

Resource Reviews

GENERAL KNOWLEDGE

Practical Support for CMMI-SW Software Project Documentation Using IEEE Software Engineering Standards

Susan K. Land and John W. Walz. 2006. IEEE Computer Society/John Wiley and Sons (<http://www.wiley.com>). 343 pages. ISBN: 0-471-73849-2

CSQE Body of Knowledge area: General Knowledge – Standards, Specifications, and Models

*Reviewed by Pieter Botman
p.botman@ieee.org*

Individuals facing the task of defining software processes and constructing a corresponding documentation set certainly earn their paychecks. The CMMI-SW, intended as a tool for measuring the maturity of software development organizations, embodies and promotes many software engineering practices, but does so without prescribing detailed technical procedures, criteria, or standards. The CMMI® can be viewed as process centric; it is a tailorable framework into which the software process designer may “plug in” specific technical practices (for example, software size estimation procedure) and standards (for example, source code format, unit test code/path coverage).

This book was intended to support individuals responsible for process definition and documentation, providing templates (based on

IEEE software engineering standards) for “all activities associated with software development projects.” The authors claim that the book will help CMMI-oriented organizations with the design and development of process and product documentation. They also assert that even organizations not pursuing CMMI accreditation “will learn how the application of IEEE standards can facilitate the development of sound software engineering practices.”

In the opening chapters, the authors review CMMI concepts, including the staged and continuous model, and the generic practices associated with levels 2 and 3. The authors introduce the IDEAL model for process improvement and provide advice related to process improvement.

The middle of the book is organized by CMMI PA. In discussing the “support documentation” for each PA, the authors first lay out the generic goals and specific goals of that PA, and map these goals to a document type or artifact type. Then, going on to discuss each of the PA process goals and practices in subsections, the authors finally introduce concepts or standards from a relevant IEEE standard. At times the authors include some detail when discussing a recommended practice or standard. For example, the decision analysis and resolution PA contains a recommended outline of a risk management plan as well as a detailed step-by-step process for a basic decision analysis and resolution process, complete with graphic depicting a decision tree. This portion of the book brings together a lot of information under a given PA.

This mapping of process goals and practices within each PA in

various documents and artifacts represents one major source of value in the book—it provides a suggested general documentation structure that is consistent, that meets the higher level process goals, and that can be examined for completeness before descending into the details of the individual documents. These long PA sections, however, could use some editing. They are at times difficult to follow, and contain many subsections that are poorly distinguished or separated.

The book includes a CD-ROM with 38 files, most of them Microsoft Word templates for forms, plans, process artifacts, and even product deliverables. The templates for the significant planning and control documents contain guidance content, both at the level of the overall document and individual sections.

The second major source of value in this book is the encapsulation of IEEE standards and recommendations into various templates. Most of the IEEE software engineering standards are referenced in this book, and many of them are directly reflected in the various template work products, such as system requirements specification, various test plans, quality assurance plan, and so on. There are a few templates originating from other sources, including the authors themselves. There are too many sample forms, checklists and document templates to review individually, but I found all of the non-IEEE templates to be thought-provoking, even if not detailed.

The book succeeds and is of greatest value to process designers who are familiar with the CMMI process framework. For them this book provides a useful bridge into

Reviews

the world of IEEE software engineering standards with the discussions and with the templates provided.

For readers and organizations not oriented toward the CMMI process framework, this book provides less value. The tracing of CMMI PAs down to generic document and work product types contains some useful software engineering discussions. Beyond mere document templates, this book presents a great deal of advice and information at both the policy level and the specific practice level. Not all of this wisdom, however, is accessible or unified; it is often buried in a structure following that of the CMMI PAs, and not the various IEEE standards. A reader focusing primarily on the various IEEE standards might be better served by a book devoted to them, such as *The Road Map to Software Engineering Standards* by James. W. Moore.

Pieter Botman is an independent consultant with more than 20 years of experience in software engineering and project/product management. He assists companies in the areas of software process assessment/improvement, project management, quality management, and product management. He is a Senior member of the IEEE, and holds the CSQE, and CSDP designations, among others.

CMM[®] and CMMI[®] are registered trademarks of the Software Engineering Institute, Carnegie Mellon University.

Weinberg on Writing: The Fieldstone Method

Gerald M. Weinberg. 2005.
Dorset House Publishing
Company
(<http://www.dorsethouse.com/>).
194 pages.
ISBN: 093263365X

CSQE Body of Knowledge
area: General Knowledge

*Reviewed by Matthew Heusser
Matt.Heusser@gmail.com*

A few years ago I had a peer who was constantly asked details about his project. Eventually, in clear annoyance, he would shout that he had sent all of those details to the entire company last week. The problem for him was not producing the documentation, it was producing useful, readable documentation. This book is for people who don't want to be this guy.

Taking all possible things to write about the project and compressing them into the salient, meaningful pieces takes skill. Writing them in such a way that they are actually interesting takes even more skill. While most computer science programs require plenty of programming, few require students to learn either of those skills.

Weinberg on Writing is a book on writing about real things for technical people. At its heart is the concept of collecting real stories, ideas, and anecdotes, then reusing them to create documentation.

The agile movement gave us documentation as a story (cards). This is a book about telling stories in a meaningful and entertaining way. It's not fluff or platitude; Weinberg produces real examples and details that allow readers to apply the metaphor concept to the written word.

There is a whole slew of techniques to learn in writing, including organization, transitions, verb tense, and author's voice; Weinberg covers all of these in a unique way.

The techniques in this book are especially helpful in writing magazine articles or when producing technical documentation.

That said, the book is not perfect. A fieldstone wall is a collection of rocks combined with mortar; Weinberg takes that process and applies it to writing, where the rocks are anecdotes and concepts and the mortar is organization and transitions. He stretches the analogy pretty far, and it gets a bit tiring toward the end. In the second half

he talks about organization, and as he is writing the book, he tells about the changes he is making to the organization. It's a great idea, but it can be hard to keep track of. At one point, I had to ask myself is he writing about a topic or writing about his writing on the topic?

Despite its flaws, I have never read a book so specific, down-to-earth, and approachable about the writing process. Most of what I have learned about writing was learned by finding "bad" writing, learning the symptoms, and trying to avoid them. This book provides positive, specific steps to improve the quality of one's writing, along with exercises.

