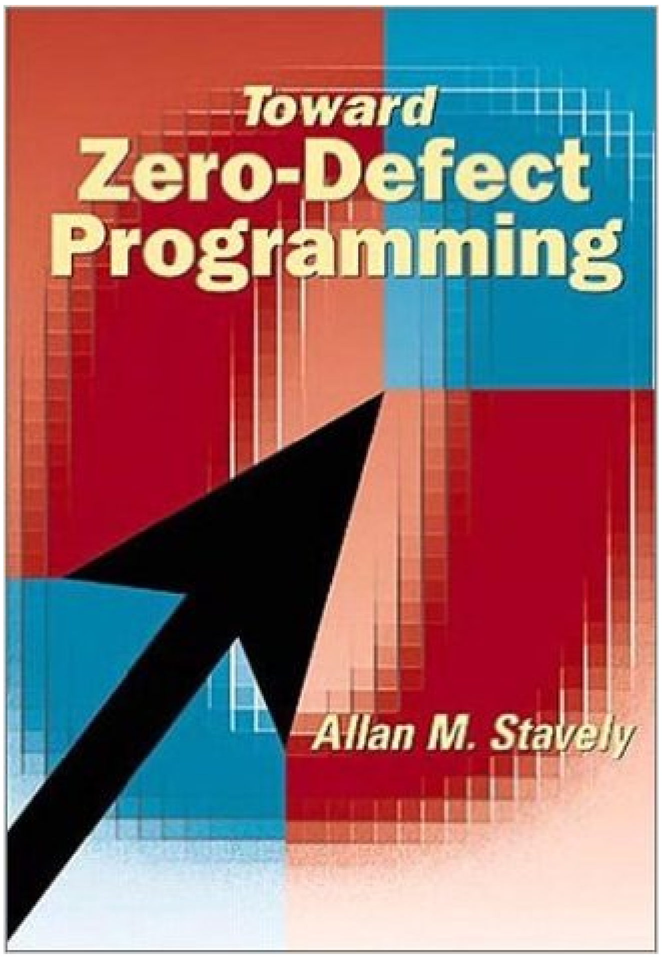
Toward Zero Defect Programming



Synopsis

"Toward Zero-Defect Programming" describes current methods for writing (nearly) bug-free programs. These methods are based on practices developed at IBM and elsewhere under the name Cleanroom Software Engineering. The successful application of these methods in commercial projects over the past fifteen years has produced defect rates that are, at least, an order of magnitude lower than industry averages. Remarkably, this reduction in defects comes at no net cost; on the contrary, it is often accompanied by increased productivity and shorter overall development time!

Book Information

Paperback: 256 pages

Publisher: Addison-Wesley Professional; 1 edition (September 24, 1998)

Language: English

ISBN-10: 0201385953

ISBN-13: 978-0201385953

Product Dimensions: 6.2 x 0.6 x 9 inches

Shipping Weight: 14.7 ounces (View shipping rates and policies)

Average Customer Review: 4.7 out of 5 starsÂ Â See all reviewsÂ (3 customer reviews)

Best Sellers Rank: #1,082,741 in Books (See Top 100 in Books)   #24 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Quality Control   #1370 inÂ Books > Textbooks > Computer Science > Software Design & Engineering   #2934 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Software Development

