Resource Reviews

SOFTWARE QUALITY MANAGEMENT

Achieving Software Quality Through Teamwork

Isabel Evans. 2004. Artech House, Inc. (http://www.artechhouse.com/). 294 pages. ISBN: 1-58053-662-X. CSQE Body of Knowledge area: Software Quality Management

Reviewed by Carolyn Rodda Lincoln
Carolyn.Lincoln@titan.com

Achieving Software Quality Through Teamwork provides a novel and valuable viewpoint on creating good quality software. The author brings together various models of quality, software development, and team dynamics to show how they normally interact and how they could interact.

The main thrust of the book is that quality is important to everyone on a software team, but each person has his or her own viewpoint, which might be different from other team members. There are five definitions of quality and five groups of people on a team; each group favors particular views of quality that are different from the other groups. The clashes and lack of communication hinder team success.

The five definitions of quality are: 1) product based, 2) manufacturing based, 3) user based, 4) value based, and 5) transcendent. The author also uses a quality framework based on the European Foundation for Quality Management (EFQA) Excellence Model. The EFQA model is generic and can encompass all the popular quality models such as ISO 9000:2000 and the Software Engineering Institute's Capability Maturity Model (CMM)*.

The author provides an overview of the five groups of people related to software: customers, managers, builders, measurers, and supporters. The next five chapters contain information on each group: how each fits into the EFQA model and how they interact with the other groups. The remaining chapters of the book provide an overview of the four stages in the software life span. The stages are start-up, development, delivery, and post-delivery. Each chapter has entry and exit criteria, views of quality by the five groups, and techniques to mitigate communication problems between groups. Appendix A has more information about the communication and team dynamics techniques. Appendix B contains templates for planning documents.

The book is both a broad sweep across many aspects of software development and very practical in how to improve the people part of software. For each group, it provides the "ah-hah" on why they view a project the way they do and why other groups view it differently. After pointing out the issues, the author provides both specific techniques and extensive bibliographies to assist in mitigating the risks. *Achieving Software Quality Through Teamwork* is an excellent addition to any library, but it is especially valuable for project managers and those who lead information technology departments. It is another source of valuable advice on how to organize people for exceptional results in creating and maintaining software.

Carolyn Rodda Lincoln is an ASQ certified quality manager and member of the DC Software Process Improvement Network. She is currently employed as a QA director for Titan Corporation in Fairfax, Va. She holds bachelor's and master's degrees in math and was previously a programmer and project manager.

STANDARDS

Achieving Compliance and Continuous Improvement in Software Development Companies

Vivek Nanda. 2003. ASQ Quality Press (http://qualitypress.asq.org/). 274 pages. ISBN: 0-87389-594-0. CSQE Body of Knowledge area: Software Quality Management

ISO 9001:2000 for Software and Systems Providers: An Engineering Approach

Robert Bamford and William Diebler. 2004. CRC Press (http://www.crcpress.com/us/). 313 pages. ISBN: 0-8493-2063-1. CSQE Body of Knowledge area: Software Quality Management

Reviewed by Scott Duncan
sduncan@tsys.com

There are a lot of books on ISO 9000, including the latest version issued in 2000. But there is hardly anything targeted toward applying ISO 9001 in software development

organizations. Most books just quote a few things from ISO 9000-3, the guidance document that addresses applying ISO 9000 to software development. (The new version of this guidance is 90003, with no hyphen.) These two books address the latest version of ISO 9000 and provide guidance on applying it to software development situations. This could have been done as two separate reviews, but a combined (indeed intertwined) review seems very appropriate, especially since both books take a paragraph-by-paragraph approach to most of the material. (For the sake of brevity, I'll use *ACCI* and *I9SSP* to refer to them in the review.)

Each book takes the opposite opinion on the value of the 9000-3/90003 guidance documents. *ACCI* says 90003 "has been used as a reference, and all implementation guidance…is consistent with and builds upon the interpretation included" in it. Indeed, *ACCI* recommends purchasing a copy of 90003. *I9SSP* quotes a study of 36 European software houses that 9000-3 "is not of great help." *I9SSP* also says it is "not clear whether [90003] will be any more useful" than 9000-3 given the "lack of market pressure for" 9000-3 and "little or no perceptible industry pressure for an update," that is, for 90003. Indeed, one of the goals of the ISO 9000:2000 revision was to make it more easily understood so fewer interpretations would be necessary for special industry domains. However, each book's stand on the official guidance document may not affect whether readers will like that book, since neither overtly references 9000-3/90003 in the course of discussing the requirements of 9001:2000. What is somewhat curious about this in the case of *ACCI*, which recommends 90003, is that 90003 has references to other ISO documents relevant to software development as places to find specific software development process implementation details for various parts of 9001.

Both books provide general history and overviews of ISO 9000. *ACCI* specifically discusses changes in the 2000 version of 9001 compared to the prior version. Both also offer guidance on implementing 9001 and a quality system, in general. *I9SSP* seems more "emphatic" in some of its guidance, while *ACCI* is more specific with regard to the software development life cycle. Although, in fairness, *I9SSP* is covering more than just software development. Both books also offer guidance on registration and selecting a registration/auditing firm.

When it comes to the sections on the clauses of 9001, *ACCI* devotes 111 pages to this, while *I9SSP* devotes 168 pages. Interestingly, each book is arranged so that chapter 4 covers clause 4 of 9001, chapter 5 covers clause 5, and so on, through clause 8. *ACCI* devotes the following number of pages to each clause. In general, *I9SSP* deals with each clause in more depth, even where page counts are the same or close. The actual software-specific guidance in *ACCI* is not truly extensive. Of course, the nature of the revision of ISO 9000 in 2000 was to make the material more applicable overall, so even 90003 has many sections with is no substantive software-specific guidance.