A few years ago I went to the writing section of the local library and checked out every book I could find. I found better books about the business of writing, pitching stories, and fundamental English rules. As for the actual process of writing, this book is far and away better than anything else I have read. Buy it today.

Contemporary Issues in Ethics and Information Technology

Robert A. Schultz. 2006.
IRM Press
(www.irm-press.com).
208 pages.
ISBN: 159140780X

CSQE Body of Knowledge
area: General Knowledge

*Reviewed by Nels Hoenig
hoenign@gmail.com*

This book is a review of the issues faced by information technology (IT) professionals in the area of ethics. The author uses the framework for ethical problems first published by the late John Rawls, a noted ethicist. The author had a close friendship to Rawls and is very open to his role in explaining and amplifying Rawls' ideas.

The book is divided into four sections:

- Section 1 is a very good review of ethics and their origin in the IT world. It covers the broad subject of ethics and why societies develop these social rules on behavior. It also provides some examples in history of violations in ethics by individuals and organizations.
- Section 2 deals with ethics and the IT professional. It was interesting to note this profession is still young when compared to that of doctors, lawyers, or accountants, all of whom have a very defined code of ethics and penalties for violations of accepted standards. The author points out that there are some codes of ethics emerging in the IT world but they have yet to gain the traction needed to become industry standards.
- Section 3 deals with the ethics of IT and the user community and covers the issues on intellectual property, privacy, and some of the many challenges created by the emerging "E" economy.
- Section 4 wraps up the entire process and asks some interesting questions on the value of IT and the conflicts inherent between the individual and the corporation or government. He also asks the questions on the ecological impact of the ever-accelerating rate of change and how all the obsolete technology will be handled.

I am not an ethicist but found many interesting ideas in this book, and I appreciated the reminder that the industry, which is filled with professionals and impacts the daily life of a majority of humanity, is just a baby when compared to the other professions with similar impact.

SOFTWARE QUALITY MANAGEMENT

Code Quality: The Open Source Perspective

Diomidis Spinellis.
2006. Addison-Wesley
(<http://www.awprofessional.com>). 576 pages.

ISBN: 0-321-16607-8CSQE

Body of Knowledge areas:
Software Quality
Management; Software
Processes

Reviewed by Scott Duncan
sduncan@westfallteam.com

Diomidis Spinellis has another book that relates to quality called *Code Reading*. The main idea in the first book is that developers should learn how to read other people's code before they are asked to write their own. *Code Quality* is about learning "how to judge the quality of software code."

Because it focuses on code, there are lots of code examples in the book and they can be downloaded from <http://www.spinellis.gr/code-quality/sourcecode.html>. They are the same code examples as those found on the CD-ROM accompanying the *Code Reading* book. Some reviews of *Code Reading* have complained about jumping between languages, focusing on language-specific issues, being good mainly for less experienced programmers, and discussing or teaching languages too much. If one is going to write extensive books like *Code Reading* or *Code Quality*, two things should not be a surprise: the use of the word "code" implies language specifics have to be an important consequence of the topics covered and, in a large book, there are only so many generic things one can say that apply to everyone under all

circumstances. I would also add that focusing on less experienced people is probably a good idea since more experienced people would know these things in order to be considered "more" experienced. Also, the focus on C, C++, and Java as the languages for the examples seems unsurprising given that these are the current languages of choice for new systems and the languages used in the examples, coming from many open source sites.

This being said, Spinellis notes that some 61 percent of the examples are for C (as the most common denominator), 19 percent for Java (covering OO topics), and 4 percent for C++ with an emphasis, in all these cases, on Unix® examples more than Windows® (since software tolls and APIs for the former also have ports to the latter whereas the reverse is not true, making it also a common denominator). As to the fact that these only add up to 84 percent, Spinellis doesn't say where the other 16 percent of the examples come from, though some "examples" are not strictly code, such as directory listings, trace outputs from code profiling, and so on.

For those who write, review, and structurally test code, this seems like a worthwhile book. I think people working with systems written in other languages could still find good ideas for their writing, reviewing, and testing activities. Since both this book and *Code Reading* are really a split of what was headed toward being a very large book, the two, together, are probably a worthwhile set to have.

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.

Reviews

Safe and Sound Software: Creating an Efficient and Effective Quality System for Software Medical Device Organizations

Thomas H. Faris. 2006.
ASQ Quality Press
(<http://qualitypress.org>).
358 pages.
ISBN: 0-87389-674-2

CSQE Body of Knowledge
area: Software Quality
Management

*Reviewed by Scott Duncan
sduncan@westfallteam.com*

The book is about what its name suggests: how to develop a quality system for medical device software development organizations. The author states that the purpose of the book is to "help regulatory affairs and quality managers and consultants of software medical device design and development organizations navigate the complex course of regulatory compliance, operational excellence, product quality, and customer satisfaction to create an effective and efficient quality system." The author does caution that the treatment given to the applicable regulations is "broad" and "is not intended to provide sufficient detailed knowledge required to implement and manage a quality system." These two statements, coming one right after the other in the Preface, may seem contradictory. Shortly after the second, though, the author acknowledges that "the quality or regulatory professional" would have to "conduct more detailed research and analysis to tailor the application of these concepts and recommendations to his or her own organization, products, and customers." So it appears that the author believes the book is a good start, but cannot address the necessary details of a specific organization and their relation to applicable stan-

dards and regulations. This is, then, an obvious but necessary disclaimer when dealing with a topic that could have serious legal/contractual implications. Readers must decide for themselves what the law/contracts require and not blame the book if they fail to do so.

What the book does do is march, straightforwardly, through a variety of topics with one chapter for each:

1. Operational Requirements for Software Medical Device Manufacturers: some basic material including an overview of regulatory matters.
2. Ramification of Software Defects: as applied to safety threats due to software defects in medical devices.
3. Software Quality: a generic "what is quality" overview.
4. Software Design and Development: a long "good practices" summary, which seems enough to make the point.
5. Safety Risk Management: a reasonably generic coverage of risk management topics.
6. FDA Regulation: a summary "discussion" of the major FDA quality system requirements.
7. International Requirements: a summary discussion of European Union expectations for "a medical device manufacturer to affix a CE marking to its product" based on the EU's medical device directive.
8. ISO Quality System Standards: a simple identification of what ISO 9001:1994 & 2000, ISO/IEC 90003: 2004, and ISO 13485:2003 are.
9. Other Compliance Requirements: a few paragraphs each about HIPPA and Sarbanes-Oxley plus a paragraph each on "some common regulatory procedural requirements."
10. Final Thoughts on Implementation: a few word listing all the other things one could look at relative to a (software) quality system such as the CMM® (not the CMMI®) and

the Malcolm Baldrige Award criteria.

