Lean Software Development: An Agile Toolkit

Mary Poppendieck and Tom Poppendieck

Addison-Wesley, 2003, \$39.99, ISBN: 0321150783

This wonderful, short book is a pragmatic guide to the realities of managing software projects, and to the pitfalls of guiding research and development projects in general. With brief examples based on the authors' real-life experiences, the reader sees how "good software practice" in its traditional and literal sense can often lead to

something between disaster and frustration. In particular, the authors argue—based on experience with developing lean manufacturing systems—that specifying a software system in its entirety before developing any code ensures there is no flexibility for changes down the road when more is known about the problem. They draw the distinction between creative parts of the development process, during which variability is valued, and later stages, which attempt to stamp out any variances. The latter regime does not really apply to most software development, virtually all of which is a creative endeavor, so mismatches arise when manufacturing-style practices and metrics are applied to software development.

The authors discuss the origins of Toyota's manufacturing systems. When Toyota started making cars, the Japanese market was small, and capital was scarce. Therefore, traditional mass production was not an option for Toyota. Instead, it built cars as closely as possible to the time the car would be sold. To do this, Toyota did many counter-intuitive things, particularly delaying decisions until the "last responsible moment." The software implementation of this concept is to have enough of an idea of customer requirements to develop a prototype on a short schedule. Customers then use this prototype, determine changes, and interact with developers on a short turnaround cycle to be certain that the most important needs are implemented first, and that the features being implemented are actually needed.

Structured as a series of tools, or principles, this book takes concepts from lean manufacturing and moves them to the software domain. Each chapter ends with a section of good, pragmatic exercises to try out with a class or

software team, and includes some suggestions about how to think about systems being developed as a whole, rather than as a series of unconnected parts.

This book has quite a bit of advice that runs against traditional management and development wisdom. It also has detailed, extremely implementable suggestions about how to write contracts without becoming trapped in overly restrictive and ultimately counterproductive business relationships.—*Joan Horvath*

Reprinted from *Computing Reviews*, © 2003 ACM, <http://www.reviews.com>.

Design Methods for Reactive Systems

R. J. Wieringa

Morgan Kaufmann, 2002, \$64.95, ISBN: 1558607552

The application of well-known specification techniques to reactive systems is the focus of this book. A reactive system is "a system that, when switched on, is able to create desirable effects in its environment by responding to events." This perspective is not restricted to classical control systems (called directive systems in the book), but is also

useful in a broader spectrum of information systems. In fact, this book's approach fits event-driven programming environments perfectly. Such environments are commonly used to develop modern graphical user interfaces.

While traditional transformation systems manifest a predetermined stateless behavior, and analyses of these are focused on how inputs are deterministically transformed into outputs, reactive systems are state-based and exhibit a dynamic stimulus-response behavior. One of the recurring themes in the book, which is highlighted by the system engineering argument, is the importance of the environment. The argument states that the assumptions about the environment, plus the design specification, determine the properties of the resulting system, once the system under development (SUD) is introduced.

Wieringa focuses on specification techniques, describing their notations, explaining their semantic details, and offering some guidelines for their correct use. These techniques are arranged into four groups: function notations (useful for reaching an agreement with the customer), entity notations (describing the subject domain),

behavior notations (focusing on the transformation of stimuli into responses), and communication notations (focusing on the information exchange between systems).

After thoroughly surveying these four types of notations, Wieringa discusses the requirements-level system architectures, similar in spirit to the Object Management Group's model-driven architecture approach.

In the final part of the book, three different specification methodologies are described to illustrate how they combine different notations. Those methodologies are postmodern structured analysis, Harel and colleagues' Statemate, and a subset of the unified modeling language (UML).

In the last chapter of this book, Wieringa presents "not yet another method" (NYAM), "the gist of existing methods," where he acknowledges that there is no silver bullet and no single recipe for all specification problems. In fact, not all notations need to be used for all systems. The proper choice must depend on the SUD functionality

and the notation's intended use.

In short, the author does his best to cover a wide range of specification techniques and offers valuable insight into their underpinnings, complementing his discussions with case studies, exercises, and an uncommonly well-written glossary.

The major shortcoming I found in this book was in its deviation from terminology and notation standards. For example, the author dismisses common terms, such as use cases or nonfunctional requirements, because of their possible connotations. In addition, Wieringa introduces his own share of new notations and variants, but we do not need more new models and notations—we need better tools and more practitioners to effectively apply known techniques. If you want to fully understand the details of, and parallelisms between, specification notations, however, Wieringa is one of the best places to start.—*Fernando Berzal* Q

Reprinted from *Computing Reviews*, © 2003 ACM, <http://www.reviews.com>.