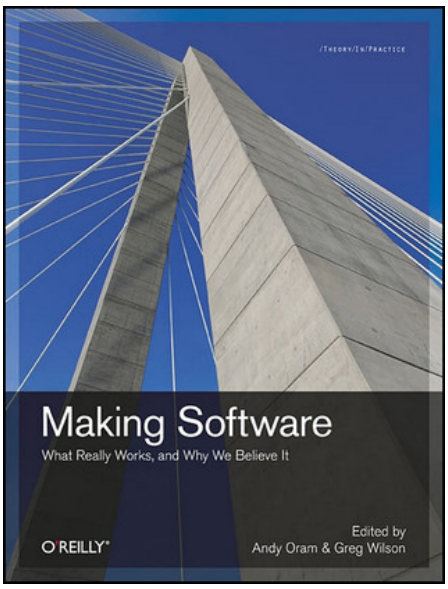

Making Software: What Really Works, and Why We Believe It

Andy Oram , Greg Wilson

Making Software: What Really Works, and Why We Believe It

Andy Oram , Greg Wilson

Making Software: What Really Works, and Why We Believe It Andy Oram , Greg Wilson
Many claims are made about how certain tools, technologies, and practices improve software development. But which claims are verifiable, and which are merely wishful thinking? In this book, leading thinkers such as Steve McConnell, Barry Boehm, and Barbara Kitchenham offer essays that uncover the truth and unmask myths commonly held among the software development community. Their insights may surprise you.

Are some programmers really ten times more productive than others?
Does writing tests first help you develop better code faster?
Can code metrics predict the number of bugs in a piece of software?
Do design patterns actually make better software?
What effect does personality have on pair programming?
What matters more: how far apart people are geographically, or how far apart they are in the org chart?
Contributors include:

Jorge Aranda

Tom Ball

Victor R. Basili

Andrew Begel

Christian Bird

Barry Boehm

Marcelo Cataldo

Steven Clarke

Jason Cohen

Robert DeLine

Madeline Diep

Hakan Erdogmus

Michael Godfrey

Mark Guzdial

Jo E. Hannay

Ahmed E. Hassan

Israel Herraiz

Kim Sebastian Herzig

Cory Kapser

Barbara Kitchenham

Andrew Ko

Lucas Layman

Steve McConnell

Tim Menzies

Gail Murphy

Nachi Nagappan

Thomas J. Ostrand

Dewayne Perry

Marian Petre

Lutz Prechelt

Rahul Premraj

Forrest Shull

Beth Simon

Diomidis Spinellis

Neil Thomas

Walter Tichy

Burak Turhan

Elaine J. Weyuker

Michele A. Whitecraft

Laurie Williams

Wendy M. Williams

Andreas Zeller

Thomas Zimmermann

Making Software: What Really Works, and Why We Believe It Details

Date	:	Published October 27th 2010 by O'Reilly Media (first published January 1st 2010)
ISBN	:	9780596808327
Author	:	Andy Oram , Greg Wilson
Format	:	Paperback 624 pages
Genre	:	Computer Science, Programming, Software, Science, Technical, Nonfiction

From Reader Review Making Software: What Really Works, and Why We Believe It for online ebook

Melvin says

This book is a collection of science on software engineering.

You most definitely wil learn something interesting when reading this book.
(Eg the harsh drop of finding bugs after 1 hour of code reviews)

However you'll also find that the state of scientific knowledge of software engineering is rather meager.
Most evidence comes from very small studies performed under harsh scientific conditions.
This book was very honest in that.

This book is not a must read, especially since the evidence is not really convincing on lots of topics.
But I found it rather interesting

Dav says

A collection of essays. I only read a few so far. The one on an academic survey study of Test Driven Development was unable to conclusively show whether TDD was effective or not, but all of the researchers claimed to believe in it anyhow and think the studies surveyed have not been constructed well enough to limn the benefits. Sigh.

