

Resource Reviews

GENERAL KNOWLEDGE

CSQE Primer

Barbara Frank, Phil Marriott, and Chett Warzusen. 2002. Quality Council of Indiana (<http://www.qualitycouncil.com>). 734 pages. CSQE Body of Knowledge areas: All

*Reviewed by Eva Freund
eva.freund@ivvgroup.com*

As one of the early takers of the CSQE examination, I entered the examination room with a notebook stuffed with copies of articles and extracts from various publications. These were supplemented with notes taken at a study session organized by a colleague. Looking around the room I saw that some individuals had wheeled carts filled with reference books. I wondered if they really expected to have the time to search for the correct answer to a difficult question.

The *CSQE Primer* contains everything I had and much more. The material is organized to follow the CSQE Body of Knowledge (BOK). The layout of each page makes the material easy to read and easy to follow. The ample margins provide space for personal notes. The authors make no assumptions about the reader's prior knowledge and, thus, everything is defined, explained, and illustrated.

Each section of the primer addresses a different section of the CSQE BOK. Each chapter contains descriptive information on the identified models, techniques, and processes. This is followed by a list of recommended readings and multiple-choice questions (answers are contained in the index). One

reader may choose to answer the questions in order to determine which sections need further study. Another reader may review the material and then answer the questions to determine his or her own level of knowledge and comprehension. As stated in the preface, "This text is designed to provide...a review of fundamental knowledge and skills. It is also...a primer for those interested in taking the certification examination."

This primer has several strengths. These include:

- It identifies the authors of the many definitions, checklists, and templates.
- It contains prolific references to consensus standards.
- It identifies the cognitive level at which the sample questions will be written.

The primer would be stronger if the authors had used IEEE Standard 1012, *Software Verification and Validation*, rather than 1059, which was a guide to software verification and validation (V&V) and was withdrawn several years ago. Had they used IEEE Std. 1012-1998, the primer might have included the concept of integrity levels. This concept states that the breadth and depth of V&V tasks is determined by the level of impact of a software or system failure. The primer also would have described the criteria for evaluating documentation such as requirements and design documents as well as for doing requirements traceability.

If I were taking the CSQE examination tomorrow, I would take this book as well as *Software Engineering: A Practitioner's Approach* by Roger S. Pressman.

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.

Great Software Debates

Alan M. Davis. 2004. IEEE CS Press (<http://www.computer.org/cspress/>). 274 pages. ISBN: 0-471-67523-7.

CSQE Body of Knowledge areas: General Knowledge, All

*Reviewed by Scott Duncan
sduncan@tsys.com*

This is a collection of essays, mostly previously published, designed "to make you think about the things that you may have taken for granted up to now about software." Davis, who has a long list of credentials including editor and co-columnist for *IEEE Software*, feels "the industry is making relatively little progress with respect to its practices." He asks readers not to necessarily agree with his positions, but to use them as a starting point "to take an active role in making things change for the better in the software industry."

There are 38 essays divided into five sections:

- "The Software Industry" (10 essays) "explores various aspects of the software industry as a whole."
- "Management" (nine essays) is based on the author's experience "as a manager in various high-tech companies" where he has spent most of his career.
- "Requirements" (eight essays) takes a "much broader view" of

the topic than "something that is documented in a requirements specification."

- "Software Research and Academe" (five essays) draws not from his own work but from "looking in" on what others consider "leading-edge research."
- "Life and Software" (six essays) applies "some principles, wisdom, and experience from life in general" to software.

Each section begins with a brief introduction. At the end of each essay, however, are questions that the author offers to generate thinking in college classes, professional seminars, and workplace discussions.

A sixth section, "The Future," written in 1998, is really a 39th essay about such a growing divergence in practice, standards, and beliefs that Davis "expects, by 2010, we will no longer consider the software industry to be an industry." The first essay, written in 1993, describes the "lemming paths" – fads some might say – that have affected the software industry, and still do, which Davis takes a devil's advocate examination of, questioning what real value they have contributed. These first and last essays seem to nicely bookend the others and allow readers to consider if the author's views still hold up today.