What, for me, barely rescues this book and makes it worthwhile are the comments and definitions interspersed throughout the main chapters plus the content of the first appendix. The latter seems to be the author's outline of a recommended quality system based on his experience with the regulations and standards alluded to in the book. The interspersed comments and definitions summarize the chapter content in terse but understandable "requirements like" form.

For readers who are entering the medical device software field, or for those who for some other reason really need to know what a quality system in that field might require, then this book would probably be a useful introduction. However, unless one is in "startup" mode, I don't think the book would advance the cause of an organization that, beyond "startup" mode, probably needs to do what the author suggested in the Preface: "Conduct more detailed research and analysis."

SOFTWARE ENGINEERING PROCESSES

Mastering the Requirements Process, 2nd edition

Suzanne Robertson and
James Robertson. 2006.
Addison-Wesley
(www.awprofessional.com).
560 pages.
ISBN: 0-321-41949-9.

CSQE Body of Knowledge
area: Software Processes

*Reviewed by Ralph R. Young
ralph.young@ngc.com*

Getting the requirements right is critical to developing good software and also to sound project management. In this second edition, the

authors build upon the concepts and approaches presented in the first edition of this book and their other book, *Requirements-Led Project Management: Discovering David's Slingshot* (Addison-Wesley 2005).

The authors are principals of the Atlantic Systems Guild, a well-known transatlantic consultancy that specializes in the human dimensions of complex system building. The guild has promoted the Volere Requirements Process Model, a process for discovering, verifying, and documenting requirements. Diagrams that look like mind maps are provided to describe the process and its major components. The authors advocate this process as a list of steps that need to be taken to identify and provide the appropriate deliverables. Chapter 2 of the book provides an overview of the process, Chapter 10 describes how to use the Volere Requirements Specification Template, and Appendix A provides the model itself. The authors assert that the model does not imply any sequence but rather is an asynchronous network—any combination of processes can be active at any time. The reader is encouraged to concentrate not on the process but on the flows of information. Learn more about the model at www.systemsguild.com. See also the materials available at www.sophist.de and www.scenarioplus.org.uk.

There are a lot of requirements-related books available; however, each has unique viewpoints and contributions that provide helpful advice and guidance. For instance, this book provides valuable guidance concerning stakeholders—a comprehensive checklist to identify stakeholders (noting that they should be individually named); thorough guidance concerning how to do a detailed analysis of stakeholder goals, constraints, risks, scenarios, and requirements; and even project sociology analysis templates to help readers discover the relevant stakeholders, identify gaps without

stakeholder representation, and agree on their decision-making structure.

The authors emphasize what I have come to believe is the fundamental challenge and root cause of many projects' problems in performing effective requirements engineering: The project must work closely with the stakeholders to "discover" the *real* requirements. They advocate use cases and scenarios as preferred requirements elicitation techniques. They enumerate and describe many ways to perform requirements discovery, and provide guidance concerning how to achieve quality in the requirements specification. Simulation (using prototypes and storyboards) is advocated as a way to help find requirements.

A surprising statement is that "Technology is important to our study only when it is outside the scope of our work The technology inside the work is irrelevant for gathering the requirements" (pp. 131-132). The authors emphasize focusing on business requirements. My experience is that technology is central to systems and software, and that one cannot really separate it from the business requirements. It would seem difficult to conceive of a software system of any size or importance in the absence of technology aspects.

The authors provide good discussions of functional requirements, nonfunctional requirements, and the requirements retrospective—an event designed to evaluate whether the requirements are deemed satisfactory to proceed with system design and development. This suggestion alone is invaluable and a good reason to spend some time digesting the book.

In short, this book is a valuable contribution to the requirements literature that will help readers perform their critical role effectively. It meets the authors' intention to provide a reliable companion to intelligent work.

Ralph R. Young is director of process improvement, systems and process engineering, Northrop Grumman Information Technology Defense Group. He is the author of *Effective Requirements Practices* (Addison-Wesley 2001), *The Requirements Engineering Handbook* (Artech House 2004), and *Project Requirements: A Guide to Best Practices* (Management Concepts 2006).

Software Engineering Quality Practices

Ronald Kirk Kandt. 2006.
Auerbach Publications
(<http://www.auerbach-publications.com>).
256 pages.
ISBN: 0849346339

CSQE Body of Knowledge areas: Software Engineering Processes, Software Quality Management, Software Configuration Management

Reviewed by Ponmurugaran Thiyagarajan
pmrcapricorn@yahoo.com

Learning from others' mistakes is certainly a better practice than learning from one's own mistakes. Ronald Kirk Kandt, the author of this book, believes in this concept and, through this book, wants to help software engineers and managers of various information technology organizations. In this book, he shares the wisdom and learning he acquired during his 30 years of software development experience. He shares useful tips and methodologies to prevent the software engineering related mistakes he has made and has seen others make. The insight he provides throughout this book may help a software engineer produce high-quality software products right the first time.

The author has done a good job of sequencing the topics. He starts the discussion by citing the industry

problems caused by low-quality software products and improper processes. He provides facts and data to prove his points. He also describes the basic concepts of product and process quality and provides common approaches to develop quality software. Chapter 1, Introduction, is a key chapter in this book as the following chapters reference it and its contents.

This book also details various approaches to efficiently perform change, personnel, and project management functions, which cannot be deemed as a pure software engineering topic. He provides practical approaches to improve the current software systems and processes from both people and project perspectives. He highlights the importance of Software Process Engineering Groups (SPEG) and talks about the best practices they must follow. To stress the importance of having good software professionals in a team, the author discusses a few experimental studies conducted by other researchers. A procedure for hiring and retaining good people is also described, along with other personnel management concepts.