Chris says

This book is an embarrassment. The only thing worse than having no clue what you're doing or why you're doing it is pseudoscience. Which this book is.

Matt says

Take a hundred bad academic computer science papers and put them in a stack from the best on top to the worst on the bottom. Flip the stack over and bind the thirty that are now on top. That's my take on this book. Even Steve McConnell's chapter at the end was lackluster.

This book was **boring**, but, more importantly, it was full of anecdotal data that most certainly will not apply to the industry as a whole or even the industry as a part. I've read quite a few software compilation books

(including *Beautiful Code* which these editors previously produced) and all of those books were better. Way better.

It took me over six months to slog through this book because of how painful it was to read. I rarely give one star reviews, but *Making Software* earned it.

Don't bother reading this.

Yevgeniy Brikman says

An important read for everyone in software development. Although the book is not executed perfectly, it raises the level of debate in the software industry from anecdotes and opinions to hard data and research.

The second half of the book is a great collection of research results across a variety of important software topics, such as:

* Learning programming: Why is it so hard? Do better tools or visual programming help?
* TDD: Does it reduce bugs? Does it lead to better design?
* Pair programming: Does it reduce bugs? Does it increase or decrease productivity?
* Code review: Does it reduce bugs? Should you do it in groups or individually?
* Women in computer science: Why are there so few? Is it due to genetic differences or cultural biases?
* Team organization: Is Conway's Law something to avoid or embrace?

This book is now my go-to source for a variety of software decisions. I just wish I had read it long ago.

The reason for 4 stars instead of 5 is that the way this information is presented is not particularly compelling. Most of the chapters in the first half of the book, and a couple from the second half, are written in a dry, academic style that's too focused on the nitty gritty details of software research methodologies. I suppose that's OK if the target audience is other researchers, but my impression is that the goal of this book is to bring evidence-based software engineering to the typical programmer, and to do that on a large scale, you need a much more approachable writing style. In other words, if the goal of this book is to motivate change, then the authors need to pick up a copy of "Made to Stick" and learn to simplify the message (e.g. gloss over the research details), make it more concrete (e.g. explain what it means in the real world), involve some emotion (e.g. these are controversial topics, feel free to make some jokes or have an opinion every now and then), and tell stories (e.g. give examples of how these results affected an actual project).

Overall, a very worthwhile read, but if you're not a researcher, be prepared to do a lot of skimming, especially in the first part.

Some good quotes from the book:

We hope the questions and answers in this book will change how you think about software development. We also hope these essays will persuade you to say, “Citation, please,” the next time someone claims that one way of laying out braces in C or Java is better than another.

Convincing evidence motivates change.

Evidence is not proof. In general, evidence is whatever empirical data is sufficient to cause us to conclude that one account is more probably true than not, or is probably more true than another.

Qualitative research has to precede quantitative research and will look at situations that are more complicated. When only few different factors are involved (such as in physics), one can proceed to quantitative investigation quickly; when many are involved (such as in human social interactions), the transition either takes a lot longer or will involve premature simplification. Many of the credibility problems of software engineering evidence stem from such premature simplification.

We found that programmers deviated from a reference group in that they are lower on Extraversion, lower on Emotional Stability, and higher on Openness to Experience. [...] Programmers are also more homogeneous than the population as a whole; that is, programmers vary less in personality than do people in general. This confirms the stereotype of programmers being neurotic, introverted, and intellectual—and, by the way, male (which I know for a fact some people consider tantamount to a personality trait!).

It makes a significant difference whether you ask someone how much time he needs to complete a given amount of work, or whether you ask how much work he can complete in a given amount of time.

Thus a possible corollary of Conway’s Law is: A software system whose structure closely matches its organization’s communication structure works “better” (defined broadly) than a subsystem whose structure differs from its organization’s communication structure.

