always fails. And yet the system still worked in spite of these errors. Eventually you trace the problem to a recent change that exposed all of the other bugs. Those bugs were always there, but the system kept on working because there was enough redundancy that one

component was able to compensate for the failure of another component. Sometimes this chain of errors and compensation continues for several cycles, until finally the last protective layer fails and the underlying errors are exposed.

That's why I'm skeptical of people who look at some catastrophic failure of a complex system and say, "Wow, the odds of this happening are astronomical. Five different safety systems had to fail simultaneously!" What they don't realize is that one or two of those systems are failing *all the time*, and it's up to the other three systems to prevent the failure from turning into a disaster. You never see a news story that says "A gas refinery did not explode today because simultaneous failures in the first, second, fourth, and fifth safety systems did not lead to a disaster thanks to a correctly-functioning third system." The role of the failure and the savior may change over time, until eventually all of the systems choose to have a bad day all on the same day, and something goes boom.

Raymond Chen

Follow