The author also writes in detail about various pure software engineering topics. He describes good practices and procedures to be followed for effective configuration management, good requirements management, efficient design, coding, testing, and inspection. He adopts a step-by-step method to describe these concepts, which is commendable. The author also provides useful tips to write good user documentation. Another highlight of this book is the discussion on various competing approaches. Approaches like extreme programming, Rational Unified Process, cleanroom software engineering, and Capability Maturity Model Integrated® are described from a practical perspective. A tabular illustration of these competing approaches to software engineering is available in this book.

This book will be a useful resource for the software professional. The contents can enrich the skills of the practicing software engineer. The author has taken care to mix theory and practical concepts of software engineering, which will be helpful from both understanding and implementation perspectives.

Ponmurugarajan Thiyagarajan is a software consultant at Tata Consultancy Services Limited. Currently he is working on a software development project assignment for a telecom company based in Omaha, NE. He holds a bachelor's degree in mechanical engineering and a master's degree in industrial engineering. Thiyagarajan is also an ASQ Certified Software Quality Engineer (CSQE), QAI Certified Software Quality Analyst (CSQA) and holds Microsoft, Sun Microsystems, and IBM technical certifications.

PROJECT AND PROCESS MANAGEMENT

Successful Packaged Software Implementation

Christine B. Tayntor. 2006.
Auerbach Publications
(<http://www.Auerbach-Publications.com>).
314 pages.
ISBN: 0849334101

CSQE Body of Knowledge
area: Project Management

*Reviewed by Nels Hoenig
hoenign@gmail.com*

This book is a wonderful resource of information on the pitfalls of buying and implementing software solutions. The book is oriented toward the customer or buyer, but I only wish all my customers had this information as a resource during the many implementations I have been on as a solution provider.

It is common knowledge that a majority of software projects have issues in meeting the schedule or budget or performance expecta-

tions. This book takes the reader on a tour of the entire process of defining the need, reviewing the options, and negotiating the agreement, and through the implementation process to completion. It also educates readers to the dangers and secrets of success in implementation packaged software solutions.

The book covers the entire process from deciding to purchase a packaged solution to the end with the post-implementation audit. The chapters are easy to read and the terminology is easy to understand. The examples are very relevant and one could easily put them into context for projects.

As anyone who has ever done a project implementation knows, there is no right or wrong way to do the various implementation steps. The author does a nice job of providing the advantages and disadvantages of the most common choices. Each chapter also contains a summary of the relevant material and a nice introduction.

The chapters are arranged into sections and then chapters within each section. The sections represent phases of a software purchase. The author also provides an example of a project charter and a list of acronyms used in the book.

The book is a little pricey at \$80 on Amazon but a valuable reference. For those who are going to buy or sell packaged software solutions, I would recommend buying a copy.

Project Requirements— A Guide to Best Practices

Ralph R. Young. 2006.
Management Concepts, Inc.
(<http://www.mngtconcepts.com>). 254 pages.
ISBN: 1-56726-169-8

CSQE Body of Knowledge
area: Software Project
Management

*Reviewed by Terry Bartholomew
Terry.bartholomew@ngc.com*

Recent advances in hardware and software technology have not helped us write better software specifications. A popular cartoon shows a software engineer asking the customer about what he wants to accomplish with the software. The customer states that he does not know what he wants to accomplish because he does not know what the software can do. The engineer then explains that she can design the software to do whatever he wants, but first she needs to know his requirements. The frustrated customer finally asks the engineer if she can design the software to tell what his requirements are. In my opinion requirements elicitation on many projects has not matured much farther than the level of this cartoon.

The buzzword for systems development in today's competitive environment is "faster, better, and cheaper." Yet many software projects struggle just to complete development period." Well-known failures such as the FBI's Virtual Case File (VCF) make the news, while many more failures go unreported. The VCF failure was due to several factors including scope creep and ill-defined requirements. The sad truth is that no software project is ever completed faster, better, and cheaper with significant scope creep and ill-defined requirements. In fact, in the June 2006 edition of *CrossTalk*, Capers Jones lists new and changing requirements during development as one of the five root causes of software project failure.

The problem is not a lack of technology or a shortage of qualified people. The problem certainly is not a lack of software development methodologies to follow. There is an abundance of all three. The problem is a lack of leadership in the basics of project management. Leadership on basic issues is what Ralph Young's

latest book, *Project Requirements, A Guide to Best Practices* (Management Concepts 2006) is all about. His first two books focused on "what" to do (*Effective Requirements Practices*, Addison-Wesley 2001) and "how" to do it (*Requirements Engineering Handbook*, Artech House 2004). In this book, he describes the basic project management practices needed to prepare the ground for performing the tasks described in the first two books. However, why stress requirements in a book aimed at the project manager?

A quick review of the table of contents in *Project Requirements* shows that the book is not for those who want to learn about the latest fad, or what tools to use, or even how to write a requirement. This book is for those who want to build a sound foundation for their software project to rest on. The real strength of this book lies in how easily it integrates requirements tasks with the Software Quality Engineer Body of Knowledge. The first two chapters focus on prevention of requirements errors early in the life cycle, before they become software errors. Chapters 3 through 5 address the all-important aspects of leadership, team building, and partnering. There are chapters for improving project communications, coaching team members in sound requirements practices, and setting goals and objectives.

The book contains chapter after chapter of good solid advice, based on Young's real-world experience. However, parts of the book contain exceptional advice. The section on using a quality assurance audit, not as a reporting tool but as a coaching tool, is a good example. The section on risk management is another area that needs more attention in requirements management. People often treat requirements as being equal to each other. Although this approach makes measuring progress easier, requirements are not all created equal. Each requirement has

a different potential cost, schedule, and technical risk impact to a project. The biggest risk a project can have is not recognizing this fact. Young provides a sound, systematic approach to integrating risk management with requirements management and mitigating this often-overlooked problem.

I recommend this book to anyone who is interested in how project management basics apply to requirements management. To ignore the basics Young covers in his book is to make the same mistake that many projects make, but do not easily recover from. To ignore these basics is the best way to see one's project cancelled, written up in the newspaper, and maybe even portrayed in a cartoon.

Terry Bartholomew has 24 years of software project development experience, most of which has been in requirements analysis and development for computer systems in the federal government.