Of the two essays not published elsewhere, one is a look at a start-up company of which Davis was one of the founders, telling the story of its demise and drawing a variety of "lessons learned" about starting one's own (software) company. The other, developed during a three-week "voyage" through Namibia and Botswana, is a consideration of technology haves and have nots and what "progress" means relative to the environment in which one finds himself.

I like books that expect that the reader will think about and react to the content in some substantive

way. A few examples of this in the book are:

- *An essay on software as art vs. engineering.* Davis comes down squarely on neither side, comparing software to customer home design/construction in which "it is recognized that different skills are required at different times." The essay suggests the need to consider implied skills and training for software professionals and questions the common view (hope) that a software developer can do all these tasks reasonably well as if an architect and carpenter were interchangeable roles.
- *An essay on why people develop software and how they measure its "goodness."* After fewness of defects and profitability, Davis suggests "contribution to humanity and/or society" as a measure that is needed. This reminded me of an ACM *Ubiquity* (issue 39, December 4–10, 2001) interview with Herb Grosch, who commented on T. J. Watson, Sr.'s view of a company's duty to "pay back" to "the broader society" for "the privileges extended to" it.

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.

SOFTWARE QUALITY MANAGEMENT

The Requirements Engineering Handbook

Ralph R. Young. 2003. Artech House (<http://www.artech-house.com>). 254 pages. ISBN: 1-58053-266-7.

CSQE Body of Knowledge areas: Software Quality Management

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

The Requirements Engineering Handbook is a follow-up to the author's book, *Effective Requirements Practices*. It is a collection of lists and suggestions on how requirements analysts (RAs) perform their work. Since it is a handbook and not a textbook, it has references to other resources rather than details on how to perform each step. It also includes "war stories" and case studies about problems when there were no requirements or poorly written ones.

The author says that a requirement is a "statement that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility for a user." *The Requirements Engineering Handbook* has 10 chapters; they include information on the importance of requirements, the roles of RAs, skills needed by an RA, the types of requirements, gathering requirements, managing requirement, and an integrated quality approach. The author's recommended strategy is to: 1) write a requirements plan, 2) design a requirements process, 3) invest in requirements activities in the life cycle, and 4) utilize effective practices, methods, tools, and training. He says that a project should invest 8 to 14 percent of the cost or effort in requirements; however, 2 to 3 percent is typical.

If an organization is following the requirements management or requirements development process area of the Software Engineering Institute's Capability Maturity Model Integration (CMMI), this book will provide a good explanation of what needs to be done, as well

Reviews

as information on how to do it. Even if one is not following a particular model, there is valuable guidance on best practices. Examples are the 28 steps for gathering requirements and the 11 most effective requirements elicitation techniques. However, there is not enough detail in the book to actually perform an elicitation technique or a step; instead, there are references to Web sites and/or books.

The Requirements Engineering Handbook is good for what it is: a handbook and pointer to requirements resources. It is meant to be a reference rather than read from cover to cover. If the reader already knows something about the subject and wants to learn more about a topic, the handbook is useful. If the reader tries to use it as an introductory text, he or she will be frustrated by having to go to other books to find the details. It is also repetitive if read straight through, although the author provides many good tips and checklists if it is used as a reference. It would be a good addition to the bookshelf for a project manager, process engineer, requirements manager, or requirements analyst.

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

Requirements by Collaboration

Ellen Gottesdiener. 2002. Addison-Wesley (<http://www.awprofessional.com>). 339 pages. ISBN: 0-201-78606-0. CSQE Body of Knowledge area: Software Quality Management

Reviewed by Scott Duncan
sduncan@tsys.com

The book's preface states that it "explains how to plan and hold workshops to meet two essential needs: efficiently defining user requirements while building positive, productive working relationships." It goes on to say that it "focuses on the essential tools you need for planning and leading requirements workshops. It integrates user requirements modeling, including use cases, business rules, and collaborative techniques." So the book is about "facilitating a requirements workshop" more than it is about the technical aspects of requirements definition and validation. Indeed, as it says later in the book, the techniques and approaches described could be used to facilitate almost any kind of workshop. Even the material on modeling, though related specifically to requirements, could be used in contexts other than a workshop.