Customer Reviews

Dr. Allan Stavely has done for the Cleanroom Software Engineering method what Martin Fowler did for the Unified Modeling Language in his book "UML Distilled." He’s analyzed the best and most useful parts of the Cleanroom method and found a great way to present them. After reading this book, you’ll be controlling defect rates and shortening development times on your own software projects.For those not in the know, the Cleanroom method is a set of practices pioneered by the late Harlan Mills. The idea is to use some simplified mathematical formalism along with group verifications. The result? You shift time away from hack-and-slash debugging towards review. Often, the entire start-to-finish development time is shortened.With object-oriented languages and template instantiation times, this is a really good thing: the compile-debug-test cycle is far too painful and too slow to support today’s shortened deadlines.The key to Cleanroom is that the mathematical formalism is simplified and "just enough." Stavely demonstrates the typical structures found in programs and shows how intended function statements (the math part) are used in the group review (the verification part) to discover defects in the code. Later, a testing group exercises the paths through the code that users are most likely to take, giving statistical metrics on mean-time to failure and feedback into the quality of the method’s practice.Stavely’s conversational writing style makes grasping the material efficient. Each chapter focus on just one aspect of the method, and exercises at the end test how well you grasped the material. Although Stavely includes hints to the answers for selected questions, I would’ve preferred complete answers to all the questions. That’d make the book more useful outside of a classroom setting.Transcripts of review sessions show how the method succeeds in the group review. Although hypothetical, Stavely based them on actual review sessions taken by his coworkers and students over the years. They help guide newcomers to the method on how to conduct the verification step.Overall, this is a great introduction to the Cleanroom method and after finishing the book you’ll be able to introduce it to your own group in no time. Buy a copy for everyone on your team!

This book is a clear, practical introduction to the Cleanroom method. Though designed as a textbook, it is also suitable for professionals.It includes a useful bibliography, with suggestions at the end of the chapters for further reading. The final chapter sketches some areas not covered, giving references.There are some areas intentionally omitted or only sketched, though references are provided. These include: (1) using "black boxes, state boxes, and clear boxes" for top-down development. (2) introducing Cleanroom methods in an organization (3) organizing a Cleanroom team.Mathematically, the book is very easy going. For example, some methods which, technically, would require proof are not proved. Those of us who easily digest such details can readily fill in gaps, while others are probably happy to be spared.The book provided an unusually high return of useful content per unit time invested in reading it, and as such I recommend it highly.

Minimizing defects when writing software is a goal that all programmers seek. Typically, a programmer will reflect and develop strategies after the "heat of the battle." Books that provide a systematic approach to a particular aspect -- that of logical correctness -- are not that common -- particularly those directed towards programmers rather than academicians -- so any addition to the literature is welcomed.Before reading this book, I was unfamiliar with either Cleanroom Software Engineering or Harlan Mills, who is credited with conceiving the method. It was developed by IBM in the 70s, and adopted more widely in the company in the 80s. Since that time, its influence has spread beyond IBM.In my reading, the heart of the method is a logical formalism which represents the operation and flow of a program in a language-neutral fashion. This is described straightforwardly in the text, and other aspects of the software development process are described from the CSE point of view.For the programmer, this approach most basically moves the focus from "what am I doing" to "how am I doing it." Considerations of logical branching and set completeness of operations come to the fore through this analysis. This is an essential step in moving from a coder to a programmer. In a team setting, this provides a "style neutral" approach to identifying the logical structure of each member's program contribution.I feel somewhat about this formalism they way I felt about flow charts in college. The concept is good. The visual provided by flow charts is helpful as a training approach. But, the formalism simply replicates the logic of the application.At the same time, well-constructed code with well-chosen function and variable names can pretty much emulate the symbolic representation of CSE. So, I'd go a step further and suggest that one simply model one's coding style on the formalism. That avoids the double work and duplication of the program logic.

Download to continue reading...

Java: The Ultimate Guide to Learn Java and Python Programming (Programming, Java, Database, Java for dummies, coding books, java programming) (HTML, ... Developers, Coding, CSS, PHP) (Volume 3) Programming #8:C Programming Success in a Day & Android Programming in a Day! PowerShell: For Beginners! Master The PowerShell Command Line In 24 Hours (Python Programming, Javascript, Computer Programming, C++, SQL, Computer Hacking, Programming) Excel VBA Programming: Learn Excel VBA Programming FAST and EASY! (Programming is Easy) (Volume 9) Python: Python Programming For Beginners - The Comprehensive Guide To Python Programming: Computer Programming, Computer Language, Computer Science (Machine Language)