Dr. Roger Pressman Series of Online Software Engineering Courses

http://www.qaieschool.com/innerpages/our_offeringsShop.asp#software_eng

Introduction

This review differs from *SQP's* usual reviews. Several reviewers examined the online course based on Roger Pressman's book, *Software Engineering: A Practitioner's Approach*, which was subsequently made into a VHS course. Some, but not all, reviewers provided comparisons to these other media.

SQP editors found some comments that were made repeatedly by reviewers. These elements were extracted and are presented here, separately, in the section of General Comments. This review then presents a section-by-section discussion, with some explanatory material about the structure of the

Reviews

online course. Not all sections are reviewed with the same degree of detail. The print version of this review emphasizes the highlights; the online version is complete.

General Comments

Note: Sentences with personal pronouns such as "I" or "my" are quotes from particular reviewers, given without quotation marks, that were echoed by other reviewers; other sentences are summaries of comments or suggestions made by several reviewers.

Content and Currency

The course follows Pressman's book closely; however, it is becoming increasingly obsolete. These examples will suffice:

1. There are many references to ISO, but no mention of ISO for software.
2. There are many references to CMM, but not to CMMI. It also talks about the 2001 year-end summary for the Software Engineering Institute (SEI) even though there are data out to 2005.
3. There are many references to CASE tools, but I don't think I've heard anyone mention CASE tools in years.

Implementation Issues

The course uses British English, rather than American English, for example, "fortnight" and "developed afresh." The course assumes that the student will use Internet Explorer as the browser. No reviewer was able to use another browser (FF 1. 5, NS 7. 2, or 8, Opera, Safari, and so on) comfortably. IE's pop-up blocker continually interfered with the site, even when turned off!

Since Firefox is gaining popularity, the user will be well served if he or she has the flexibility to use either IE or Firefox when accessing the course material.

The course requires the use of a low-resolution setting, and displays the material fully across the monitor. It will not run in a window on a higher resolution display.

The program takes over the entire screen. The graphics and scenarios on the right side of the screens are somewhat superficial and do not seem to contribute to the user's understanding of the topics. This area should be enhanced and the space should be used to insert more content or other, better defined, learning aids.

Navigation Issues

There is more than one set of forward/backward/navigation buttons/arrows showing on a page at one time, which makes it confusing at first. There is no provision to return to the middle of a lesson. When the student signs off, he or she can only return to specific points within a module. Several reviewers found it difficult to know where they were and where they had been.

I ran into what seemed to be a minor flaw in my section. All courses are divided into three "compartments" or segments: the main course body, a quiz, and a follow-up survey. Ordinarily the learner selects one of these segments when entering the main course page. I was within the course body, and reached the end of that segment, expecting to be forwarded automatically to the following quiz. The navigation buttons on the final screen are a little unclear, as the primary navigation button for the course screens allowed me only to return to previous screens. Another navigation button on the top of the screen seemed to be operative, but did not take me to the next section (the quiz). This aspect of navigation could have been a little better designed and implemented.

It has too many windows to get places, and the tracking mechanism seems erratic. Several reviewers reported, "I have taken all parts of the module but it says 'started - not completed.' I went through all sections again, quickly, and it still only says 'started.'"

I also had a problem taking the final quiz at the end of a module; it locked up my browser.

Usability Issues

There are too many different kinds of browser tricks and techniques in the first module. I had to pay more attention to "what would happen next" instead of "what am I being taught." With the second module I understood the user interface better and could deal with it. This can be frustrating for a new user.

"Too long to get places. It runs slow" was noted by both DSL and broadband users. Moving within a module was generally faster than moving from one module to another.

Regarding the test, the same help text is presented for all the links. Hints or explanations are not available for any link while taking the test.

The technical presentation of the material is adequate, but not visually appealing, nor is the application very intuitive to use. The instructions for each screen should be more apparent, and the possible actions from the links on the pages should be better explained.

The courses do not take advantage of the Flash animation and imaging technology. Most screens present one or two concepts in written form, accompanied by some static illustration or graphic. Even in scenarios in which a lengthy dialogue is presented, the illustrations are still static, cartoon animation, and not dynamic animation or true video. This is adequate, but less than impressive. At several points in the course an audio clip of Roger Pressman was integrated into the presentation. This is a nice touch, as it allows Pressman to make succinct summary points in an authoritative manner. The scenarios and lengthier conversations would have been more compelling in true video, but this is not really significant to the learning outcome.

The material is presented in an organized, clear manner, although there is substantial redundancy between the sections of the module.

On a positive note, I do think the audio tracks add to the online experience.

At certain times the “check your understanding” device comes in the form of a multiple-choice question, and at other times it comes in the form of an open-ended question posed to allow the learner to consider the concept just presented (and to answer on a “scratchpad.”) In the latter case, after the learner considers the question and enters his or her thoughts on the scratchpad, the system offers “our thoughts,” a response to the question proposed by the course author(s). This form of “check” also is valid, but one or two of the responses could have been better worded.

The “check your understanding” questions are a nice learning device, but the wording and the precision of the responses (“our thoughts”) proposed by the course authors is disappointing.

One slightly negative impact of the overlap of topic areas is the design of the quiz. In the Formal Technical Reviews course quiz, I was surprised by the inclusion of several questions on “background topics” not listed in the course outline (for example, McCall’s quality features). Such questions are certainly related to my sections, but learners taking a single course might be discouraged by one or two questions on the quiz that do not pertain directly to the central topic(s) for that course. Perhaps the quiz questions for all courses are generated from a central question bank, but I would suggest to the course authors that they validate the quiz questions and answers from a course-specific perspective.

Management Issues

I have several comments to make. First, QAI has set up some administration and planning tools that allow training managers or supervisors to designate course selections for their employees, complete with a “learning plan.” There is an intention perhaps that QAI is hoping to become a mass training contractor by providing these features for training managers

or supervisors, directly on the QAI system, because the system provides status information about students’ progress in their respective courses, and allows students to “request” courses (presumably to their supervisors, who approve online within the QAI system). Naturally, individuals planning and purchasing courses on their own behalf will not gain much from these features.

Second, QAI surrounds each online course with a “learning community” of sorts. From what I was able to discern, this community consists of an online discussion forum for the course, a clearinghouse for posting related articles, and possibly one or more “mentors” for that course. None of these resources was actually present for my sections at the time they were reviewed.