Having attended a workshop given by Gottesdiener on the subject of facilitation at the Agile Development Conference last June in Salt Lake City, I can say that the advice related to facilitation is quite good. Of course, seeing someone facilitate a workshop as a way of demonstrating the kind of advice in this book is more effective than reading about it. I can say, however, that the ideas and advice in the book can work well in practice. Indeed, in Chapter 1, Gottesdiener notes how approaches in the book "are a modern-day variant of joint application design, also known as joint application development."

To cover the material, the book is divided into three parts:

- Three chapters of "Overview," which provide a "high-level overview of the various models that are the primary deliverables of a requirements workshop" and describe "the elements you need

to achieve success with a requirements workshop."

- Six chapters of "Framework," which discuss having a common goal, what roles people play, workshop ground rules, workshop products, meeting place/time logistics, and how the workshop should flow.
- Three chapters of "Design Strategies," which cover "navigation" (for example, breadth-first, depth-first), some case studies, and making "workshops a best practice."

The topic of quality assurance, though, is dealt with in about four pages that list some "tools" to apply to check the quality of the already elicited requirements (for example, checklists, walkthroughs, matrices). So this is not a book about requirements traceability, testability, and so on. It does, however, do a good job of facilitating workshops with requirements elicitation as its major application example.

SOFTWARE ENGINEERING PROCESS

Java Design: Objects, UML, and Process

Kirk Knoernschild. 2002. Addison-Wesley (<http://www.awprofessional.com>). 277 pages. ISBN: 0-201-75044-9.

CSQE Body of Knowledge areas: Software Engineering Processes: Analysis, Design, and Development Methods and Tools

Design Patterns Java Workbook

Steven John Metsker. 2002. Addison-Wesley

(<http://www.awprofessional.com>). 475 pages. ISBN: 0-201-74397-3.

C# Design Patterns: A Tutorial

James W. Cooper. 2003. Addison-Wesley (<http://www.awprofessional.com>). 392 pages. ISBN: 0-201-84453-2.

Reviewed by Ray Schneider
rschneid@bridgewater.edu

It's hard to believe that almost 10 years have passed since the publication of *Design Patterns* by the "gang of four" (GOF)—Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Christopher Alexander had written books on architectural patterns much earlier: *A Pattern Language* (1977) and *Timeless Way of Building* (1979). These works inspired a surprising group—software developers. It led to a boiling, bubbling froth of activity culminating in, among other things, the GOF book and Christopher Alexander, an architect, speaking to people he had never dreamed of inspiring: a huge room filled with software people at OOPSLA '96.

Patterns exist at all levels. Alexander's books dealt with different levels of granularity, more fractal-like in the sense of self-similarity than hierarchical. His patterns could apply to cities, towns, communities, neighborhoods, and dwellings, often expressing similar ideas at different scales. The three books in this review are largely focused on both the Unified Modeling Language (UML) and the programming patterns of the GOF book. All use UML or UML-like diagrams. Two focus on Java and the third on C#. All promise insight into design and offer the reader a means of becoming more skilled as a designer and thereby take programming to a higher, more professional level.

Knoernschild's book, *Java Design Objects, UML, and Process* approaches the issue of design at the highest level. It should be no surprise that as soon as readers enter the book they find themselves in a decidedly object-oriented universe. This is not only because the book features Java, but also because the book sets out to be object-centric. The objective as Knoernschild states it in the preface is to "... give developers a deeper understanding of how to design cleaner Java applications." A key theme is *convergence* on issues of software design, especially the feature creep and constant demand for enhancement and change without encountering what Paul Bassett calls "brittle software." Knoernschild's focus is on best practices. Not only is the book convergent but broadly integrative, pulling together Java, objects, UML, process, patterns, and principles.

The first chapter is a high-level statement of principles and patterns in a concise and eloquent manifesto that puts principles before patterns. The usefulness and proliferation of pattern literature has obscured the fact that using them effectively requires principles. It simply isn't automatic. Knoernschild divides his work into two broad parts. The first four chapters cover basics of UML, object-orientation, and software process taken independently, but in a cumulative fashion. The next seven chapters converge more narrowly on the broad themes sounded earlier in the book.

