The practice of building software is a "new kid on the block" technology. Though it may not seem this way for those who have been in the field for most of their careers, in the overall scheme of professions, software builders are relative "newbies." In the short history of the software field, a lot of facts have been identified, and a lot of fallacies promulgated. Those facts and fallacies are what this book is about. There's a problem with those facts—and, as you might imagine, those fallacies. Many of these fundamentally important facts are learned by a software engineer, but over the short lifespan of the software field, all too many of them have been forgotten. While reading Facts and Fallacies of Software Engineering, you may experience moments of "Oh, yes, I had forgotten that," alongside some "Is that really true?" thoughts. The author of this book doesn't shy away from controversy. In fact, each of the facts and fallacies is accompanied by a discussion of whatever controversy envelops it. You may find yourself agreeing with a lot of the facts and fallacies, yet emotionally disturbed by a few of them! Whether you agree or disagree, you will learn why the author has been called "the premier curmudgeon of software practice." These facts and fallacies are fundamental to the software building field—forget or neglect them at your peril!

Most Helpful Customer Reviews

I have read a fair number of software engineering books, and this is one of the more enjoyable books that I have read. When I first heard about it, I thought the concept of a sort of summary of the state of the art sounded really interesting. Although I haven't read any of the author's previous books, I have read and enjoyed his columns in IEEE Software and Communications of the ACM, so I had high hopes about this book. And I wasn't disappointed. Facts and Fallacies of Software Engineering is divided into 55 facts and 10 fallacies. Each fact and fallacy is presented in the same way. There is a headline/slogan that summarizes it, usually one or two pages of Discussion giving more details, then a Controversy section describing what (if anything) people disagree about and finally Sources and References. The 55 Facts are divided into the following sections and sub-sections: Management (People, Tools and Techniques, Estimation, Reuse, Complexity), Life Cycle (Requirements, Design, Coding, Error Removal, Testing, Reviews and Inspections, Maintenance), Quality (Quality, Reliability, Efficiency) and Research. The 10 Fallacies are divided into Management, Life Cycle and Education. This way of organizing the material works really well, and makes the book very accessible and easy to read. It also allows you to jump around and read what interests you the most first (which is what I did, although I eventually read all of it). Many of the facts are well known (for example Fact 3 "Adding people to a late project makes it later", Fact 16 "Reuse-in-the-large remains a mostly unsolved problem" and Fact 24 "Requirement errors are the most expensive to fix during production"), but that doesn't matter. It is actually good to be reminded of these facts even if you already know them, and the author does a very good job of summarizing them. Another thing I like about the book is the Sources and Reference parts (although I think they might as well have been combined into just one Reference section). Often there are references to research papers where the original fact was presented. It is nice to know that what is presented as a fact is indeed a fact that has been validated by research, and not just the opinion of the author (although there is certainly room for opinions in a lot of places as well). There are also lots of references to other books on software engineering, and a lot of the classic books (like The Mythical Man-Month, Peopleware and Design Patterns) are referenced. So there is plenty of leads in case you want to find out more about a certain fact. Among the facts I liked the most were Fact 12, Fact 21 and Fact 26. Fact 12: "Since estimates are so faulty, there is little reason to be concerned when software projects do not meet estimated targets. But everyone is concerned anyway." This fact and the related ones...
simply state that when a project is late, it is very likely because the estimated time to complete it was unrealistic. Very true. Fact 21: "For every 25 percent increase in problem complexity, there is a 100 percent increase in complexity of the software solution." I had never seen this fact before, but it does ring true to me. And as the author writes, it explains a lot of the other facts in the book as well. Fact 26: "Explicit requirements 'explode' as implicit (design) requirements for a solution evolve". In other words, each explicit requirement leads to many more implicit requirements (by a factor of up to 50). This too I had never seen stated, but I definitely recognize this from my own experience. The Fallacies section list ten common misunderstandings in software engineering, and this is where I disagree on a couple of points. Fallacy 7 states "Random test input is a good way to optimize testing". I agree that it can not be the only test approach, but since he also writes "It may or not survive as a test approach", he is skeptical to it in general. My own experience is that it is an invaluable complement that helps flush out numerous what I call "timing dependent" bugs caused by the nature of asynchronous events. I also don't think all his arguments in Fallacy 8 are valid. I agree that since there is no data demonstrating the truth of "Given enough eyeballs, all bugs are shallow", we should not just accept it as truth. But I think he misses the point when he refers to research showing that inspections don't get much extra benefit beyond 2 to 4 participants. My interpretation is that the "enough eyeballs" are not so much inspecting the software in question as running it and debugging it when there is a problem. And the "all bugs are shallow" should not be interpreted too literally. Of course the bugs may still be difficult, but if many people look at it, the chances of someone solving it fairly quickly increases. Those two examples notwithstanding, I did find myself nodding my head and saying "yes, I agree with that" almost all of the time reading this book. There are many more interesting facts that I have not commented on, and if you are interested in software engineering I highly recommend this book.

The other reviewers have done a fine job of covering the content of the book. I will comment about its usefulness. In short, this book is truly valuable to the developer who has recently been promoted to team leader. While developers would benefit greatly from this book, the reality is that most developers would rather read books like "Effective C++", "Design Patterns", "Expert One on One Oracle", etc. To the new manager, though, this book is a gem. The book talks about specific management issues as well as the development life cycle and quality. In short, the book focuses exactly on what the team leader does and the team leader's team. In addition to the material presented in the book, the author gives a great number of sources and reference for further reading. Written in the style of "Effective More Effective ", this book presents what the author asserts are 55 facts about software engineering. While you will see the obligatory "Adding people to a late project makes it later" section, the book also introduces several 'facts' that I have never really thought much about e.g. "Enhancements represent roughly 60 percent of maintenance costs. The true gems of this book are the 'source' and 'reference' section of each fact. Their purposes are twofold. Firstly, they serve to validate the author's claim for each of these facts. Secondly, they provide readers with good follow-ups. Amazingly, many if not most of the software classic are somehow mentioned in this book. (Even the cult classic Zen and the Art of Motorcycle Maintenance!) This book manages to capture most of the essence of software engineering literature of today. Certainly, you may not agree with what the author terms as facts. The author does attempt to address these issues under 'Controversy' for each fact. If you read this book, be sure to follow up on your reading with one of the many mentioned articles/books. Otherwise, you could potentially be left with only a surface understanding of the many issues involved. Fact 56: "This reviews is written from the viewpoint of a 4 year old software developer in Singapore" Bob Glass is a programmers' programmer -- he's at the other end of the scale from software engineering gurus. This collection of fifty-five software facts and ten fallacies is distilled from his forty-five years of programming experience. Much of it is the kind of thing your grandmother could have told you -- if your grandmother was a programmer -- but some items will surprise and others annoy some people. Take the following two items as examples: "The 60/60 rule: 60 percent of software's dollar is spent on maintenance, and 60 percent of that maintenance is