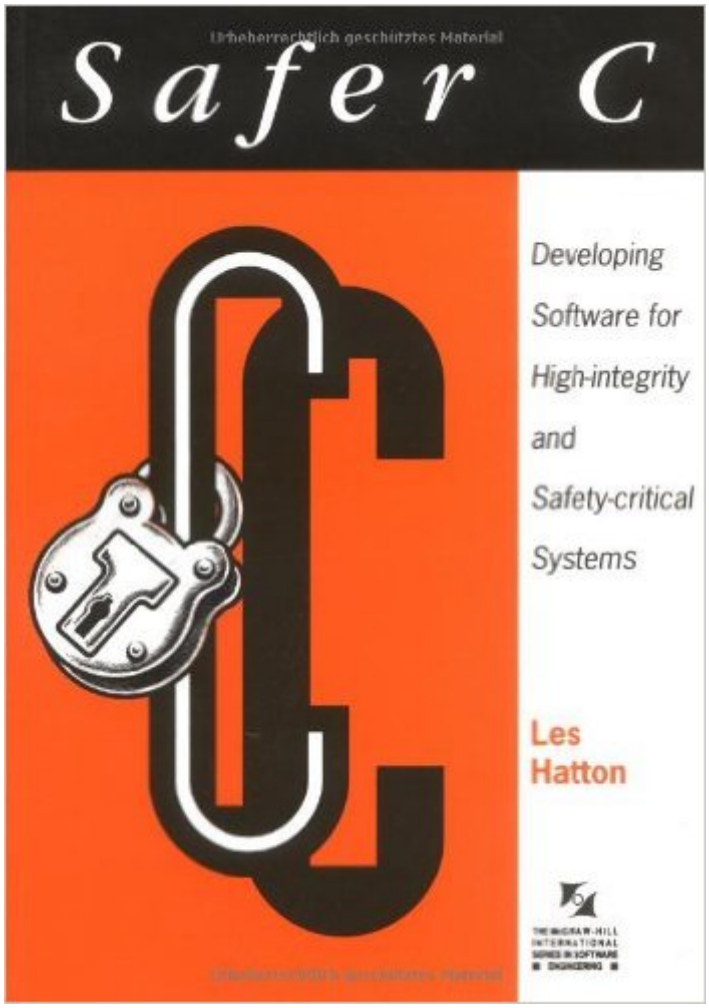
Safer C (McGraw-Hill International Series In Software Engineering)



Synopsis

This book is aimed at C developers in safety-related or high-integrity environments.

Book Information

Series: McGraw-Hill International Series in Software Engineering

Paperback: 228 pages

Publisher: McGraw-Hill Companies (1995)

Language: English

ISBN-10: 0077076400

ISBN-13: 978-0077076405

Product Dimensions: 9.7 x 6.8 x 0.6 inches

Shipping Weight: 1 pounds

Average Customer Review: 3.6 out of 5 starsÂ Â See all reviewsÂ (5 customer reviews)

Best Sellers Rank: #2,508,937 in Books (See Top 100 in Books) #57 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Quality Control #750 inÂ Books > Computers & Technology > Programming > Languages & Tools > C & C++ > C #7083 inÂ Books > Textbooks > Computer Science > Programming Languages

Customer Reviews

Well when first introduced to this book the title evoked some interest in me, my development background being in C/C++. And though my first impressions remained hopeful, it soon became clear that this book has a very defined audience indeed.I must admit I find much of what Les Hatton says about the C language and its application in high integrity very interesting but perhaps its relevancy today isn’t what it once was. Don’t get me wrong, there is some good information in this book and you may even have an epiphany about the C language while you read it. But Les’s lack of lucidity makes the book very dry to read, and unless you are an avid fan of standards and rigid processes or can find some humor in what he says, you might find getting through any of the chapters somewhat of a challenge.Moreover the reader is constantly and implicitly reminded while reading this book of the author’s staunch support for the C language. And though this takes many different forms throughout the book, it became clear to me at least, that here may be a case where someone has gone through a great deal of work in the defense of a programming language instead of propounding the use of a more appropriate one. That is not to say there may not be situations where the C language is necessary in mission critical or high integrity software; I just don’t think the author has given ample justification in these cases for considering other more strongly typed

languages than C for them.However, there are some sections and chapters of the book that are worth some note. I found the following of interest because of it potential applicability to what all software engineers need to be aware of, particularly in Tivoli.Chapter 2. This entire chapter is useful to those individuals who really want to teflon coat their C code. The author skillfully wades through all the potentially caveats and misbehaviors that can haunt someone writing in this language. Along with his obvious vast experience with standards it becomes painfully clear that he has fallen victim to some of them.Sections 5.1.2 and 5.3. The former section has a good synopsis on test coverage and for those who do any development or testing with C may find what he has to say here of some use. The latter section contains some good information on automation and tools, and though you will definitely find yourself skipping through it (especially the parts on standards and ISO), much of what he has to say has some value to what we do in Tivoli.This book does not belong on everyone's shelves and is clearly not meant for the novice or uninitiated programmer or engineer. It is useful however in those handful of cases where one absolutely needs to understand what the limits of C are, and then to take the necessary design, development or verification action to meet those potential challenges. I find it a useful though a decidedly optional appendix to any Kernighan and Ritchie...

