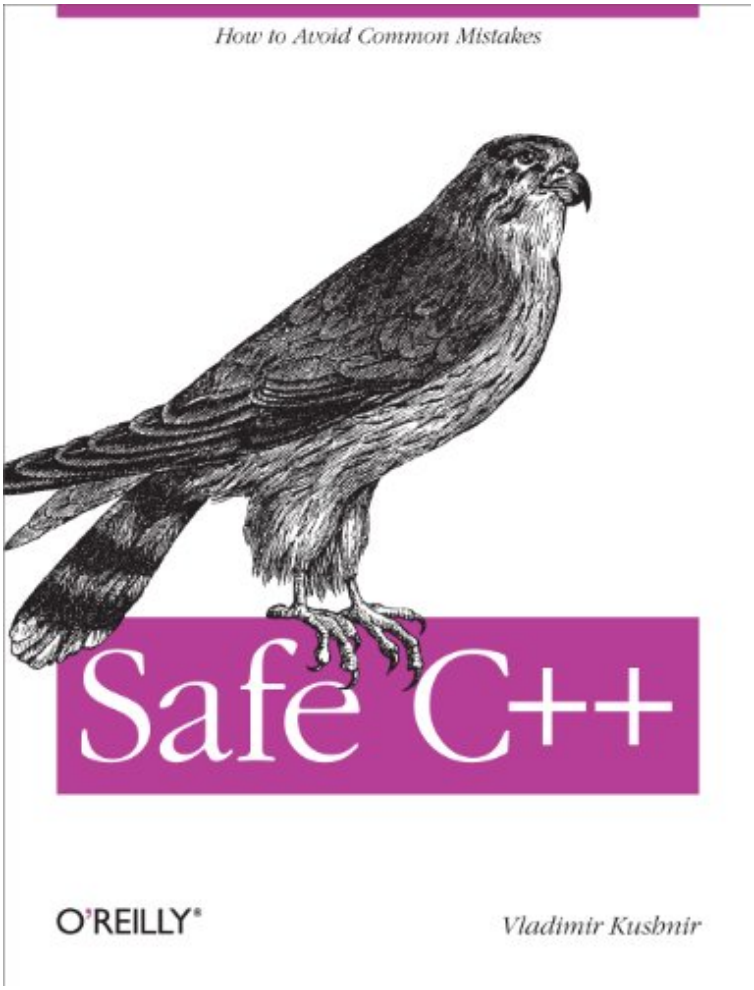
Safe C++: How to avoid common mistakes

Von Vladimir Kushnir
*audiobook | *ebooks | Download PDF | ePub | DOC*



Produktinformation -Verkaufsrang: #754152 in eBooksVerffentlicht am: 2012-05-30Erscheinungsdatum: 2012-05-31File Name: B00885RVNS | File size: 36.Mb

Von Vladimir Kushnir : Safe C++: How to avoid common mistakes before purchasing it in order to gage whether or not it would be worth my time, and all praised Safe C++: How to avoid common mistakes:

KundenrezensionenHilfreichste Kundenrezensionen0 von 0 Kunden fanden die folgende Rezension hilfreich. I have mixed feelings here Von mkoSafe C++ is quite interesting book, however, after reading it I have mixed feelings. At some point it is targeted at begineers. If you dont know how to avoid memory related mistakes, how to recover from the run-time errors, or how to hunt bugs you dont know much about safe coding yet. In that case this book is definitelly for you. On the other hand, it requires quite good knowledge in C++ related area. I think, that knowing C++ well triggers some knowledge regarding topics covered in the book as well. Anyway, there are few nice topics there, and let me discuss them briefly.Book itself is divided into three parts. First describing what can be the source of issue, second providing overview for some basic strategies that can save your time and effort, and third telling you how to prepare the code for delivery. I think that last part is there just for the purpose of being there. It is short, condense and