Third, in addition to promoting contact with other students and mentors, the QAI system allows for learner feedback. Within the structure of each course, after the course quiz, a third segment allows the learner to enter feedback to QAI concerning the contents of that course. The questions on the course survey cover content as well as organization and delivery. In another context, the QAI system allows learners (and perhaps supervisors, mentors, or others) to “contribute” to any given course. The contributions can include references to Web sites, books, and articles. No such contributions had been posted for the courses at the time of the review.

Other Issues

The time I required to complete my section was slightly under the rating. Keeping in mind that the course is targeted at software practitioners with zero to two years of experience, I believe the ratings are reasonable.

One reviewer found the length of the course to be a good time-frame in which to absorb the data, although it is not quite as long as the course list indicates.

The material is presented in many ways: cartoons/animation, videos, questions to answer, references, glossary, and so on. This is good for those who like a multimedia approach, but others may find it distracting.

Errors

There are a few typographical errors (for example, “lacina” for “lacuna”), none of which caused serious difficulties. More disturbing is that a quiz has errors: two different phrases have the same definition stated to be the right answer.

Software Engineering Process Approach, SPA (Mod 1)

This module, Software Engineering Process Approach, is almost 30 hours long, and contains the following sessions:

- SE101: An Introduction to Software Engineering (four and a half hours)
- SE102: Software Process Models (five hours)
- SE103: Common Process Framework (five and a half hours)
- SE104: Software Process Improvement (seven hours)
- SE105: Advanced Software Process Model (seven and a half hours)

SE101: An Introduction to Software Engineering

The availability of educational course work online is an appealing idea because it provides another avenue to discover new fields and learn more about one’s current industry. I appreciate the opportunity to review this course for QAI e-school.

Many things contribute to the success of an online training program, among them are the vast technical aspects that present the material to the student in an attractive and easy-to-use manner. The instructional content of the program must also be clear, concise, and current to deliver meaningful information. Each area should be fully developed for the success of the program as well as that of the student.

Reviews

Since I have not read Pressman's book on this subject, my review refers only to the above referenced QAI e-course.

The breadth of the course content seems to be on target for an introductory course, but the discussion doesn't flow very well. As a result, it is difficult to follow the lessons on some of the screens.

In summary, while there are some areas that need improvement in this course, I am confident that these items will be addressed in future enhancements. I'm also optimistic that this course and QAI e-school will have much success in the education marketplace.

SE102: Software Process Models

I checked the Pressman book out of the library, and found that the online course does follow the book.

My review is not very positive. I don't think the online course is worth the money, as one can get the same information from the book. If one is not disciplined enough to read the book, then the online course would be the way to go. I don't know the cost involved with the online course, so I can't compare pricing.

I did not enjoy this course; however, it did refresh my memory on a few things. Some of the information in the course is dated and should be updated to current trends in the industry (agile development process is one of them). Also, the course is more for someone who has very little or no experience. It might be more suited for a college student rather than someone in the marketplace.

SE103: Common Process Framework

I got interrupted a few times and it took me about four hours to take the course including initially figuring out which "next" buttons were needed to go from section to section versus slide to slide, some note-taking, making comments as I went along, taking the test, and providing some feedback on a post-

course survey. In his book, Pressman does not go into the same detail as in the course about defining what a common process framework (CPF) is and what framework vs. umbrella activities are. Indeed, his book only mentions CPF on three pages, with one being devoted to a small object-oriented (OO) example. However, there seem to be bits of CPF ideas at other points without the reference to CPF. The course details seem to head toward Pressman's own Adaptable Process Model (APM), which he points to from this course where licensing registration seems to be needed for any significant detail.

Initially, I thought a course of this kind (and proposed length of five and a half hours) could use some takeaway material so people would not feel the need to write down a lot from each screen. However, there is enough redundancy that I took very few notes. Separate links exist for a glossary and a summary of some of the other courses, but I did not find them necessary (or useful) in taking this course.

The module starts with a simulated "case study," that is, an example used throughout the material. Basically, it just introduces some "characters," noting that one, due to company growth, now has to come up with a process everyone can use. Then the two main module topics start. After the second item ends, the case study starts up, but just uses the characters to state that a process framework that can be used by all has been accomplished. I consider this pre- and post-material to be a waste of time, as it is disconnected from the rest of the material, that is, none of the characters or their situations are ever used in the two main module topics.

It does become clear after the first few slides of the course and again later that this module expects users to have taken another module outlining classic process models. This may not be a course that, overall, one could expect to take piecemeal.

I do not think any of my comments relate to the lack of prior knowledge, however, other than to point out where this seemed needed or assumed.

Introduction to the CPF (First Main Topic—25 slides)

My first comment is on slide 4 where it asks for my ideas on what a project manager (PM) needs to know to plan for software development activities. The course example that came up read like what a process engineer/designer would want to know to define activities. I've never met a PM who thought this kind of "design" planning was his or her job. I was thinking about the kind of historical data, domain info, experience levels of staff, and so on would be needed to "plan" activities. The course suggests issues of entry/exit criteria, tasks steps, deliverable types, and so on.

On the next slide (no. 5), there is an exercise that the course example seems to show requiring/assuming knowledge that clearly does not come from anything I have seen in the module (for example, what standards would be used for defining the analysis activity for which the course example as a specific IEEE one you would have to have known beforehand).

Later, on slide 17 is the first pointer to Pressman's Web site, which, if all his links are followed there, could preclude the need for this module in the course. (Along with judicious reading of his book.) I also ran into a small annoyance here, which may just be an oversight on this one course slide: Mousing over text in the graphic brought up information boxes. Some prior slides indicate that this should be done and highlighted that text. This slide does not, so for a while I got into the habit of mousing over graphics on every slide just in case.

Slide 19 states that a CPF contains all the task sets one needs to adapt the overall CPF to a specific project's needs. That implies that

Reviews

someone has to get these sets from somewhere. Pressman never goes into detail about this level of the framework. Later on it becomes clearer that, indeed, this is what is required. At that point, though, Pressman is suggesting, and using examples from his already prepared APM version of a CPF.

