At first glance, a reader looking at this book might think, "Humph! Another book on software engineering!" But this is definitely not a typical software engineering book.

Designed for use in the classroom, the author's mission is to teach newcomers to the field the fundamental facts of software engineering and point out fallacies in the thinking of both practitioners and theorists. A distillation of Glass's half century of software experience, the book offers a personal, somewhat biased, look at the field that may actually be more helpful to people who have worked in the industry for a few years, including those involved in maintenance, development, and project management.

Glass groups his fifty-five facts about software engineering into four broad categories: management, lifecycle, quality, and research. He groups fallacies into three categories: management, lifecycle, and education. Within these categories, he classifies by topic, so readers can easily focus on the facts and fallacies of particular interest to them. (Although there is no prescribed order for reading most of the facts in the book, Glass does occasionally refer to previously mentioned facts.)

In each instance, Glass states the fact or fallacy clearly and simply, explains what it means, discusses the controversy surrounding it -- drawing on various books, articles, and research papers published by credible authors -- and concludes with sources and references. Many of these facts and fallacies will be familiar to readers, and I found it gratifying that Glass articulates some of the opinions that I've formed, based on my own industry experience.

He also consciously avoids domain-specific words (jargon), although he does refer to debuggers, coverage analyzers, and the like -- and the book could use a Glossary to define these references. The book concludes by
classifying and presenting four underlying themes for the facts and fallacies: complexity, estimation, disconnect, and hype.

A Few Facts and Fallacies

Not all the facts are fully explained, however. Take this one for example:

Fact-36: Programmer-created built-in debug code, preferably optionally included in the object code based on compiler parameters, is an important supplement to testing tools.

Although the term "testing tool" might convey the impression that this is a technique testers should use before production (which is true if the testing team does white-box testing), in my experience, the built-in debug code is also useful for debugging after a system goes into production.

Some of the facts also seem counterintuitive upon first reading:

Fact-45: Better software engineering development leads to more maintenance, not less.

But then Glass goes on to clarify that more maintenance is required because end-users want to change the software so that they can use it longer -- longer than their creators might have predicted. This was true for much of the software that caused so much alarm as the year 2000 approached, for example. In some of his other facts, the author clearly reveals his bias that software maintenance is more challenging than software development.

Actually, many of Glass's "fallacies" are simply his way of packaging the personal wisdom he wants to pass on to novice programmers. E.W. Dijkstra once observed that programmers should prove the correctness of their programs before they program. But given the complexity of today's software systems, says Glass, it may not be practical to follow this advice. So in Fallacy 10, he suggests that, instead, programmers should study the programs of other proficient programmers and learn from them.

Glass also believes that a programmer has a natural attachment to his/her programs; hence:

Fallacy-3: Programming can and should be egoless.

He also thinks it's a mistake for organizations to try fitting square pegs in round holes:

Fallacy-4: Tools and techniques: one size fits all.

The set of tools and techniques you use should be adaptable for the project at hand, he goes on to explain; some tools and techniques may not be suitable at all for a particular type of project.

But he quickly adds that we already have quite enough methodologies to adapt to a variety of situations, in the form of:
Fallacy-5: Software needs more methodologies.

Just as with the facts, however, some of the fallacies are not explained fully enough.

For example, there is:

   Fallacy-6: To estimate cost and schedule, first estimate lines of code.

Based on my experience, this statement is certainly true, but Glass provides no detailed, real world examples to prove it, and the discussion in the controversy section for this fallacy does not clarify his position or present the right way to estimate cost and schedule.

I also found it irritating that Glass uses a nintumber of culture-specific words, such as "betcha," which is not listed in many popular dictionaries or at www.dictionary.com. As the book is for an international audience, it seems to me he could have avoided such words.

Conclusion

This book has significant strengths: It consolidates wisdom the author gained through decades of practical experience and covers various disciplines within software engineering; in addition, the facts and fallacies are presented in simple language and are typically easy to translate into practice.

It's a highly personal view of the software world, though; many of the references are to the author's own books and papers, and many facts and fallacies are simply platforms for Glass to express his biases.

I would not suggest this as a text for either college students or those just entering the industry. However, I would use it in software project management workshops for experienced practitioners as a way to launch discussions and review important facts and fallacies. I'd also recommend it as a useful reference for practitioners to consult during "live" projects.

-Saya Sreenivasulu
Rational Team Lead, Bangalore
IBM Software Group

For more information on the products or services discussed in this article, please click here and follow the instructions provided.

Thank you!