Up front, Knoernschild says that programming only accounts for 15 percent of the time spent developing software. The other 85 percent is mostly design. Chapter 2 introduces UML in the context of the importance of modeling and architecture in design. Then drilling down, chapter 3 provides the fundamentals of UML. Readers might want to go elsewhere for UML in depth, but this is a good start.

Chapter 4 is one of my favorites because here best practices and the software development life cycle are both explored. The best practices Knoernschild is interested in are those that contribute most to the requirements gathering and design processes. He focuses on six:

1. Behavioral-driven requirements, explored in chapter 8
2. Architecture centric design, with its many design decisions explored in chapters 7, 8, and 9 with chapter 10 specifically exploring the modeling of architectural mechanisms
3. Iterative development, which uses the spiral model or a prototype enhancement model
4. A commitment to refactoring
5. Visual modeling to obtain feedback from peers and mentors
6. Simple prototypes used in a throwaway fashion early in the design cycle

As I read this book I felt constantly validated, hearing in my head phrases like "right-on" and "you said it." The remaining seven chapters revisit the themes already established, modeling applied to requirements, behaviors, structural composition, architecture, and subsystem design.

This is not a standalone book. It doesn't drill down far enough, but it shouldn't. It is more a book on the wider themes, a motivational book, showing why and how one ought to drill down and integrate and converge. It is invaluable because of that point of view and how well it expresses it. I think it needs to be joined by other books, which is why I did a group review.

The second book is Steven John Metsker's *Design Patterns Java Workbook*. One can't learn how to program without programming, and one can't learn about programming patterns without doing code to illustrate and in that sense *feel* what the patterns are really all about. Seeing and doing are very

Reviews

different things. Metsker's introductory chapter starts off with five whys? Why patterns? Why design patterns? Why Java? Why UML? Why a workbook? These are all elements in Knoernschild's book. The foreword starts with a thematic quotation from Benjamin Franklin, a man constantly focused on personal improvement. "Tell me and I forget. Teach me and I remember. Involve me and I learn." That's what the workbook is all about.

Both *Design Patterns Java Workbook* and James W. Cooper's *C# Design Patterns: A Tutorial* focus primarily on patterns. Both cover every pattern in the GOF book with code examples. Code is provided by both books, Cooper's on a CD and Metsker's in the appendices and at his Web site www.oozinoz.com.

One aspect of the workbook I found interesting was the refactoring of the pattern categories. The GOF classified their 23 patterns into three categories: creational, structural, and behavioral. Metsker provides the same 23 patterns but in five categories: interface patterns, responsibility patterns, construction patterns, operation patterns, and extension patterns. These titles are the chapter headings of the book. Each pattern is presented and an illustration of its use is shown in the context of a hypothetical fire-works company called Oozinoz. Sometimes the examples are strained, but that doesn't really matter. Each pattern includes challenge questions, UML diagrams, a small programming challenge, and a summary. Generally, the illustrations in Java use the awt and swing libraries. However, this is not primarily about Java. It is primarily about design patterns with Java as a vehicle and UML as an illustrative aid. Working through this workbook will give readers a good feel for patterns. It seems well suited to a semester course in patterns or a self-study.

C# Design Patterns: A Tutorial is a straightforward presentation of C# followed by a systematic plow through the 23 GOF patterns in almost exactly the same order they are presented by the GOF. The book has lots of code and illustrations, both screen shots and UML diagrams. The programs in the book were written using Visual Studio .NET, and the CD-ROM that accompanies the book contains the project files for each project. The C# code doesn't utilize the unique features of C#, having more the cast of a Java-like translation, but I'm not sure that is really a defect. It is often helpful to avoid language-unique features in code that is intended to be transportable. The book is mainly aimed at relative novices and will be helpful in that context. It does give as much insight as the other two books into patterns, but it is a nice introduction to a Java-flavored C# with a pattern emphasis—which is no mean achievement.