On slide 21 it says the task set chosen is based on project "characteristics," but what a characteristic is does not get defined here, nor in the glossary. (There is brief mention of two to three example characteristics on slide 16 of the next topic on establishing and adapting a CPF.) Also, it says the main activities of a CPF break down into actions, but these are pretty high level, based on the example where it says the "framework activity" called "modeling" breaks down into the two actions of analysis and design. I'd say Pressman's use of some terms (like "action") do not mesh with the level of detail of most other authors when they use that term. He also notes that, in a specific adaptation of this framework activity, design may not be needed. But he states that a following framework activity, called construction, must be performed by any adaptation of the process. So it is not clear, from the normal definition of design, how one skips doing anything like that and would be able to go from analysis to coding, for example, which is the first action under construction. An example would help here, but there really are none from any real-life situations or even one "made up" for a case study link.

Establishing and Adapting a CPF (Second Main Topic – 32 slides)

I was surprised near the beginning when slide 4 says to "identify the process model that best suits the organization," since I thought the CPF was not about picture one process model. But then, on slide 6, it says that an organization may need more than one such model. The again, in the closing case study

material, it suggests that the Spiral Model might be the one most organizations would want to pick because one can fit a more waterfall-like, sequential model into part of the Spiral, but not the reverse.

Slide 9 says there are five to seven framework activities, but Pressman only mentions a CPF with five: customer communication, planning, modeling, construction, and deployment. What are the two others?

By the time I got to slide 13 of this topic, I sensed considerable redundancy with material from the introduction topic. By slide 18, Pressman starts referring to his own APM version of a CPF.

Slide 21 shows an example of characteristics to use to come up with a specific set of tasks for a specific project type by using a set of criteria and weightings to average into a type selection.

Slides 23–28 show specific APM characteristics for defining project "rigor," that is, the formality expected within each type selection. Basically, users evaluate a set of characteristics on a 1–5 scale. Pressman gives the APM set. Then each scale value is multiplied by a "weighting factor" from 0.8 to 1.2 (although where the specific value in this case comes from is not explained in this module).

Slide 30 then notes how one takes the results and averages them together to give one final number that is used to look up the appropriate project type in a table. This all looks very much like the approach used by COCOMO or Function Point "adjustment factors" for project estimation. The caveat comes when Pressman says that, besides this numerical calculation, one also uses one's experience and common sense to pick his or her project type, which then determines what specific task sets one will use. Unfortunately, an example of a task set is never provided, even from Pressman's own APM version of a CPF.

After this topic ends, we get back to the closing of the case study. Besides its other problems, it also does not suggest how long it might have taken the process engineer character to interview everyone to discover all the kinds of projects done, then create a CPF with the levels of detail for people to use. From my experience doing such process design, it's a lot of work. Maybe that's why Pressman skips discussing the time element and directs people to his already defined APM?

My constant harping on Pressman's APM is not because I have doubts about its individual value, just that I doubt one could do this module and then hope to begin to create a CPF on one's own. So, overall, I would not want to pay for the material like this module has in it. It would be an interesting 45-to-50 minute talk at a conference or perhaps a webinar on an approach to process design. The notes I took for this amounted to perhaps a quarter-page of handwriting, which were mostly definitions of his key ideas related to framework and umbrella activities (which all process instantiations from a CPF need to do).

SE104: Software Process Improvement

I did relearn some things I had forgotten.

It is a good overview of the material, but it could be much better. I'm not sure I would recommend it to anyone who had to pay for it.

The book does not even have CMM in the index – only CMMI. The text is all about CMMI and nothing about CMM.

SE105: Advanced Software Process Model

This course gives an enjoyable although largely surface survey of software process models ranging from component-based process models through formal methods to cleanroom software engineering and agile methods. The coverage is accurate if somewhat shallow, as might

Reviews

be expected in a single course with about 110 visuals.

The course begins with an opening scenario represented as a dialogue that the QAI folks call a "movie." It's actually a set of visuals that are mouse-click advanced, which illustrate a workplace interaction usually showing how a problem comes up in practice. A question addressed by software processes is: "How are we going to get control of our software development?"

Defined processes address such things as reuse concerns and the perennial problem of changing requirements without changing the expected delivery date. The course makes the issues clear. It begins with coverage of component-based software development. It moves on to discussing formal methods. Forty percent to 60 percent of defects are traceable to errors in requirements arising from subjectivity, ambiguity and incompleteness, and lack of verifiability. These are things that formal methods were designed to address. The course introduces users in general terms to specification languages such as Spec, Z, CSP, and LARCH.

Then nine visuals address cleanroom software engineering focusing on the object of zero defects. The cleanroom process strives to replace the conventional analysis, design code, and test sequence with a more formal process ensuring correctness verification and statistical use testing aimed at producing certifiably correct code.

Finally, the course addresses the recent flock of agile software processes, which focus on light-weight methods. These are adaptive and quick, aiming at responding to fluid requirements with methods that ensure close customer involvement and incremental delivery of software builds. The agile methods examined include extreme programming, SCRUM, dynamic system development method (DSDM), and feature driven development (FDD). On the whole the course gives an

effective overview of the whole software process landscape. The course concludes with a closing scenario, a 19-panel movie that summarizes when to apply the various processes in a storyboard kind of format. I enjoyed taking the course. The seven and a half hours given as the time required is rather liberal. It can be completed more quickly. However, if the student spends time examining and reading the collateral material incorporated by reference he or she could easily spend that much time on the course. The student acquires a broader perspective on the range of software processes available to address development concerns. To actually apply those processes will, of course, require more in-depth work.

SPM (Mod 2)

This module, Software Project Management, is 37 hours long and contains the following sessions:

- SE201: Basic Concepts of Software Project Management (eight and a half hours)
- SE202: Software Project Measurement and Metrics (three and a half hours)
- SE203: Basic Concepts and Techniques of Estimation (six and a half hours)
- SE204: Measuring the Size of Software Products (six and a half hours)
- SE205: Outsourcing Project Work (five hours)
- SE206: Risk Management (seven hours)

SE201: Basic Concepts of Software Project Management

No specific comments

SE202: Software Project Measurement and Metrics

I checked the Pressman book out of the library and see that the online course does follow the book.

My review is not very positive. I do not think the online course is worth the money, as one can get the same information from the

book. If one is not disciplined enough to read the book, then the online course would be the way to go. I don't know the cost involved with the online course so I can't compare pricing.