This book was recommended to me by a collegue. Looking at the title, I don't think I would have picked it up myself. My interests tend towards the practical. But this was a surprising delight to read. Hatton is sharp, well-informed and very funny. And I was surprised at how similar the challenges that Hatton addresses are to the ones faced by the development groups I've worked with.Hatton's primary motivation is to defend C as a language for use in safety critical systems. And that's an argument that I don't have much interest in. But he defends it by identifying unreliable features and usages of C and showing how these can be automatically detected and avoided. This is the interesting part. His analysis is deep and entertaining. He obviously has spent many many hours reading draft standards and commentary. The task has nearly turned his brain to jelly. Yet he pressed on and has identified and categorized long lists of issues that the standards leave open to interpretation.The second basis for his argument are empirical studies he has made of occurances of faults in software. These are also very interesting. He also has made studies correllating complexity metrics to these fault metrics and thereby comes to well-defined complexity criteria. But having defined some metrics, he also has good ideas about how to make use of them.Hatton doesn't have much faith in process standards (such as the CMM, or ISO 9000). He feels that if you have to have rules, they need to have automated enforcement mechanisms. Indeed, one of the

purposes of this book is to plug tools that do just this that he has developed. But he also describes competing and complementing tools with fairness. Indeed, his descriptions of the different static and dynamic code analysis tools is the best one I've seen. I learned several useful things from this section alone. He also realizes that you can't have arbitrary limits. So he often suggests that a grey zone be defined where formal written justifications can allow code to exceed the lower limits. It is clear to me that if we are going to continue to call ourselves software engineers, we will need to be spending more time defining, following and justifying the engineering practices we use. Hatton's analysis is a useful model, because it shows how this can be done without hopping on to one of the latest management buzzword models.His background as a geologist shapes his expectations for computer languages and software. He doesn't have much sympathy for ambiguous specifications and standards. He expects a lot, but he also knows that we have the knowledge and ability to meet these expectations -- expectations that anyone else who wants to be called an engineer accepts.

This book teach me some weakness of C language, like undefined, unspecified and implementation defined.It was base of MISRA-C. Good job.After this book, C language has more strong type and definitions. Thank you.

In the early 1990s, Modula-2 and Ada were declining in general popularity and developers of safety-critical software were faced with converting to C (and later C++) to maintain access to a larger labor pool. Hatton wrote this book as an argument that C is actually a good language for safety-critical work. His main idea is that while the Modula-2 language (for example) is designed with fewer gotchas than C, the tooling available for C makes it a better choice.The greatest part of his advice on writing "safer C" is to avoid all use of implementation-dependent behavior. He lists the many unspecified, undefined, and implementation-defined behaviors in C. No one can keep all of these in mind, so he the emphasizes the need for automated code review tools.If you have a moderate background in C and appreciation of good engineering practice, I don't think you get will much out of this book, but I think it would give a beginner a decent introduction to the issues of critical software development. Regardless of your level of experience, it is worth going to Hatton's web site and reading his articles on MISRA C. He is very critical of MISRA C, saying that application of the rules gives far too many false positives. Based on his analysis of observed faults in commercial code, he advocates using a list of 20 coding rules (versus 141 rules in MISRA C 2004).

Just skimmed it. Appears to be well-written and contains a LOT more detailed explanations than I

expected. Don’t know how I missed this book in my many years of C involvement.

Safer C (McGraw-Hill International Series in Software Engineering) Non-Functional Requirements in Software Engineering (International Series in Software Engineering) Software Process Design: Out of the Tar Pit (Mcgraw-Hill International Software Quality Assurance) Principles of Corporate Finance (The Mcgraw-Hill/Irwin Series in Finance, Insurance, and Real Estate) (McGraw-Hill/Irwin Series in Finance, Insurance and Real Estate (Hardcover)) Introduction to Chemical Engineering Thermodynamics (The Mcgraw-Hill Chemical Engineering Series) ADA as a Second Language (McGraw-Hill series in software engineering and technology) McGraw-Hill’s National Electrical Code 2014 Handbook, 28th Edition (McGraw Hill’s National Electrical Code Handbook) McGraw-Hill Education 5 TEAS Practice Tests, 2nd Edition (Mcgraw Hill’s 5 Teas Practice Tests) McGraw-Hill Education: Top 50 ACT English, Reading, and Science Skills for a Top Score, Second Edition (Mcgraw-Hill Education Top 50 Skills for a Top Score) McGraw-Hill Education SAT 2017 Edition (Mcgraw Hill’s Sat) McGraw-Hill’s SAT Subject Test: Biology E/M, 2/E (McGraw-Hill’s SAT Biology E/M) McGraw-Hill Education LSAT 2016 (McGraw-Hill’s LSAT) McGraw-Hill Education LSAT 2017 (McGraw-Hill’s LSAT) McGraw-Hill Education 500 MAT Questions to Know by Test Day (McGraw-Hill’s 500 Questions) McGraw-Hill’s ASVAB Basic Training for the AFQT, Second Edition (McGraw-Hill’s ASVAB Basic Training for the Afqt (Armed Forces) McGraw-Hill Education 500 Financial Accounting and Reporting Questions for the CPA Exam (McGraw-Hill’s 500 Questions) McGraw-Hill Education 500 Business Environment and Concepts Questions for the CPA Exam (McGraw-Hill’s 500 Questions) McGraw-Hill Education 500 Regulation Questions for the CPA Exam (McGraw-Hill’s 500 Questions) McGraw-Hill’s Praxis I and II, Third Edition (McGraw-Hill’s Praxis I & II) McGraw-Hill Education: Top 50 ACT Math Skills for a Top Score, Second Edition (Mcgraw-Hill Education Top 50 Skills for a Top Score)