Overall, for books of comparative size, *I9SSP* seems to provide more value than *ACCI*. *ACCI* has the CD-ROM, but there are many places on the Internet offering quality manuals and other document templates.

Scott Duncan has 30 years of experience in all facets of internal and external product software development with commercial and government organizations. For the last nine years he has been an internal/external consultant helping software organizations achieve international standard registration.

ISO 9001:2000 for Software and Systems Providers: An Engineering Approach

Robert Bamford and William J. Deibler. 2004. CRC Press (http://www.crcpress.com/us/). 309 pages.
ISBN: 0-8493-2063-1.
CSQE Body of Knowledge area: Software Quality Management

Reviewed by Carolyn Rodda Lincoln
Carolyn.Lincoln@titan.com

ISO 9001:2000 for Software and Systems Providers is a book about how to use the ISO 9001:2000 standard for organizations that are doing software and systems engineering. Its purpose is to "determine how requirements might be effectively and efficiently satisfied by and to the benefit of an engineering organization." It does not include the ISO standard per se, but it does provide a paragraph-by-paragraph explanation of the standard with information on how to apply to engineering projects.

There is some introductory material for those new to ISO 9001:2000. The first chapter and one appendix give an implementation road map for the ISO 9001:2000 standard. This is for organizations that want to follow the ISO standard and do not already have a quality program in place. The second chapter is an explanation of the particular terminology used in the standard.

The rest of the book consists of a chapter on each major section of the ISO standard. It gives explanations and implementation considerations for each paragraph or requirement. The authors want to "ensure that the reader understands not only the requirements encompassed in the paragraphs but also the relationship among the paragraphs—especially

when that relationship is critical to efficient implementation." As an example, 7.3.5 in the standard is two sentences about design and development verification. The book has three and a half pages of commentary on how to do verification. There are also extensive appendices that include a case study on life cycles, template for procedures, and a sample questionnaire for registrar selection.

This book is an excellent resource for both implementers and auditors. The ISO standard is short (14 pages) and generic so that it applies to all types of businesses. Unlike the Software Engineering Institute's Capability Maturity Model Integration (CMMI), which is designed for software and systems engineering, it is not immediately obvious how the ISO 9001:2000 standard applies to engineering situations. A commentary like this book is essential to properly understand the intent of the ISO standard and to implement it as part of continuous improvement for software and systems engineers. It would be very valuable for an organization that is just beginning to look at the standard. It is also valuable for auditors who have worked with other types of organizations but not engineers. It can be used both as a text for understanding the standard and as a handbook for reference during an audit. *ISO 9001:2000 for Software and Systems Providers* should be in the library of everyone using ISO 9001:2000 for engineering projects.

Carolyn Rodda Lincoln is an ASQ certified quality manager and member of the DC Software Process Improvement Network. She is currently employed as a QA director for Titan Corporation in Fairfax, Va. She holds bachelor's and master's degrees in math and was previously a programmer and project manager.

SOFTWARE ENGINEERING PROCESS

MDA Distilled

S. Mellor, S. Kendall, A. Uhl, and D. Weise. 2004. Addison-Wesley Professional (http://www.awprofessional.com/). 176 pages. ISBN: 0-201-78891-8. CSQE Body of Knowledge area: Software Engineering Process

Reviewed by David Reed david.reed2@hp.com

Throughout the history of computing, programmers have developed languages to further abstract programs from hardware. This allowed for increased productivity and accessibility to programming, and is a key factor in the accelerating development of new programs. Today, systems span many subject areas, and often require high levels of integration. To remove the dependence on specific software or languages, model-driven architecture (MDA) was created to drive abstraction one level further. MDA is a standard framework from the Object Management Group (OMG) that allows developers to link object models together to build complete systems.

In practice, MDA is proposed to function like a higher-level compiler. It takes its set of rules and runs them through a black box that makes code. In this way, the developer needs no knowledge of the underlying details. Fifth-generation languages were to accomplish this as well, but the problem scope ultimately overwhelmed them.

This book is a well-written introduction to the MDA standard, its tools, and its technologies. The book explains the basic features of MDA, how they fit together, and how they can be used. It covers the MDA framework, including the platform-independent model and the platform-specific model; the meta object facility (MOF) — the OMG's adopted standard for metamodeling; horizontal, vertical, and merging mappings between models; building marks and marking models; elaborating models, including viewing generated models, and managing manual changes; building executable models with executable unified markup language (UML); and agile MDA development.

The sections on building marks and marking models are very interesting, because it is here that language definitions for domain-specific requirements are described. The ability to accommodate rich domains is essential for the viability of the standard, and the authors explain it well.

A weakness of the book is its lack of realistic examples, to bring the ivory tower tone down to earth. The authors state clearly that MDA is still under development, but more detail would make the topic more real and its limits better defined. Right now, no one knows how workable MDA would be in software development. Seeing a concrete example would be a great benefit.

Another concern is the context of the book. This means that this approach seems suited only to problems of large scope. How maintenance, rapid development, or smaller applications could use or integrate with MDA is not considered. These issues are a significant segment of the work performed today. Undoubtedly, MDA is a significant attempt, but its eventual success or failure is down the road for true enterprise adoption.

