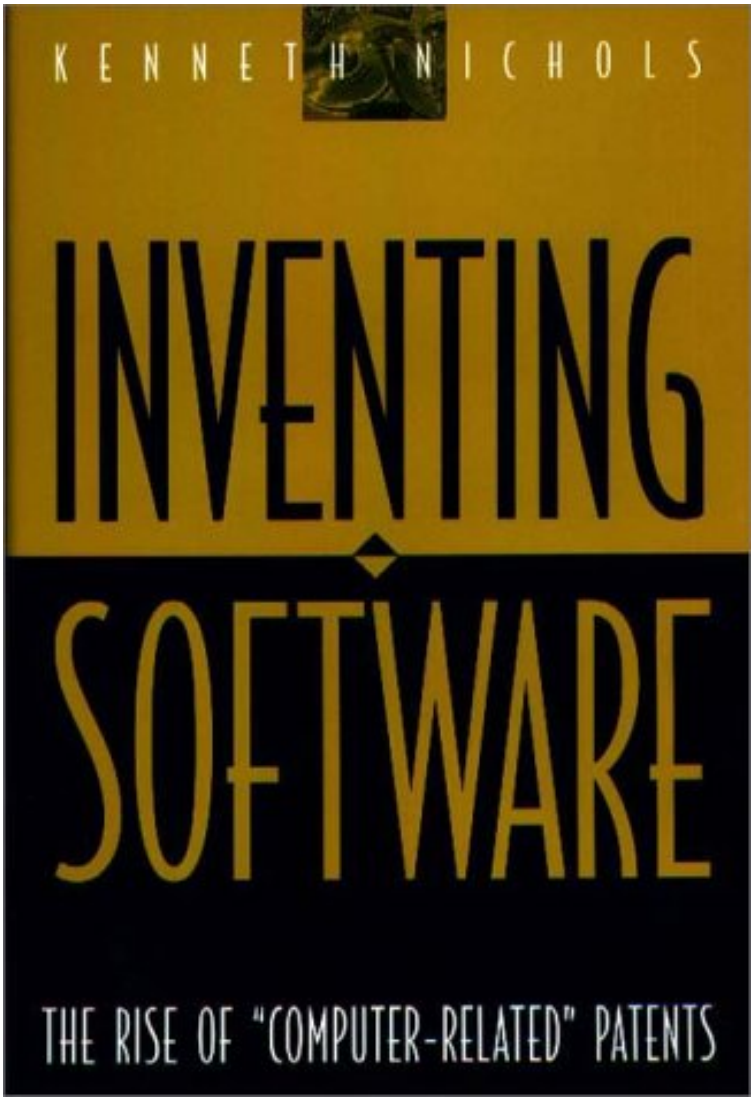
Inventing Software: The Rise of Computer-Related Patents

Kenneth Nichols
*DOC | *audiobook | ebooks | Download PDF | ePub*



#4504712 in Books Praeger 1998-04-16Original language:EnglishPDF # 1 9.21 x .44 x 6.14l, 1.10 #File Name: 1567201407184 pages | File size: 28.Mb

Kenneth Nichols : Inventing Software: The Rise of Computer-Related Patents before purchasing it in order to gage whether or not it would be worth my time, and all praised Inventing Software: The Rise of Computer-Related Patents:

2 of 2 people found the following review helpful. Good attempt at a discourse on a difficult subjectBy Erik GfesserFrom the first lines of the preface of this work by Kenneth Nichols, it is apparent that what was initially planned to be of more concise scope grew into a discussion of all things associated with software patents, and therein lies the rub. It is interesting that the author notes in just the second sentence that he "came to the conclusion that the