doesnt teach you how to make things right. First part tells some obvious things regarding issues you can encounter during development. If you are real begginer this will be a good source of the information for you. If you are experienced developer, feel free to skip it. Most interesting part of the book can be found in second part. Here you will find the essence of the book. You will go over the sections covering different types of issues and you will be told how to overcome them. Each section contains brief summary that makes it much easier to learn things.One thing that stroke me during the lecture was the strong assumption of the author that you will write self estime code from the scratch. Which is not quite a use-case in real life. Of course, following the rules proposed by Vladimir will definitelly help you develop better and safer applications, but sometimes this is not possible to develop in isolation. You will depend on external libraries, you will have to access external data sources, you will deal with the inherited code nobody remembers already. In these cases simple rules are not enough. What I would like to find in this book are the tools and methods for tracking the issues, memory leaks, code analysis etc. Let me explain by few examples. For example, Vladimir suggests using dedicated scpp::vector class instead of std::vector. This sounds fine, but he makes very strong assumption that you will never cast to std::vector even though you inherit from it. This is very strong assumption taking into account that most of the people use std::vector and you will have to cast sooner or later. Another example is to use smart pointers to track the allocations. But what about things allocated inside some legacy code? How to track memory leaks there? The same reffers to the section covering the code being easier to debug. The code might be better for debugging but definitelly not easier to read for developer. I know that you always have to make a tradeoff, but still, I think there are better ways to make debugger friendly code comparing to #ifdef DEBUG based inner fields of classes.Overlally, I think this book will be very interesting for beginners who have already learned C++ but they lack the real life experience. When it comes to experienced developers, I am pretty sure you know most of the solutions presented in the book already. Otherwise you wouldnt be able to survive in the industry.0 von 0 Kunden fanden die folgende Rezension hilfreich. I have mixed feelings here Von mkoSafe C++ is quite interesting book, however, after reading it I have mixed feelings. At some point it is targeted at begineers. If you dont know how to avoid memory related mistakes, how to recover from the run-time errors, or how to hunt bugs you dont know much about safe coding yet. In that case this book is definitelly for you. On the other hand, it requires quite good knowledge in C++ related area. I think, that knowing C++ well triggers some knowledge regarding topics covered in the book as well. Anyway, there are few nice topics there, and let me discuss them briefly.Book itself is divided into three parts. First describing what can be the source of issue, second providing overview for some basic strategies that can save your time and effort, and third telling you how to prepare the code for delivery. I think that last part is there just for the purpose of being there. It is short, condense and doesnt teach you how to make things right. First part tells some obvious things regarding issues you can encounter during development. If you are real begginer this will be a good source of the information for you. If you are experienced developer, feel free to skip it. Most interesting part of the book can be found in second part. Here you will find the essence of the book. You will go over the sections covering different types of issues and you will be told how to overcome them. Each section contains brief summary that makes it much easier to learn things.One thing that stroke me during the lecture was the strong assumption of the author that you will write self estime code from the scratch. Which is not quite a use-case in real life. Of course, following the rules proposed by Vladimir will definitelly help you develop better and safer applications, but sometimes this is not possible to develop in isolation. You will depend on external libraries, you will have to access external data sources, you will deal with the inherited code nobody remembers already. In these cases simple rules are not enough. What I would like to find in this book are the tools and methods for tracking the issues, memory leaks, code analysis etc. Let me explain by few examples. For example, Vladimir suggests using dedicated scpp::vector class instead of std::vector. This sounds fine, but he makes very strong assumption that you will never cast to std::vector even though you inherit from it. This is very strong assumption taking into account that most of the people use std::vector and you will have to cast sooner or later. Another example is to use smart pointers to track the allocations. But what about things allocated inside some legacy code? How to track memory leaks there? The same reffers to the section covering the code being easier to debug. The code might be better for debugging but definitelly not easier to read for developer. I know that you always have to make a tradeoff, but still, I think there are better ways to make debugger friendly code comparing to #ifdef DEBUG based inner fields of classes.Overlally, I think this book will be very interesting for beginners who have already learned C++ but they lack the real life experience. When it comes to experienced developers, I am pretty sure you know most of the solutions presented in the book already. Otherwise you wouldnt be able to survive in the industry.

KurzbeschreibungIts easy to make lots of programming mistakes in C++in fact, any program over a few hundred lines is likely to contain bugs. With this book, youll learn about many common coding errors that C++ programmers produce, along with rules and strategies you can use to avoid them.Author Vladimir Kushnir shows you how to use his Safe C++ library, based in part on programming practices developed by the C++ community. Youll not only find

recipes for identifying errors during your programs compilation, runtime, and testing phases, youll learn a comprehensive approach for making your C++ code safe and bug-free.Get recipes for handling ten different error types, including memory leaks and uninitialized variablesDiscover problems C++ inherited from C, like pointer arithmeticInsert temporary and permanent sanity checks to catch errors at runtimeApply bug prevention techniques, such as using separate classes for each data typePursue a testing strategy to hunt and fix one bug at a timebefore your code goes into production