The authors also mention that MDA will have the ability to transform business rules into good code, without manual rewrites or tuning. However, there are no examples of this in the book. This is where the rubber meets the road, and though MDA is still developing, some sort of end-to-end example would have been appropriate.

Other MDA volumes address many of the same issues. This book is more concise and tractable than these. Even with the dearth of examples, if I had one book to recommend to my peers for MDA, this would be it.

David Reed is a 21-year veteran in quality and computer systems. He has built very large systems for clients such as Amazon.com, American Airlines, and MCI. Currently, he is responsible for enterprise architecture and data quality at Hewlett-Packard. He is a senior member of the IEEE as well as a member of IEE, ACM, and the British Computer Society. He is also an adjunct professor at Portland State University, teaching graduate courses in computer science.

Pair Programming Illuminated

Laurie Williams and Robert Kessler. 2003. Addison-Wesley Professional (http://www.awprofessional.com/). 265 pages.
ISBN: 0-201-74576-3.
CSQE Body of Knowledge area: Software Engineering Process

Reviewed by Ajit Ghai

A Google search on "pair programming" yields 1,230,000 search results; many of these relate to the book *Pair Programming Illuminated* by Laurie Williams and Robert Kessler. And for once, this is a good thing. Fun to read, this book is a practical guide for those who want to seriously experiment with pair programming as well as for those who don't know what pair programming is all about.

Programming is typically done by individuals sitting in front of a screen, typing away on their keyboard. It is a lone occupation. Not that programmers do not interact or engage others, but it has been and still is mostly an individual exercise (like most other things done in an organization).

Williams and Kessler define pair programming as a "style of programming in which two programmers work side by side at one computer, continually collaborating on the same design, algorithm, code or test." The key words here are "two programmers," "one computer" and "same design." The "driver" programmer types or writes the design, while the "navigator" observes the work of the driver, looking for tactical and strategic mistakes or defects. The navigator is the strategic, long-range thinker. The navigator and driver can brainstorm at any time. Effective pairing is very active with the two roles being switched periodically.

The claim is that pair programming will significantly reduce the number of things that go wrong, and increase the number of things that go right. Pair programming will benefit an organization in quality, timeliness, morale, trust and teamwork, knowledge transfer, and enhanced learning. If one has programmed for any length of time, and even if one hasn't, it does make sense. Williams and Kessler cite sufficient evidence to back up this conclusion. They do this in a manner that is engaging and fun.

Pair programming rests on the assumption that complex tasks can be done better with collaboration. The word programming in "pair programming" refers not just to the writing of code but also includes design, debugging, testing, and so on.

This is all good theory but how does one convince his or her managers to put twice the number of developers on the same amount of work? A chapter entitled "Overcoming Management Resistance to Pair Programming" is devoted to the economics of that and concludes that pair programming yields better quality code in about the same amount of time. If readers are familiar with how much time programming really takes when all the defects are found and fixed, they will realize, thanks to the evidence presented, that this is not such a tall claim.

The authors note that pair programming is not "all sunshine and roses, a miracle cure for all your ills." Hence they offer some cautions along the way. These include issues related to scheduling of pairs, dealing with experts and heroes, collocation of workers, noise considerations, skills imbalance between pairs, and handling disagreements.

Case studies of pair programming are presented within the extreme programming methodology, as well as in a collaborative software process, a process developed by Williams during her pair programming research. The message conveyed is that pair programming can fit within any software development methodology, and for that matter can be used by programmers who do not use any formal methodology.

The book ends with appendices that cover economic analysis, pair programming in the classroom, and an introduction to test drive development. A pair programming tutorial is also included.

An excellent book for practitioners of software development, this is a must read for programmers, designers, and their managers. Even if readers do not intend to practice pair programming, this book contains valuable lessons for all software professionals. Also, it is loaded with great references, a good table of contents, and a useful index. Beg, borrow, or buy it.

Ajit Ghai has worked in the technology industry for more than 25 years in systems/product development and project management. He lives in Ottawa, Canada, where he is currently project director at the Canadian Pharmacists Association. He has consulted for Bank of Montreal E-Business, headed project management at 3M-AIT, and held positions of director, project management, and director, product line management, at Philips Electronics NV. He can be reached at ghai@rogers.com .

A Requirements Pattern: Succeeding in the Internet Economy

Patricia L. Ferdinandi. 2002. Pearson Education, Inc. (http://www.pearsoned.com/). 506 pages.
ISBN: 0-201-73826-0.
CSQE Body of Knowledge area: Software Engineering Process

Reviewed by Hillel Glazer
hillel@entinex.com

Get this book. If only business executives would take the time to appreciate how non-trivial requirements engineering really is, and just how much return on effort to the company and to the success of software projects is to be gained by doing proper requirements engineering!

A Requirements Pattern: Succeeding in the Internet Economy focuses on requirements engineering for Internet and e-commerce software projects. Through the example of Internet/e-commerce software applications, Ferdinandi demonstrates the importance of understanding, appreciating, and accounting for all the relevant stakeholders in a software project. Chapters are well organized, use creative and thoughtful examples, and include a box with the key topics and how they relate to Internet projects.

The first three chapters initialize readers with an introduction to requirements engineering and its specific application to the uniqueness of Internet products, then boots-up into greater detail about the processes of eliciting, gathering, and managing requirements before instantiating the requirements into sets of information based on various focus areas. An Internet *requirements pattern* being defined as "a reusable template of questions that can be used to elicit the requirements necessary for building an Internet product" is an important question lightly answered in the first chapter. Noting that, the pattern must account for "all the different types of requirements that must be captured." As an example of just how poorly understood or applied requirements engineering is, Ferdinandi obliges readers with a straightforward definition of a "requirement" and of "requirements engineering" and points out a common error in the software industry. Rather than attempting to satisfy all stakeholders' needs with a single list of requirements, Ferdinandi says that there should be a unique requirements "set" for each perspective/view of the requirements.