I did not enjoy this course, although it did refresh my memory on a few things. I think some of the information in the course is dated and should be updated to current trends in the industry. Also, the course is more for someone who has very little or no experience. It might be more suited for a college student rather than someone in the marketplace.

SE203: Basic Concepts and Techniques of Estimation

Basic Concepts and Techniques of Estimation is a basic course, but its prerequisite is Basic Concepts of Software Project Management. The estimation course is supposed to be six and a half hours, but it only has 63 screens. This course has a more logical flow and provides a good explanation about why estimation is so important. It also explains creating a work breakdown structure through functional decomposition and two estimation techniques at a high level. One technique is COCOMO, which is based on size. The other is a more bottom-up approach where each component is compared to the effort of a similar one.

The advantage of an online course should be that the student gets the benefit of an instructor-led course but can take it at any time. It was not obvious to me that the content or presentation of these courses is as good as reading a competent text, and there is a significant price difference. For example, the estimation course costs \$100; an excellent book called *Estimating Software-Intensive Systems* provides almost 1000 pages of material for \$60. Audio and video can have advantages over print in presentation and understanding, but these courses do not embrace that advantage.

Reviews

SE204: Measuring the Size of Software Products

For individuals planning to manage software development, some means of estimating and measuring the scope of the effort is essential. The QAI course on measuring the size of software products is an excellent introduction to the problem and current approaches to solving it.

The course begins with a short seven-panel “movie,” which is a sequence of drawings illustrating a dialogue between individuals discussing the problem of how to estimate how much work it is going to take to accomplish a software development. As one might expect there are at least two elements to the problem: the size of the product to be produced and the productivity of the team doing the work. But the problem is more complex than that as the course immediately illustrates.

The fact that quantity of code doesn’t tell anything about the quality of code is one focus. Indeed the quantity of code itself doesn’t say much about the functionality. Traditional measures such as lines of code (LOC) come in for inspection and analysis. What about defects and defect density? What about the expressivity of different languages? All lines of code are not created equal.

Enter function points (FP) as an alternative. FPs are explained as an alternative to LOC, which in any case is an observable only after the product is developed. The student learns how to estimate FPs and how to retrospectively estimate the productivity in FPs of past projects for which LOC measurements are available by using the backfiring technique developed by Capers Jones. An incidental benefit of the course is that the student is exposed to data on the range of productivity offered by different computer programming languages.

Examples help tie the course to reality, and a thread of examples gives the student practical insight into the complexities and methods of resolving them inherent in estimating the size and scope of software products. Only experience in the real world will hone the student’s ability to use the technique he or she has been exposed to, but the course will be an eye-opener for those who have not thought about these problems. I think it is worthwhile, and I’ve used these techniques and taught them in the past. The course effectively presents a complicated topic in an insightful and effective manner.

SE205: Outsourcing Project Work

No specific comments.

SE206: Risk Management

The risk management chapter of Pressman’s book is 17 pages long. The basic information seems to be the same as the material in the course. If one carefully reads the chapter it would take about the same length of time as the course. The book also links to a Web site that has much more information.

Some of the information is good, but it wouldn’t take novices to the point where they could do risk management—just gives an overview of the concepts.

The order of the material is wrong. Risk identification is the last section before closing the scenario, but the background material it provides is needed in the Risk Model section.

The opening and closing scenarios are pretty hokey; a movie rather than cartoons would have been more tolerable.

SQM (Mod 7)

This module, Software Quality Management, is 21 hours long and contains the following sessions:

- SE301: Basic Concepts of Software Quality (five hours)
- SE302: Software Quality Assurance (nine hours)
- SE303: Formal Technical Reviews (seven hours)

SE301: Basic Concepts of Software Quality

Basic Concepts of Software Quality is an online course on the definition and measurement of quality for software. It emphasizes that meeting cost, schedule, and quality expectations are essential to making the customer happy. The material explains McCall’s factor, criteria, and metric framework (FCM). The one measure that is shown in detail is defect removal efficiency, which is errors/(errors + defects). Errors are problems found before delivery; defects are found after delivery.

The name of the course is “basic concepts” and they are pretty basic. It was supposed to take five hours, but there are only 80 screens, so it will typically take much less time than that. It is a very high-level view and only mentions a handful of quality measures. The student will have to obtain other instruction to really know how to use the techniques. Even the defect removal efficiency requires knowledge of defects for the first year after production, so it is not a predictor of quality. There are no prerequisites for this course.

SE302: Software Quality Assurance

This course covers basic software quality assurance (QA) concepts. It covers the rationale behind software QA and mapping verification and validation activities to the specific life cycle chosen for a project or organization. A valuable section of this course deals with the costs of software QA, and in following the themes of Pressman’s text, an important reference is made between fundamental quality principles and software QA.

The content covered the topic area. The course covers many topics that touch on the central topic, such as the definition(s) of software quality, software project planning, and software life-cycle processes. The topic groupings and related topics in Pressman’s book for the subjects are all covered.

Reviews

SE303: Formal Technical Reviews

This covers the distinction between the various forms of reviews, both formal and informal, and it reinforces the rationale for formal reviews with a (possibly duplicated but strong) discussion on the relative costs of fixing errors downstream.

This course, and the previous, provides a few sidebar discussions found in Pressman's book. For example, there is some specific detail about Fagan inspections. There are hyperlinks to some of Fagan's key papers. In general, there are follow-up references for both courses, but one would certainly expect this in a course associated with Pressman. In fact, Pressman's

own site is presented as a reference as well.

The course covers the relatively small topic area in very good depth. The topic groupings and related topics in Pressman's book for the subjects are all covered.

Overall, both the formal technical reviews and software QA courses offer value to beginning software practitioners. Learners from this audience will benefit from some repetition in basic software QA concepts. In addition, by working through the course quizzes, learners have to think through the concepts. Prospective learners may consider that the content is frequently aligned with the contents of Pressman's textbook. And while the QAI

course designers did not add a tremendous amount of content, they have formulated a step-by-step course. This makes the decision between online course and textbook quite straightforward - those needing this kind of instruction and repetition will indeed benefit from the online course format.

Summary

The overall impression of the program is not favorable. The application has problems that should be addressed before considering a release to a paying public. The content has a long list of problems that must be addressed if this application is to be sold to a technology-minded market.