

## Software Engineering for the Rest of Us

**Diomidis Spinellis**

*Code Craft: The Practice of Writing Excellent Code* by Pete Goodliffe, No Starch Press, 2006, ISBN 978-1-59327-119-0, 624 pp., US\$44.95.

Most software engineering books are written as textbooks. Geared at hapless students who typically don't have a say in the course content, they often stray toward dry lists of term definitions and methodologies. *Code Craft: The Practice of Writing Excellent Code* claims to be "a survival guide for the software factory." Indeed it is, but it's also a brilliant software engineering book.

Software engineering isn't an abstract academic topic. It's the body of knowledge that every developer should be familiar with and apply day in and day out. Author Pete Goodliffe starts his work where the rubber meets the road—with code. He presents defensive programming, presentation and naming, commenting, and error handling. He then moves on to what he calls the code's "secret life": tools, testing, debugging, build management, performance, and security. Only then does he discuss factors affecting the code's shape, such as design, architecture, growth, and maintenance. The last three parts of the book cover less tangible but no less practical topics: programmer characteristics, teamwork, source control, specifications, code reviews, estimation techniques, methodologies, and specialized programming disciplines.

*Code Craft* is an ideal introduction to software engineering for both students and budding practitioners. By emphasizing coding—the activity that, at the start of our careers, monopolizes our time and interest—it imparts practical hands-on advice that we can immediately apply the next time we're at a keyboard.

More advanced topics follow naturally as issues that we inevitably stumble upon as we advance through our careers.

Academics looking for detailed references for each topic discussed won't find them here. On the other hand, readers are guaranteed to understand every paragraph, and they'll also get almost 100 pages answering the end-of-chapter exercises. If a fresh developer asks you to recommend one book, *Code Craft* is a perfect candidate.

**Diomidis Spinellis** is an associate professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of *Code Quality: The Open Source Perspective* (Addison-Wesley, 2006). Contact him at [dds@aueb.gr](mailto:dds@aueb.gr).

### Mapping Specifications to Design

**Naseem Mariam**

*Software Specification and Design: An Engineering Approach* by John C. Munson, Auerbach, 2005, ISBN 0-8493-1992-7, 400 pp., US\$79.95.

As its title indicates, *Software Specification and Design* takes a simple engineering approach to the critical issue of mapping software specifications to design. It also explains software measurements and advocates the use of call graphs during software maintenance.

Author John Munson has participated in

software development projects at Rockwell, IBM Federal Systems Division, and NASA. These experiences form the basis for his strategy for software development, measurement, and traceability.

### Explaining the mapping

Munson advocates using informal, natural language for specifying and designing software modules; analyzing how operations map to functionalities; mapping operations to modules; and mapping operations, functionalities, and modules to the code itself.

An operation's attributes are its name, stimulus, description, sequential dependencies, and latency. The specifications for functionality are control, temporal, data, protocol, functional transformation, operational linkage, and module linkage. The module consists of a header section, data section, algorithm description section, linkage section, functionality name, functionality file name, and version.

This methodology's beauty lies in its accurate, complete cross-links that Munson traces between the descriptions of the operations, functionalities, and modules.

Sprinkled across the book are concepts including

- operational model constraints (such as reliability, security, availability, maintainability, survivability, performance, compatibility, and size),
- architectural trade-offs (such as hardware versus software, partitioning large systems, threads, and APIs—nothing new here), and
- design trade-offs (such as module cohesion, program path count, testability, algorithmic and data structure complexity, and use of functional pointers and dynamic data).

A suggestion to the author: highlight these tips in separate boxes and checklists for easy reference and include examples that illustrate the trade-offs.

### Some praise

The book discusses key aspects of software specification and design, introducing novices to the specifications' holistic nature and its impact on the software deliverables. Call graphs need a lot

more exposure and advocacy, and Munson's chapter on them is commendable. They reduce the time to maintain software, even for a completely new team. (For more information on call graphs, see "Call Graphs: A Move towards Better Productivity," a case study I coauthored with Avanish Kumar Ojha and Nigel Joseph Johnson; [www.futsoft.com/pdf/CallGraphs.pdf](http://www.futsoft.com/pdf/CallGraphs.pdf).) The book also has comprehensive sections on general software design concepts and trade-offs.

### Some suggestions

In the preface and first chapter, Munson makes many sweeping statements that might bias readers (especially software engineers) against the rest of the book. So, I suggest that you start reading at chapter 2 and return to the preface and chapter 1 only when you've finished the rest of the book.

Mapping operations to functions, functions to modules, and modules to code—the book's core—is simple. However, creating and maintaining these mappings for large software systems could prove cumbersome. This methodology could be useful once a tool exists to create the skeletal descriptions and verify the documented links' correctness and completeness. A section on this mapping methodology's advantages and disadvantages would be helpful.

The process of transforming specifications to design needs to be addressed explicitly. The mapping methodology

**Author John Munson introduces novices to the specifications' holistic nature and its impact on the software deliverables.**

only deals with how to document your design. Munson could also improve the book's readability by including actual case studies of projects that used the methodology. Including software development and maintenance metrics of projects that did and didn't use the methodology would give readers a more objective view of the advantages.

### My recommendation

I recommend *Software Specification and Design* for those starting their software development careers. More experienced readers might feel that the book's concepts and trade-offs are common knowledge and that the methodology is just a documentation effort. They can use the book as a quick reference.

**Naseem Mariam** is an architect and engineering head at Aricent Chennai. Contact her at [naseem.mariam@aricent.com](mailto:naseem.mariam@aricent.com).