She makes creative use of the popular game "Tetris" to demonstrate the effects of gaps in requirements, and the life cycle of requirements: new, analyzed, specified, prioritized, and organized. Although *patterns* and *anti-patterns* are introduced in the first chapter, they are dealt with in detail in later chapters.

An important concept that makes the pattern approach viable is that of the *dimensions* of a requirement. Requirements are depicted as existing in a three-dimensional matrix with each requirement occupying a cell on the matrix and its location defined by its three primary dimensions: community, perspective, and focus. Focus addresses the who (people), what (information), where (location on technical topography), when (event), why (five types of business rules), and how (processes) of each requirement.

Demonstrative of her superior insight into software development, Ferdinandi points out two nonfunctional requirement focuses: product and project constraints. To be truly comprehensive, these nonfunctional requirements must be addressed with every requirement throughout the requirements engineering life cycle.

The next sequence of chapters walks through the concept of *patterns* and *anti-patterns*, and quality as applied to requirements engineering. The theories/concepts of patterns are more easily internalized than anti-patterns. The chapter on patterns provides further detail into the process of kicking-off the project using patterns, laying out the pattern's specifics, and performing the gap analysis on requirements using the pattern.

Anti-patterns are then explained in chapter five as unwanted (but common) behaviors modeled to purposely seek out and close gaps in knowledge, ensure all stakeholders are involved, and challenge and validate business processes. The chapter on antipatterns amounts to a post-graduate education in requirements. Ferdinandi invests an appreciable amount of space and mental equity in helping readers understand how to formalize the requirements engineering effort to ferret out missing requirements and gaps in business processes, all of which are constant threats on typical software projects.

Ferdinandi's chapter on quality will be refreshing to readers of AQP. To that end, Ferdinandi draws on IEEE 830 for eight characteristics important in creating "quality" requirements. The final chapters are about the human factors of requirements: managing them, the roles and responsibilities of people working with the requirements during the development life cycle, and practical realities of working through the requirements pattern.

On managing requirements readers are given a primer on the Software Engineering Institute Capability Maturity Model® for Software (SW-CMM®). From the SW-CMM®, Ferdinandi discusses requirements management and configuration management (two of the six level 2 key process areas) and how they are instrumental in effectively and meaningfully managing requirements. Understanding everyone's (possibly changed) roles in the requirements step of the development life cycle is critical to the success of the entire project.

Reviews

A word of caution to readers: Ferdinandi provides an excellent level of detail so that implementers of this process can "hit the ground running" regardless of project size or complexity. With a textbook as rich as this one, however, one must not get caught in the trap of believing that the level of detail and formality described is appropriate for every Internet project.

Ferdinandi does not assert that the requirements pattern eliminates either of these traps. Even if readers aren't keen on following the details of text's methods, the extensive bibliography will cut significant time out of anyone's research on requirements engineering, management, processes, quality, and standards.

Hillel Glazer is the principal and CEO of Entinex, Inc. and a member of ASQ's Washington, DC, Section (509). He brings a broad spectrum of experience in process engineering and technology management. He has written and spoken on the subject of process discipline in the information age. Since 1988, he has successfully adapted and evolved these disciplines professionally, across the Internet, software, and manufacturing industries.

Managing Software Engineering Knowledge

Aybuke Aurum, Ross Jeffery, Claes Wohlin, Meliha Handzac, eds. 2003. Springer-Verlag (http://www.springerny.com/). 375 pages.
ISBN: 30540-00370-3.
CSQE Body of Knowledge areas: Software Engineering Process; Project and Process Management

Reviewed by John W. Horch
J.Horch@ieee.org

The audience for this book is alleged by the editors to be "researchers and practitioners on knowledge management in software development." To be sure, there is something for both categories of readers. But perhaps the most avid audience would be "postgraduate students undertaking research in software development." This is because this is a book of 17 chapters, averaging about 18 pages in length, but averaging slightly more than 38 references each. Even the preface and part introductions have references. This wealth of references will give a researcher a marvelous start in finding information and information sources.

The four editors have each been responsible for one part of the book. Part 1 has three chapters dealing with "Why Is It Important to Manage Knowledge?" Handzac says that there are three main categories of reasons for starting knowledge management initiatives. These are minimizing risks, seeking efficiency, and enabling innovation. The papers in Part One review a bit of software engineering history, discuss knowledge management solutions, and present some challenges to be addressed by knowledge management in software engineering.

Part 2 presents five papers discussing knowledge management frameworks, looking at some that may help manage software engineering knowledge, and addressing existing knowledge management problems. This is a good read for software developers.

Part 3 is also five papers in length and deals with "some possible methods to use when working with knowledge management in software engineering." Quality assurance and control play a major role in this part, being important discoverers of knowledge that should be preserved.

Finally, Part 4 discusses ways to actually bring the knowledge management techniques, processes, and tools into the industrial setting. The four papers provide insight into various challenges faced by actual implementation situations.

This book struck me as an exercise in creating academically oriented papers, citing as many references as possible (most of them in proceedings or journals of interest to researchers, but not usually easily available to practitioners). It is not mentioned, nor is it probably important, whether these papers were written expressly for this book, or have been presented elsewhere as well. If they have been, they were probably well received by those whose interests or needs they served.