The punch line is that all three of these books are worth reading. One problem with the way the pattern movement has solidified and concretized the GOF patterns is that they are seen more as abstract modules, and that is not actually what patterns are about. To understand patterns one needs, like Christopher Alexander illustrates, to see things in fractal-like levels. Knoernschild provides a wider context for patterns, but doesn't treat the GOF patterns in their entirety. The books by Metsker and Cooper drill down to all the patterns, Metsker somewhat more successfully than Cooper, but that judgment will depend on the reader's learning style and willingness to recognize that all examples in books tend to be toys.

What is really important is to see that one book is not enough. Each book casts its tentacles out and intertwines with the others. It

is essential to cast one's net wider and also get one's feet wet in order to catch those octopi, uh patterns.

Ray Schneider holds a bachelor's degree in physics, a master's degree in engineering science, and a doctorate in information technology. He is a licensed professional engineer in the state of Virginia.

Software Development for Small Teams: A RUP-Centric Approach

Gary Pollice, Liz Augustine, Chris Lowe, and Jas Madhur.

2004. Addison-Wesley (<http://www.awprofessional.com>). 272 pages.

ISBN: 0-321-19950-2.

CSQE Body of Knowledge area: Software Engineering Process

Reviewed by Joel Glazer
joelglazer@ieee.org

The preface of this book states, "This book chronicles the authors experiences with developing a working software project, 'PSP Tools.' The goal of the 'PSP Tools' is to provide automated support for Watts Humphrey's Personal Software Process (PSP). The book contains screen shot, tables, and other snippets showing progress, as well as the final software, ...to show how the team worked, rather than an idealized interpretation of how the team wished they had worked."

The book breaks down the notion that small teams "don't need no stinkin' process." Typical industry teams consist of members working at the same company and usually location, selected from a pool of available software engineers, drawn together by a project manager assigned by a senior member of the company, given a charge number and, hopefully, clear requirements, objectives, and a reasonable schedule to meet. The team in this book is

not the typical project team found in industry. Team members had day jobs at different companies and volunteered their time and effort in a moonlighting fashion. This team was assembled based on personal knowledge and acquaintance, and team members' time was not "clocked" in the usual manner by project management.

The book contains 11 chapters, three appendices, as well as several links to useful Web sites. Chapter 1 introduces the team members, and chapter 2 defines "small" projects and the relevance of a process such as Rational Unified Process (RUP) to such a project. In chapter 3, "People, Process and Tools," the authors establish 14 project guidelines to creating a working project team. They are:

1. Get the right mix of people on your project. (Staff your project with people possessing different skills and experience so they complement each other.)
2. Provide a learning environment. (Make sure that every team member has the opportunity to learn on the project. Things to learn can be technical, organizational, or managerial.)
3. Generate trust. (Help the team members trust and respect each other.)
4. Allow disagreement. (Provide a way to resolve disagreements.)
5. Ask, don't demand. (Make requests, not demands.)
6. Recognize achievement. (Recognize achievements – sincerely and often.)
7. The prime directive of process. (Only do those activities and produce those artifacts that directly lead to delivering value to your customers and stakeholders.)
8. Have a risk-driven process. (Make sure you perform activities and produce artifacts that reduce risk.)

9. Start with a proven foundation. (Base your process on proven practices, techniques, and principles.)
10. Configure your process. (Tailor your process for every project. Don't assume that every aspect of a process will automatically apply to your situation.)
11. Apply common sense liberally. (Having a process to guide you in your development efforts does *not* absolve you from using your brain and applying common sense.)
12. Make sure tools are worth the effort. (Ensure that your tools support the process and people on your project.)
13. Provide time to learn tool usage. (Provide instruction on tool usage and schedule the learning time into your project.)
14. Build tools when necessary. (Build your own tools when necessary, but make sure that the benefits outweigh the cost.)

Chapter 4 details how a collection of individuals working on a project becomes a team, including the obstacles that had to be overcome, such as geographic dispersion and turnover, progress reporting, conventions to use, and artifacts to keep. A critical aspect of keeping artifacts is the formality level used for configuration control.

Chapters 5 through 10 detail the development efforts following the RUP phases of inception, elaboration, construction, and transition, and what the team did in each phase. Chapters 7 and 9 go into technical details about code and technology used.