Every page in this book has been checked over by an editor. Why? Because even if you’re the smartest, most capable, most experienced writer, you can’t proof-read your own work. You’re too close to the concepts, and you’ve rolled the words around your head for so long you can’t put yourself in the shoes of someone who is hearing them for the first time. Writing code is no different. In fact, if it’s impossible to write prose without independent scrutiny, surely it’s also impossible to write code in isolation; code has to be correct to the minutest detail, plus it includes prose for humans as well! (You do write comments, don’t you?)

Niklas says

Mix of uninteresting or even boring articles and interesting or even thought provoking articles which highlight various aspects of software development. Sometimes it could be just more in-depth look into common topic, or a completely new angle to something. After reading the book I think have more rich and nuanced view on the field of software development. You wont find any practical tips you could immediately apply to be a better software developer.

Matt says

There are some great chapters and some weak chapters here. By "weak", I mean topics that don't interest me and so it was a chore to get through.

I would rather read this in a blog or journal format.

The community needs reading like this, but I'm not sure this is the ideal way to throw it all out there.

Thankful for it nonetheless.

John says

We need more books like this: spelling out what we actually know about making software, and making it clear what we don't know.

I'd give it 5 stars, except that as a book of essays it has a few faults: some of the essays were eye-lid-droopingly dull, there was a lot of overlap (not surprising as there is a paucity of good data out there), and some of the referencing was inconsistent.

I still think it's an essential read for anyone running a software team.

Filippo Pacifici says

I was not impressed by this book.
On one side the book is quite academic and not very applicable to concrete problems, on the other hand it still fails in producing strong evidence for most of the claims made (small samples, questionable assumptions).
Finding scientific evidence from real world projects is really hard in software, as the authors admit, this makes me think the approach of the book is wrong.

Said that, there are some interesting chapters, like the one on the Conway corollary and the one on code copy paste (I would dispute the conclusion). Still I think the book tries to find scientific evidence on topics where intuitive evidence is satisfactory, and misses really important problems like architecture and verification, where some harder evidence would be desirable.

Gary Lang says

Every engineer should be required to present engineering evidence for their beliefs. This book tries to do this for object-oriented programming, design patterns, and so on. It's a compelling example of what engineers should do every day.

Dale says

In the interest of full disclosure, I got this book from the library and only read about 20% of it. However, I do plan to purchase a copy, because the articles that I did read were dense with information, and I have every hope that the others will be just as good. This is a set of empirical studies of the practice of software engineering, bolstered with theoretical models in most cases.

Xavier Shay says

I can't really criticize an academic book for being too academic, but it did go on at times. That's the problem with reading research papers back to back, you end up reading the same threats to validity and research methods over and over again. Still, some good research in here:

- Code review great at catching bugs.
- Coverage not great at predicting bugs, churn and team organization much better.
- Pairing is generally pretty good but not for all tasks.
- Programmers spend *heaps* of time communicating.
- And more, my recall is a bit hosed because I chained all 30-something chapters in the space of two plane trips.

Tom says

An important book for people interested in what actually makes for good software (purely in engineering terms). Incredibly dry and not much fun. I read maybe 60% before deciding I'd had enough. The quality of writing varies from dreadful to acceptable.

Tuba says

Evidence-based software engineering, and the way it is explained in the book, gives you a different perspective in making judgments. Chapters written by different authors doing systematic reviews on separate topics give good examples of how to do such reviews on a subject. It is definitely worth to read.

Erika RS says

Like many collections of works, the quality in this volume varied. I only remember one of the articles being particularly bad. A worthwhile number stand out as good.

The first four articles cover how to read software engineering research. I wish that I had read it in grad school. If you find yourself reading academic papers in computer science, it's worth reading these four articles.

The rest of the articles cover research about different areas of software engineering. If you're like me, your opinion of each article will be partially influenced by its quality and partially influenced by your interest in the topic. That said, there was a general pattern that the essays that tried to very narrowly investigate whether or not some piece of common sense wisdom were supported by evidence were, simultaneously, the best research, in terms of not overreaching, and the worst reading. :-)