That said, the editors have done a good job of collecting the contents into a reasonably well-organized volume. For those in the field of software engineering who are interested in or tasked with managing the plethora of knowledge that arises from software engineering activities, this could be an important source. It may be worth the time to give it a once over.

John Horch has more than 40 years of experience in all aspects of software development and management. He is author of *Practical Guide to Software Quality Management–Second Edition*, a Senior member of IEEE and of ASQ, and serves on the Editorial Board of ASQ's *Software Quality Professional.*

CMMI Distilled: A Practical Introduction to Integrated Process Improvement, 2nd Edition

Dennis M. Ahern, Aaron Clouse, and Richard Turner. 2003. Addison-Wesley (http://www.awprofessional.com). 320 pages.
ISBN: 0-321-18613-3.
CSQE Body of Knowledge area: Software Engineering Process (NOTE: First edition previously reviewed in *Software Quality Professional* vol. 4, no. 4)

Reviewed by Pieter Botman
p.botman@ieee.org

The authors of this book have issued a second edition in order to take into account changes to the CMMI (up to version 1.1, released in 2002) and to reflect some of the initial user experience gained since the release of the first edition. Changes to the CMMI models in version 1.1 included a change of discipline emphasis from "acquisition" to "supplier sourcing," and a new process area called "integrated supplier management." Several aspects of integrated product and process development (IPPD) have been added to the integrated project management process area.

The basic structure and tone of the book remain unchanged, and it still serves as a good introduction to the CMMI, its rationale, and its use in process improvement. The sections on selecting staged/continuous representations and assessments (now lumped under the more general term "appraisals") remain.

The book retains as its centerpiece a good overview of the models and disciplines (systems engineering, software engineering, IPPD, and supplier sourcing) that make up version 1.1 of the CMMI–SE/SW/IPPD/SS.

In this edition, the authors expand on the case studies and business benefits of process improvement using the CMMI, and offer new thinking about CMMI evolution, including a discussion about the development of a "partially staged" model. In a partially staged model, a subset of CMMI process areas (rather than all) would be staged at a maturity level. Appraisals and their interpretations would naturally change, but the authors suggest that this approach might allow the CMMI to be used more flexibly, perhaps in domains outside of engineering. The book contains some thought-provoking sample requirements to be met by such a new partially staged representation.

The second edition retains and improves upon the value of the first edition, and is a worthwhile read for anyone learning about the CMMI.

Pieter Botman is a professional engineer (software) registered in the Province of British Columbia. With more than 20 years of software engineering experience, he is currently an independent consultant, assisting companies in the areas of software process assessment/improvement, project management, quality management, and product management.

PROJECT AND PROCESS MANAGEMENT

Practicing Software Engineering in the 21st Century

Joan Peckham and Scott J. Lloyd, eds. 2003. Idea Group (http://www.idea-group.com/). 296 pages. ISBN: 0-931777-50-0. CSQE Body of Knowledge areas: Project Management, Software Engineering Processes, Metrics

Reviewed by Trudy Howles
tmh@cs.rit.edu

This book is a compilation of 18 technical papers categorized into three sections: applications and implementations, system design, and managing software projects. The purpose of this book is to "introduce new and original work from around the world." Only a few papers were authored in the United States, and unlike other compilations of this type, most of the content does not appear to be reprints of journal articles or conference proceedings.

The editors discuss the theme of the book and admit that the topics expand the body of common knowledge in software engineering. They describe the ordering of the book as an attempt to tell a story; however, the topics did not flow well.

The collection is interesting, but it seems unusual for some topics to be included in the same book. As examples, two papers describe narrowband filter designs and a third discusses a framework for network service discovery. It seems a stretch to consider these topics to be elements of traditional software engineering.

The book has several problems with diagram and figure layouts. Several figures have been reduced to the point where the text is too small to read and the fonts are blurred. A paper by Argentinean authors contains two figures with Spanish text—clearly a language translation oversight.

Readers will find a wide breadth but a limited depth in any one topic area. Some papers are very theoretical and others are more general. Those deeply entrenched in specific body of knowledge areas may find that the paper topics are too diverse and that only a few are of interest.

Trudy Howles is an assistant professor of computer science at the Rochester Institute of Technology in Rochester, NY, after working for many years as a software development engineer and consultant. She holds a master's degree in computer science and is currently pursuing her doctorate. Howles, a CSQE, chairs the ASQ Rochester Section's Software Quality Task Group and is a member of the section's Executive Committee.

Lessons in Project Management

Tom Mochal and Jeff Mochal. 2003. Apress (http://www.apress. com/). 312 pages. ISBN: 1-59059-127-5. CSQE Body of Knowledge area: Project and Process Management

Reviewed by Christiana Melton
cmelton@ilrd.com

In this easy-to-read introductory primer to project management, Mochal and Mochal present the day-to-day experiences of a new project management advisor as he helps coworkers to understand project scope, recognize problems, and resolve issues related to project management.

The authors provide lessons in which concepts, tools, and techniques are used to obtain workable solutions for commonly experienced project management challenges. The lessons, or stories, are relevant everyday examples of situations experienced in a fast-paced business environment.

The authors guide the reader through the fundamental necessities of initiating, maintaining, and tracking the success of a project through the TenStep Project Management Process (TenStep), a complete project management methodology developed by Tom Mochal. The TenStep process provides a scalable solution for defining the work, building and managing the workplan, managing issues and communication, managing scope and risk, managing documents, managing quality, and gathering metrics. The reader is able to quickly utilize presented concepts outlined in the TenStep methodology by accessing forms provided in an enclosed CD-ROM.