Chapter 11 describes what the team learned from this experiment and what it might do differently next time. It is the "Post-Mortem: How Will We Improve the Next Version" and should be read by all who lead projects. The authors detail the benefits, goals, conduct,

and self-critique of the good, bad, and ugly.

In summary, regardless of project size, all project leaders who must balance customer satisfaction, schedule, cost, and delivering working products should read this book. It is a useful addition to any software group's library.

Joel Glazer has more than 30 years' experience in the aerospace engineering, software engineering, and software quality fields. He has dual master's degrees from The Johns Hopkins University in computer sciences and in management sciences.

Enterprise Patterns and MDA

Jim Arlow and Ila Neustadt.

2003. Addison-Wesley

(<http://www.awprofessional.com>). 507 pages.

ISBN: 0-321-11230-X.

CSQE Body of Knowledge area: Software Engineering Process

*Reviewed by Scott Duncan
sduncan@tsys.com*

Not surprisingly, given the title, the introduction says this book is about "a set of essential patterns for enterprise computing," which "are not technical 'design patterns'; rather they are essential business patterns..." and "how you can use the emerging discipline of Model Driven Architecture (MDA) to apply these patterns with a high degree of automation." Further, Unified Modeling Language (UML) is used, embedded "in a narrative such that the patterns can be understood, validated, and adapted even by nontechnical readers." The authors say the time is ripe for this because of improvements in modeling and automated tool support for UML modeling.

The authors identify "four main threads":

Reviews

- "The theory of archetypes and archetype patterns" defines "archetype" and "pattern" and shows how to generate a UML profile to address modeling them and, if briefly, how models can be used to address architectural quality, that is, how components of the system fit together.
- "Pattern automation using MDA" provides a short introduction to MDA and how archetype patterns are automated using an "MDA-enabled modeling tool." The concept of "pattern configuration rules" is introduced, stating that a PCR is "a series of statements of truth about what combinations of pattern features constitute well-formed configurations of a specific pattern." (The authors would have liked to use the object constraint language (OCL) in UML for defining such rules but claim the support they need isn't currently present in OCL, so they define a pattern configuration language (PCL).
- Making UML models "accessible to a wide audience through literate modeling" by overcoming the limitations of having to know the formalities of visual models. The answer, according to the authors, is "a narrative description that is accessible to many different readers" besides those who know UML icons, syntax, semantics, and so on. This "thread" discusses the "comprehensibility and accessibility" of the various UML models and how to apply "literate modeling" to them where possible.
- "A valuable pattern catalog" of some 340-plus pages, comprises the remaining chapters of the book. The archetype pattern catalog describes and defines nine patterns: party, party relationship, customer relationship management,

product, inventory, order, quantity, money, and rule.

The target audience for the book, according to the authors' own table for how to use the book based on goals in reading it, are OO analysts/designers, OO programmers, architects, business analysts, project managers, and software engineers. Thus, other than to build an understanding of this pattern approach, literate modeling, and using MDA, there is only tangential relationship to the CSQE BOK, probably in the area of assessing design quality expressed in such terms.

PROJECT AND PROCESS MANAGEMENT

Effective Risk Management: Some Keys to Success (2nd Edition).

Edmund H. Conrow. 2003. American Institute of Aeronautics and Astronautics, Inc. (<http://www.aiaa.org>). 526 pages. ISBN: 1-56347-581-2. CSQE Body of Knowledge areas: Program and Project Management; Risk Management.

Reviewed by John Richards

"The purpose of this book is two-fold: first, to provide key lessons learned that I have documented from performing risk management on a wide variety of programs, and second, to assist you, the reader, in developing and implementing an effective risk management process on your program." Conrow, p. xvii

The lessons learned alluded to include more than 700 tips to success and traps to avoid that the author, Edmund Conrow, has collected from a 20-year plus career in risk management. The majority of his experience has been

on government programs working with the U.S. Air Force, U.S. Army, U.S. Navy, the Department of Defense (DoD), and the National Aeronautics and Space Administration (NASA), as well as other governmental and commercial agencies. He has worked on programs with a life-cycle cost ranging from \$20 million to more than \$50 billion. It should be noted that the risk discussed is project management risk and not business or financial risk. However, project management clearly impacts these two.

