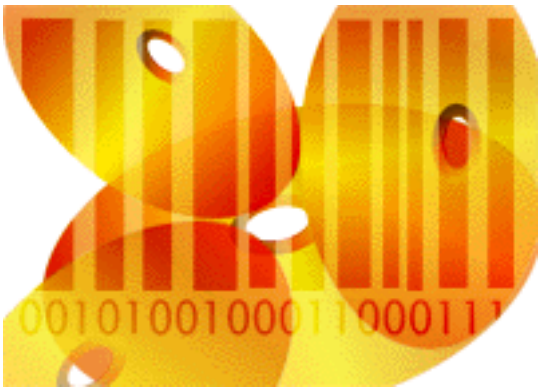
Book Reviews



Inventing Software

Author: Kenneth Nichols
published by Quorum Books, 1998
ISBN: 1-56720-140-7

Review by: Lynellen D. S. Perry

Kenneth Nichols is both a computer scientist and a lawyer. This gives him a fantastic background from which to write about software patents. The author offers the following motivation for reading 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 explains the patent process, why software patents are a problem, and how software patents can be enforced. Software patents are currently defined in terms of algorithms. Nichols discusses how the court system defines algorithms and how that definition differs from one that might be provided by a computer scientist. Coding paradigms like distributed computing, self-modifying programs, and imperative, object-oriented, functional, and declarative paradigms present problems for the software patent process. Nichols examines the details of these situations and shows the role of software engineering techniques in documenting the information needed to write a software patent specification.

Important terminology of the patent process as it relates to software is covered in chapter three. 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 provides an in-depth analysis of four actual software patents for a text-searching system, an object-oriented database, a C source code blocker, and a special-purpose sorting method.

Chapter four tackles the debate of whether or not software patents are a good idea, and examines each side in detail. The practical concerns of enforcement and detection of infringement are among the key points covered. After looking at this debate, Nichols presents the SDKR (named after the last name

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:

- merging software copyright and software patent law
- focusing only on mass market software
- preserving the software market while still fostering software innovation
- creating a clear set of legal rules specifically for software
- providing legal protection of a shorter duration than current patents, acknowledging that the software lifecycle is very different from other patentable inventions
- providing protection for a program's *behavior* instead of just the exact implementation that produces a behavior
- protecting ``innovation'' in software instead of the stricter ``inventiveness'' required by current patent law

Comprehensive practical advice for software developers seeking information about protecting their software is provided in chapter six. The last two chapters provide a summary and some thoughts on 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 decide to license your software.