The publisher-suggested audience is intermediate to advanced; however, the basic fundamental concepts presented would suggest a beginner audience.

Christiana Melton is a project manager at Instrumentation Laboratory in Lexington, Mass. She holds a master's degree in engineering management from the Gordon Institute of Tufts University, and a bachelor's degree in medical technology from Bloomsburg University in Pennsylvania.

Agile Project Management

Jim Highsmith. 2004. Addison-Wesley (http://www.awprofessional.com/). 284 pages. ISBN: 0-321-21977-5. CSQE Body of Knowledge area: Project and Process Management

Reviewed by Scott Duncan
sduncan@tsys.com

Highsmith is a well-known figure in the agile methods community. He is a coeditor with Alistair Cockburn of the Addison-Wesley series on agile software development, the director at Cutter Consortium for Agile Project Management, and a winner of the Jolt award for his book *Adaptive Software Development*. Highsmith says this book is aimed at project managers who should not be viewed as "functionaries who merely comply with the bureaucratic demands of schedules and budgets," but who must be "intimately involved in helping teams deliver products."

There is a lot of material in this book, but one of the most important is getting at the author's philosophy behind the subject matter. Highsmith gives this right away in the first chapter. He says successful projects, when the intended product presents the organization with "opportunity, uncertainty, and risk," are driven by "evolution and adaptation, not planning and optimization." By reducing "the cost of experimentation" (that is, iterative and incremental development), our model of the cost of projects changes "from a process based on anticipation (define, design, and build) to one based on adaptation (envision, explore, and adapt)." To do this requires "highly [self] disciplined" teams that can do things with urgency, not in a rush. "Anyone who practices ad hoc development under the guise of agile methods," Highsmith says, "is an imposter."

Highsmith next outlines the "five key business objectives for a good exploration process such as agile project management (APM):

1. Continuous innovation – to deliver on current customer requirements
2. Product adaptability – to deliver on future customer requirements
3. Reduced delivery schedules – to meet market windows and improve return on investment
4. People and process adaptability – to respond rapidly to product and business change
5. Reliable results – to support business growth and profitability"

The role of project management in all this is one of "systematically reducing the uncertainty and mitigating the risk over the life of the project." This is how APM helps teams deliver products.

After describing the four "core agile values" from the Agile Alliance's Manifesto (that is, responding to change, working products, customer collaboration, individuals and interactions), Highsmith notes that project management too often focuses on "compliance activities, not delivering value" because people are taught to be "project administrators, not project managers." Compliance, documentation, status reporting, milestone approvals, and so on "assist" in meeting various regulatory and fiduciary requirements as well as in providing information to control projects, "but they don't add value." Agility, according to Highsmith, "is the ability to both create and respond to change in order to profit from a turbulent business environment" and to "balance flexibility and stability."

The rest of the book discusses topics such as:

- Customers and products—about delivering value and dealing with required, but nonvalue-added tasks
- Leadership-collaboration management—about reliability vs. repeatability, collaborative exploration (of needs/requirements), and building adaptive teams
- An APM framework—consisting of phases named envision, speculate, explore, adapt, and close
- Building large adaptive teams—about scaling agile practices

- Reliable innovation—about "delivering innovative products to your customers" and being "excited about going to work every day"

For those who are interested in "how to do" agile projects, regardless of the specific method and associated developer practices (for example, Crystal, DSDM, FDD, Scrum, XP), this book is very informative and worth reading.

Scott Duncan has 30 years of experience in all facets of internal and external product software development with commercial and government organizations. For the last nine years he has been an internal/external consultant helping software organizations achieve international standard registration.

Advances in Software Maintenance Management

Macario Polo, Mario Piattini, and Francisco Ruiz, eds. 2003. Idea Group Publishing (http://www.idea-group.com/). 305 pages.
ISBN: 1-59140-047-3.
CSQE Body of Knowledge area: Project and Process Management

Reviewed by Scott Duncan
sduncan@tsys.com

This is a collection of 10 "proposals" from several countries "with the goal of exposing recent techniques and methods for helping in software maintenance." At first glance they seem very theoretical/academic in nature, as they do not explain to a programmer how to "do" maintenance. But the intent of the book, as the quote suggests, is to examine maintenance and various software engineering approaches that could help in performing maintenance. The focus of the book seems to be at the organization rather than practitioner.

The opening chapter is about organizational structure and its affect on managing change (which, after all, is what maintenance is about). The next chapter follows up by discussing definitions of various types of maintenance, taking the IEEE definitions to task for not clearly enough delineating the types of maintenance and presenting problem management ideas from the author's corrective maintenance maturity model.

A third chapter summarizes extreme programming (XP) concepts and attempts to draw a link between XP practices and maintenance costs. A variety of graphs are presented, but except for one from classic sources like Boehm, DeMarco, and Gilb, no empirical data seem to have been used to construct any of them. They are, it appears, "thought experiments," that is, if the XP practices work as described, then the pattern of maintenance costs should be as shown on the charts. The only real conclusion in this chapter relates to the size of an XP team, which the author says should be around 10 people for effective use of the XP techniques. This is consistent with what many sources on agile methods say about agile teams, in general.