particulars of the debate [within the programming community over the desirability and ultimate effect of software patents]...are not that interesting or enlightening". Really? I find it odd that an individual with a J.D. and an M.S. in Computer Science would make such a statement. Of course, his follow-up statement that the "larger and more important story" involves the fact that "software development is a new kind of creative activity, one that defies the neat and mutually exclusive categorizations of intellectual effort as either artistic or scientific" might be true, Frederick P. Brooks wrote extensively on this topic in his classic tome on software engineering long before the USPTO permitted the filing of related patents. Fortunately, Nichols dedicates most of his work to the more boring subject matter, although unfortunately the explanations provided are a bit boring themselves. Putting aside the fact that this book was written in the late 1990s, however, there is a lot to offer here to the professional software engineer as well as the non-technical management community associated with software interests of this nature. For example, the author's concise 20-page introduction introduces well much of the background subject matter. Nichols also discusses well the topics of algorithms, programming, computer science, software engineering, and how patents fit in with this universe. The heart of the text, the third chapter which presents various software patent examples, however, is very poorly constructed in my opinion, although the explanation in its first few pages on major terms such as "novelty" and "nonobviousness" is concisely written. The diagram provided that depicts concepts related to nonobviousness, written in the style of a Venn diagram, is especially well suited for the discussion. The fourth chapter covers the software patent controversy, and the reader is well rewarded in my opinion when reaching this point in the book after 50 pages of rambling. It is strange that the seventh chapter is dedicated to a definition of programming, a discussion best suited for the introductory pages of the book, although some of the quotes provided in these pages are quite entertaining, such as the W. Saba quote from an IEEE periodical that "hardware engineers have gone further than their software counterparts because hardware design became a science...Unlike hardware engineers, software engineers still deal in magic and witchcraft". Interestingly enough, it was just recently that The Economist discussed the emergence of computer science as the foundational branch of science. Last of all, while the sixth chapter boldly begins by proclaiming "we've finally come to the good part" because it deals with recommendations for software developers, only 6 pages are dedicated to the subject! If Nichols is to write further on software patents, I recommend an entire book dedicated to recommendations.10 of 11 people found the following review helpful. Every software developer should read thisBy Lynellen PerryKenneth Nichols is both a computer scientist and a lawyer. This gives him a fantastic grounding from which to write about software patents.Why should YOU read this book? Just as a literate programmer reads journal articles, design documents, and source code, so should he or she be able to evaluate software patents. This ability is fast becoming a competitive necessity, for economic as well as legal reasons. (page 55)What's in this book?Nichols first explains what the problem is. Why are software patents a problem? How can one enforce a software patent? In what ways does software not fit into the regular patent process? What exactly is a patent anyway?Software patents are currently defined in terms of algorithms. But what exactly does the court system understand to be an algorithm, and how is that different from the viewpoint of a computer scientist? Answering this question is the subject of chapter two. Along the way, Nichols shows the problems with trying to patent (under the current system) several different programming paradigms such as self-modifying programs and distributed computing, imperative mode versus object-oriented mode versus functional and declarative paradigms. He also shows how software engineering techniques can help document the pieces needed in writing a software patent specification.Chapter three defines some major terminology of the patent process and shows how it relates to software. For example, there is a distinction between 'novel' and 'nonobvious'. Also, a software patent must carefully delineate the 'scope' and 'claims' of the patent. To help make this book as practical as possible, this chapter examines (in depth), four actual software patents: a text-searching system; an object-oriented database; a 'C' source code blocker; and a special-purpose sorting method.The next chapter takes on the debate of whether or not software patents are a good idea, and examines each side in detail including the practical concerns of enforcement and detection of infringement. After looking at this debate, Nichols presents the SDKR (named after the lastname initials of its authors: samuelson, Davis, Kapor, and Reichman). The SDKR is a proposal which ``advocates a special form of intellectual property for computer software" to replace software patents by doing several things: merge software copyright and software patent; focus on mass market software only; preserve the software market and still foster software innovation; create a clear set of legal rules specifically for software; provide legal protection on a shorter length than current patents, acknowledging that the software lifecycle is very different from other patentable inventions; provide protection to a program's behavior instead of just the exact implementation that produces a behavior; and protect ``innovation" of software instead of the stricter ``inventiveness" required by current patent law.Chapter six is chock full of practical advice for software developers who want to protect their software. The next two chapters are a summary and a look at the future of software programming. I highly recommend this book to anyone who plans to develop software as a livelihood... software patents can not only protect your investment of time, energy, and creativity in the programming process, but they can even earn you some extra money if others are forced to license your software from you in order not to violate your patent.2 of 3 people found the following review helpful. Practical explanations about how the law applies to softwareBy Janet WilsonI liked this book because of its practitioner focus. It really explains the law and

how it does (and doesn't!) apply to software. Kenneth Nichols also does a good job of "putting things in a universe," helping the reader to understand how the application of patent and copyright law to software has become so muddled.I especially liked the chapter on "Software Patent Examples" where he discusses the vagaries of applying the law to various typical software products.The book is also pleasingly short (169 pp.)--fits well in a briefcase as an airline read.So many books on patent law are written by lawyers who aren't practicing software developers. As a developer and lawyer, Kenneth Nichols is uniquely positioned to comment on this subject. I see that Quorum Books, which I gather is a quality publisher of law titles, agreed.

Since the introduction of personal computers, software has emerged as a driving force in the global economy and a major industry in its own right. During this time, the U.S. government has reversed its prior policy against software patents and is now issuing thousands of such patents each year, provoking heated controversy among programmers, lawyers, scholars, and software companies. This book is the first to step outside of the highly-polarized debate and examine the current state of the law, its suitability to the realities of software development, and its implications for day-to-day software development.Written by a former lawyer and working software developer, Inventing Software provides a comprehensive overview of software patents, from the lofty perspectives of legal history and computing theory to the technical details and issues of actual patents. People interested in the legal aspect of software patents will find detailed technical analysis of actual patented software, the legal strategies behind the wording of the patents, and an analysis of the ease or difficulty of detecting infringements. Software developers will find ways to integrate patent planning into their standard software engineering practices, and a practical guide for studying and appraising their competitors' patents and safeguarding the value of their own. Intended primarily for programmers and software industry executives and managers, Inventing Software will also be useful, illuminating reading for attorneys and software company investors.

"Nichols is an Oracle database consultant, with degrees in computer science and law, so it is not surprising that he was able to write a good mix of law and software concepts. As such, I can strongly recommend the book for both lawyers and software engineers....Nichol's book is well worth reading if your work will be affected by software patents."-American Inventors"This book will make software inventors aware of what constitutes infringement and thus help them avoid such pitfalls when inventing new software....Overall I would recommend this book."-Bimonthly of Law Books?This book will make software inventors aware of what constitutes infringement and thus help them avoid such pitfalls when inventing new software....Overall I would recommend this book.?-Bimonthly of Law Books?Nichols is an Oracle database consultant, with degrees in computer science and law, so it is not surprising that he was able to write a good mix of law and software concepts. As such, I can strongly recommend the book for both lawyers and software engineers....Nichol's book is well worth reading if your work will be affected by software patents.?-American InventorsAbout the AuthorKENNETH NICHOLS is an independent Oracle database consultant in Great Britain. He holds an M.S. in Computer Science from California State University and a J.D. from UCLA, and has had more than 15 years experience in the computer industry.