The book consists of eight chapters and 11 appendices. The chapter topics are:

1. Introduction and Need for Risk Management
2. Risk Management Overview
3. Some Risk Management Implementation Considerations
4. Some Risk Planning Considerations
5. Some Risk Identification Considerations
6. Some Risk Analysis Considerations
7. Some Risk Handling Considerations
8. Some Risk Monitoring Considerations

The appendices topics are:

- A. Comparison of Risk Management for Commercial and Defense Programs
- B. Current DoD Program Phases and Milestones
- C. Comparison of the Microeconomic Framework to Actual DoD Program Outcomes
- D. Some Characteristics of the Cost-Performance Slice of the Technical Possibility Surface
- E. Changing the Definition of Risk – Why Risk It?
- F. Program Structure Models
- G. Sample Language for Including Risk Management in an RFP and Evaluating It During Source Selection

- H. Some Characteristics and Limitations of Ordinal Scales in Risk Analysis
- I. Example Technical Risk Analysis
- J. Estimative Probability Values for Subjective Probability Statements
- K. Development of Ordinal Probability Scales from Cardinal Data

This is a very comprehensive and detailed volume. The first two chapters are a good introduction but a little basic for the typical reader. The rest of the book is extremely useful and detailed. This second edition contains a new appendix written by another well-known risk management consultant and authority, Dr. Robert N. Charette, on the role of opportunity in the definition of risk. This book is essential for individuals doing project management work, especially on DoD projects, to include the preparation of the proposal in response to the RFP to win the work.

John Richards is a principal member of the professional staff in the Defense Information Services Sector of SRA International. He has more than 30 years of proven accomplishments in a variety of leadership, research, and teaching positions.

Hiring the Best Knowledge Workers, Techies & Nerds

Johanna Rothman. 2004.
Dorset House
(<http://www.dorsethouse.com>).
342 pages.

ISBN: 0-932633-59-5.

CSQE Body of Knowledge
area: Project Management

*Reviewed by Scott Duncan
sduncan@tsys.com*

"Hiring technical people...is vastly different from hiring purely skill-based staff" because it is not just "what they know but how they apply that knowledge." With these

statements, the preface identifies the major issue this book seeks to address: how to find, attract, interview, and interest "qualified technical workers." Though aimed at managers, Gerry Weinberg notes that the book should be of interest to "everyone who participates in the hiring process," and I would agree.

Rothman lists the following as what she "hope[s]" this book can do for every hiring manager who uses it as [she] intended it:

- Save you time and money every time you hire.
- Help you hire people who can perform the required work well.
- Help you screen, evaluate, and hire the right staff for your specific organization.
- Eliminate the wasted time and suffering that result from having to fire people who should not have been hired in the first place.
- Help you develop and demonstrate fundamental management competency."

The book is divided into five parts with two to four chapters each:

- "Defining Requirements for Yourself and Your Candidates," covering hiring strategy, job analysis and job descriptions. Starting out on the right path is critical as one can hardly hope to hire the "right people" if he or she doesn't have decent ideas/descriptions about what would be "right." Rothman has a step-by-step approach to this, even indicating the amount of time to spend on the tasks involved.
- "Sourcing and Selecting Candidates to Interview," covering recruiting, ads, and reviewing resumes. Job ads are often written as laundry lists of technologies/skills, giving little idea of how people would be expected to work. Rothman provides many examples of the

kinds of language to use and how to emphasize the company, the technology, and so on based on what one feels would be selling points. The material on resumes is particularly useful and can help people writing, as well as reviewing, them.

- "Preparing to Interview Candidates," covering interview questions and techniques, phone-screening, in-person interviews, and interview follow-up. Often, it can seem like an interviewer isn't sure what he or she is looking for or could have cleared it up over the phone ahead of time. Rothman covers this crucial aspect of the hiring process very well, including specific suggestions for what to explore with developers, testers, senior staff, project managers, and so on. Using this part of the book alone should help immensely in getting the people one needs and interesting them in the job.
- "Bringing in the Candidate," covering checking references and making offers. Rothman describes how to make use of references and emphasizes using all of them. As in other parts of the book, there are specific examples given with explanations for how and why they are useful.
- "Making the Most of Hiring Opportunities to Control Uncertainty and Risk," covering the first day on the job, hiring technical managers, and dealing with issues "when no one seems right" and the need is critical. Hiring the "best" is only part of the story—one has to keep them and Rothman covers making sure that, when that "best" person arrives, his or her first day reinforces the decision to work for the organization. She also discusses adapting the advice in the book to the specific case of hiring a

Reviews

manager who will impact several, if not many, other "best" people, very directly. Finally, she addresses what to try when the "right" person just doesn't seem to be there.