The fourth chapter discusses "patterns" and their impact on maintenance. One obvious type of pattern has to do with those related to actual software design. The article summarizes a variety of sources on such types of patterns, but does not go into detail on any of them. The second main type of pattern mentioned in the chapter is for the maintenance process itself and how patterns for reengineering, for example, might be applied. The final type mentioned has to do with organizational-specific patterns, and the author suggests that "maintenance is so organization dependent that it is difficult to codify, in pattern form or any other way, much that can genuinely be of use in many organizations."

Standards are addressed in a fifth chapter, but largely with regarding to modeling methods and languages (for example, UML, XML). Indeed, considerable space in the chapter is devoted to discussing an XML-based Unified Meta-Model, of which just four pages (in a 37-page chapter) explain how XUM applies to maintenance.

A sixth chapter addresses "Migrating Legacy System[s] to the Web" largely from a business process reengineering perspective. Using various "wrapping" techniques to handle server issues and reverse engineering of the user interface seem to be the focus of this chapter, which emphasizes a case study of a COBOL-to-Web project driven by a shortest-time-to-migrate strategy.

The next two chapters focus on fairly standard material on risk management and cost estimation as applied to maintenance issues. However, the material is a good summary of risk and estimation issues and considerations.

The ninth chapter discusses "A Methodology for Software Maintenance" (MANTEMA) developed by a university research group in Spain and "a multinational maintenance organization that provides services to big banking and industrial enterprises." The researchers who wrote this chapter noted that one European study "has shown that most software organizations do not use any methodology for software maintenance, although they do use it for new developments." They find this "really surprising" since "61 percent of the professional life of programmers is devoted to maintenance work." Besides "planneable (sic) maintenance," there is a brief discussion of the use of a predictive model ("any" one) "to plan for error corrections" of what are called "nonplanneable maintenance."

The final chapter discusses managing maintenance and an "environment" for doing so using their MANTIS project work that they term the Big-E Environment. As in other chapters, models and meta-models are discussed and, not surprisingly, MANTEMA is the methodology proposed.

So, for those who want a practitioner's book on current software maintenance methods, tools, and issues, this is not the book. On the other hand, for those who are looking for a survey of some general maintenance topics and want to see an example of what some European organizations are doing to explore new approaches to maintenance, this book does a nice job.

Scott Duncan has 30 years of experience in all facets of internal and external product software development with commercial and government organizations. For the last nine years he has been an internal/external consultant helping software organizations achieve international standard registration.

The Hands-On Project Office: Guaranteeing ROI and On-Time Delivery

Richard M. Kesner. 2003. Auerbach Publications (http://www.auerbach publications.com). 333 pages. ISBN: 0-8493-1991-9.
CSQE Body of Knowledge area: Project and Program Management

Reviewed by Eva Freund
Eva.freund@ivvgroup.com

This is a book readers will love to hate, especially if they are current or information technology (IT) manager "wanna-bes." It is ponderous reading because it contains a lot of valuable information including links to an electronic library of the tools, templates, and examples cited in the text. The library includes completed templates, in addition to blank templates, that enable the reader to move quickly from an understanding of the book's context to the hand's-on application of that knowledge in the workplace. Additionally, it also serves as a handy compendium of best practices and practical recommendations.

In the past, as Kesner writes, the IT executive labored in relative obscurity while the IT organization, begrudgingly, was viewed as a necessary expense to the organization. However, today IT serves as a core constituent in most business plans and is expected to demonstrate value-added contributions in the form of quantifiable return on investment (ROI). Kesner describes tested processes, techniques, and tools that will immediately improve the timeliness and quality of the delivery of IT products and services that demonstrate IT's value-added contributions.

Today's IT organization also has to demonstrate a reasonable level of operational economy, consistency, and reliability in the execution of its assignments. At the same time it has to achieve and exhibit success in addressing customer requirements. Critical factors for achieving these crucial objectives are: communications, delivery management, resource management, architected and managed solutions, and collaboration. Each of these areas is covered.

Chapter 1 creates an internal economy model systematically reviewing the operational, organizational, financial, and human resources dimensions of a typical enterprise-level IT organization. Chapter 2 introduces the concept of a project management office (PMO) and describes the roles and responsibilities within the PMO. Whether staffed or virtual, the PMO serves as the focal point for leveraging best practices. Chapter 3 outlines a practical alignment and planning process to ensure that efforts and resources are focused appropriately on those elements that are of the highest priority to the enterprise.

Chapter 4 examines service delivery management best practices, including the design and implementation of service-level agreements, the measurement of service-level performance, the maintenance of reporting processes, and general support of the IT customer relationship. Chapter 5 encompasses project scoping and commitment making, risk and resource management, day-to-day oversight of project engineering and delivery processes, delivery measurement and reporting processes, and general coordination of IT project activity. Chapter 6 offers a comprehensive tool set for discovering and documenting customer needs in ways that are comprehensible to nontechnical readers and relevant for those assigned to build, operate, and service IT systems.

In Chapters 7 and 8 the author draws upon his own experience to illustrate how the tools of knowledge management may be used to improve overall IT team performance. Chapter 7 describes how an IT organization can identify its own community of best practices. Building on Chapter 7, Chapter 8 demonstrates how the principles of knowledge management can be used to better manage the information relating to new technologies and assets within the IT organization. This chapter provides practical advice on leveraging the services of the PMO to oversee IT investment strategies. While Chapter 9 reviews the primary roles of the PMO and provides practical examples of how investment in a PMO will benefit the entire organization, it also sums up the approach to assisting IT organizations to achieve repeatable success in their service and project delivery efforts.