## Acquisition Meets CMMI

### Caroline Pepa

**CMMI for Outsourcing: Guidelines for Software, Systems, and IT Acquisition** by Hubert F. Hofmann, Deborah K. Yedlin, John W. Mishler, and Susan Kushner, Addison-Wesley, 2007, ISBN 0-321-47717-0, 464 pp., US\$54.99.

*CMMI for Outsourcing: Guidelines for Software, Systems, and IT Acquisition* provides valuable information to both acquirers and suppliers. The former will find a wealth of information on improving the acquisition process, and the latter will better understand the acquirer's challenges and how both parties can work together to make the collaborative effort more effective and efficient.

I particularly liked how Hubert Hofmann, Deborah Yedlin, John Mishler, and Susan Kushner used fictional characters to portray various roles in an acquisition scenario. This provided an added dimension beyond flat words on a page, letting readers "sit in" on an exchange of ideas for applying process im-

provement to the situation at hand, rather than just repeating the model and process areas. The fictional dialogue presented real-world scenarios. Some books on process improvement and CMMI describe process improvement and process areas matter-of-factly, with many cross-references to other process areas, leaving readers to determine for themselves what it all means and how it applies to given situations. So, I found this book's fictional acquisition-team approach a refreshing change.

### Salvaging a difficult outsourcing project

In the introductory chapter, we meet the senior executive assigned the dubious honor of salvaging a troublesome outsourcing effort. He believes that the assignment's success or failure will affect the future of his career. He plans to add the personnel he considers best suited to serve on the team, starting with a lead project manager and moving on to supplier staff members and an internal customer. This section emphasizes the importance of having knowledgeable, experienced people involved in the team effort, working together to improve the situation.

The team decides early on that to improve the process, it must "identify the differences between an internal development process and an acquirer-supplier process." The executive chose his team on the basis of their process knowledge and experience. This background knowledge means the team can discuss process improvement as it relates to acquisitions without rehashing basic improvement areas—the authors created experienced characters with the necessary knowledge to lay out the plan for saving the acquisition. Of course, the fictional team couldn't perform a miracle on its own, but readers observe that the leaders are competent and assume that they would have chosen the right people for their respective teams (acquisition and supplier) to accomplish the mission.

### Preparing for acquisition

Companies worldwide are incorporating process improvement methodologies into their business practices, as evi-

denced by the number of appraisals listed on the Software Engineering Institute's Web site ([www.sei.cmu.edu](http://www.sei.cmu.edu)). However, not until recent years have companies outsourcing their software development realized the problems involved with applying the CMMI process areas—designed to address systems and software development—to outsourcing. Luckily, the fictional project manager found the initial draft of *Adapting CMMI for Acquisition Organizations* on the SEI Web site ([www.sei.cmu.edu/pub/documents/06.reports/pdf/06sr005.pdf](http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06sr005.pdf)) and used it to guide the team through the various stages of preparation.

### Working through the project stages

Four lengthy chapters follow the introduction, which provides background information on the CMMI for Acquisition (CMMI-ACQ) and sets the stage for the remainder of the book. These four chapters discuss the processes related to each life-cycle stage of the fictional acquisition effort.

As the team progresses through the project's various stages, examining each scenario, the characters' conversations make clear the reasoning that leads to their conclusions. Following most dialogue is a tip that succinctly describes the important points the authors just covered. Finally, each chapter discusses how the CMMI-ACQ applies to the preceding section.

The three-section appendix provides the specifics of the CMMI-ACQ, outlining the six specific acquisition process areas and the 16 process areas common to all CMMI models. It also discusses process improvement using CMMI and includes Level 2 and Level 3 diagrams applicable to an acquisition effort.

**C**MMI for *Outsourcing* presents the CMMI-ACQ and process improvement for acquirers well. I recommend this book to anyone interested in improving the outsourcing process, whether an acquirer or supplier. ☞

**Caroline Pepa** is a senior software engineer at Tybrin. Contact her at [caroline.pepa@tybrin.com](mailto:caroline.pepa@tybrin.com).

## How to Reach Us

### Writers

For detailed information on submitting articles, write for our Editorial Guidelines ([software@computer.org](mailto:software@computer.org)) or access [www.computer.org/software/author.htm](http://www.computer.org/software/author.htm).

### Letters to the Editor

Send letters to

Editor, *IEEE Software*  
10662 Los Vaqueros Circle  
Los Alamitos, CA 90720  
[software@computer.org](mailto:software@computer.org)

Please provide an email address or daytime phone number with your letter.

### On the Web

Access [www.computer.org/software](http://www.computer.org/software) for information about *IEEE Software*.

### Subscribe

Visit [www.computer.org/subscribe](http://www.computer.org/subscribe).

### Subscription Change of Address

Send change-of-address requests for magazine subscriptions to [address.change@ieee.org](mailto:address.change@ieee.org). Be sure to specify *IEEE Software*.

### Membership Change of Address

Send change-of-address requests for IEEE and Computer Society membership to [member.services@ieee.org](mailto:member.services@ieee.org).

### Missing or Damaged Copies

If you are missing an issue or you received a damaged copy, contact [help@computer.org](mailto:help@computer.org).

### Reprints of Articles

For price information or to order reprints, send email to [software@computer.org](mailto:software@computer.org) or fax +1 714 821 4010.

### Reprint Permission

To obtain permission to reprint an article, contact the Intellectual Property Rights Office at [copyrights@ieee.org](mailto:copyrights@ieee.org).