Finally, the appendices include hard-copy versions of the major tools discussed throughout the book and also cite the Web site where the user can find an expanded set of models, templates, tools, and forms, as well as examples of the tools as employed in actual business settings.

This is a book readers hate to love. It focuses on simple changes in organizational process that can lead to higher customer satisfaction. The author encourages readers to adapt the recommendations to their own unique needs. Most importantly, readers are taken on a journey of exploration and an examination of one's own business context.

Eva Freund is a subject-matter expert: Independent Verification and Validation (IV&V). She currently performs IV&V activities for the National Archive's ERA Project. Freund received her bachelor's degree from Fairleigh Dickinson University and her master's degree from Goddard College.

METRICS

Software Metrics: A Guide to Planning, Analysis, and Application

C. Ravindranath Pandian. 2004. CRC Press LLC (http://www.crcpress.com/). 286 pages. ISBN: 0-8493-1661-8. CSQE Body of Knowledge areas: Software Metrics, Measurement, and Analytical Methods

Reviewed by Carolyn Rodda Lincoln
Carolyn.Lincoln@titan.com

Software Metrics is a guide to software measurement, metrics, and models. The first few chapters provide some introductory theory on metrology, the science of measurement. Then there are chapters on data analysis in the frequency, time, and relationship domains. The next major section explains models for process, estimation, defects, and decision support. The final chapter is on metrics system implementation.

Although the topics are relevant and the examples are software-specific, the author does not provide useful information. The level of discussion is such that one would need an extensive background in the subject to understand what was being presented. In that case, however, the reader would not need the book. Many techniques are presented, but often with only a paragraph, formula, and/or sample graph. It is not obvious when the technique is used, how it is used, or what benefit it provides.

There are many other worthwhile books on software measurement, including abundant free material from Practical Software and Systems Measurement. Anyone interested in measurement should take advantage of other resources rather than spending time or effort on this book.

Carolyn Rodda Lincoln is an ASQ certified quality manager and member of the DC Software Process Improvement Network. She is currently employed as a QA director for Titan Corporation in Fairfax, Va. She holds bachelor's and master's degrees in math and was previously a programmer and project manager.

SOFTWARE CONFIGURATION MANAGEMENT

Jessica Keyes. 2004. Auerbach Publications (http://www.crcpress.com/). 640 pages. ISBN: 0-8493-1976-5. CSQE Body of Knowledge areas: Project Management; Configuration Management, Software Metrics

Reviewed by Eva Freund
eva.frueund@ivvgroup.com

Perhaps the most striking relevance of this book is that while the process of software configuration management (SCM) has not fundamentally changed much in the past 30 years, the program elements being managed have changed immensely. The information technology (IT) environment has moved from centralized mainframes with a scattering of programming languages to an environment replete with decentralized, networked, Web-enabled systems employing thousands of clients using hundreds of software packages and dozens of programming languages.

To manage SCM in the present environment one needs to understand that having a finely tuned SCM process remains key to implementing and managing SCM in this diverse environment. To further complicate the mix, add in automated tools along with their library systems. A well-integrated configuration management (CM) process works to ensure that SCM will not exist in isolation. Thus, a CM program will require solid project management inherent with a CM plan describing the CM activities and tasks, the work breakdown structure, the CM schedule, and also the risks and metrics.

By adopting the processes and concepts outlined in this book, readers will have everything they need to implement and execute a sound SCM organization. Whether one is a project manager with responsibility for CM, a CM manager, or a CM team lead, this book would be beneficial. From documenting the planning results in the CM plan, using consensus standards, to evaluating and selecting a CM tool, to identifying CM metrics, and to identifying and then managing a CM process, the reader may find value within its pages.

Perhaps, the best feature of this book is that it is life-cycle oriented. For each phase of the generalized life cycle it identifies the relevant SCM activities and the SCM milestones. The template for the CM plan requires not only the traditional activities of configuration identification, configuration control, configuration status accounting, and configuration audits, but also identifies the engineering objectives and describes the SCM responsibilities for each phase of the development life cycle.

In summary, using this book as a guide will allow one's CM organization to be:

- *A support function* in that it supports program engineers and developers, the program, the organization, and often, the customer
- *A control function* in that it controls specifications, documents, drawings, requirements, tools, software, and other deliverables

- *A service provider* in that it supports people and controls data

Eva Freund is a subject-matter expert: Independent Verification and Validation (IV&V). She currently performs IV&V activities for the National Archive's ERA Project. Freund received her bachelor's degree from Fairleigh Dickinson University and her master's degree from Goddard College.

DOMAINS OF APPLICATION

Telecommunications Essentials
Lillian Goleniewski. 2002. Addison-Wesley (http://www.awprofessional.com/). 588 pages.
ISBN: 0-201-76032-0.

Reviewed by Scott Duncan
sduncan@tsys.com

Designing and testing software certainly requires one to understand the application area and this book explains the telecommunications domain. This book does so quite comprehensively.

It's about telecommunications architectures, technology, and functionality. It covers the basics of transmission, bandwidth, multiplexing, and signaling through to explanations of broadband, wireless, the Internet, and home networks. Along the way topics such as standards, multimedia, and security are covered, and a comprehensive 57-page glossary is included. There's even mention of Arthur C. Clarke, in 1947, theorizing about a geosynchronous orbiting satellite as a communications broadcast tool.

It would seem valuable to have references explaining major industries where software plays a significant role and telecommunications is one of them. If I had to take an "open book" test on telecommunications or review the field in preparation for a job interview, I'd find this book very handy.