Two appendices cover: a) a fictional case study in staffing up a growing organization when multiple people have to be hired, and b) a collection of the "worksheets and tables" used in the book in "generic" form rather than with the sample content filled in.

As Weinberg says, anybody involved in the process of recruiting and hiring technical people can make use of the book, including the technical candidates themselves. And, at any time, any of us could be a part of that process, or could influence it in some way. This is a book all people could read and find useful if they want to improve their own company's hiring practices or, at least, their part of it.

OTHER BOOKS

The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad
Michael Cusumano. 2004.
Free Press. 352 pages.
ISBN: 074321580X.
CSQE Body of Knowledge areas: None

*Reviewed by Noreen Dertinger
Noreen.Dertinger@cognos.com*

The Business of Software is aimed at practitioners who are interested in the strategies that contribute to the successful running of a software company. The book will not add to the specialized body of knowledge even a novice software quality professional should have. The focus of the book is on software development strategies and as such it does include a discussion on

the importance of software testing. This may appeal to software quality professionals who wish to expand their understanding of the place of quality software engineering in the overall software development cycle. Readers looking for new insights on and new aspects of software quality practices should look elsewhere.

The book contains a wealth of information, based on real case studies and the author's personal experience, about leading software companies, including startups, through bad and good economic times. Chapter 4, "Best Practices in Software Development," does have a short section on "Testing and QA" where Cusumano acknowledges that testing and quality assurance is an essential part of the development process to become a successful company. He supports this with a brief presentation of Microsoft's quality practice. Overall, I did not think this book added any particular information toward the areas defined by the ASQ CSQE Body of Knowledge.

Noreen Dertinger (Noreen.Dertinger@cognos.com) earned a master's degree from the University of Ottawa, a certificate in information technology from the University of Victoria, and completed her CMII certification. She has 15 years' experience in the software industry in development, configuration management, and quality assurance. Dertinger is a software quality control analyst with Cognos in Ottawa, Canada, working on Cognos ReportNet.

Software Architect Bootcamp
Raphael Malveau and Thomas J. Mowbray. 2004. Prentice Hall (<http://phptr.com>).
353 pages.
ISBN: 0-13-141227-2.
CSQE Body of Knowledge area: None

*Reviewed by Scott Duncan
sduncan@tsys.com*

The book is "about achieving and maintaining success in your software career," that is, of being a software architect. It is also "about an important new software discipline and technology, software architecture." Thus, it is not specifically a book about software architecture, but about career preparation for someone wanting to be an architect. It's designed to help someone "make the successful transition from software developer to software architect." Thus, besides material about software architecture(s), there are chapters about leadership, managing expectations, career advice, process design, and teamwork.

After a brief introduction, the second chapter addresses "major schools of software architecture thought," which the authors classify as:

- Zachman framework (a traditional approach)
- Open distributed processing (from the ISO/ITU communities and to which a lot of space is devoted)
- Domain analysis ("a process for systematic management of software reuse")
- 4+1 view model (from the Rational Unified Process)
- Academic software architecture (which "avoids proven architectural standards and practices in order to achieve originality...").

The third chapter discusses OO and component-oriented design as well as client-server, distributed, and Web-based architectures. The fourth covers a very high-level approach to "doing software correctly" (that is, a high-level software/architecture development life cycle). The fifth quickly covers complexity, system integration, making the business case, and how software architecture relates to software development. Another covers UML notation and modeling,

while yet another addresses "gathering knowledge" (which includes conducting reviews and walkthroughs). Others address the "career guidance" aspects mentioned earlier. And the subject of quality engineering as it relates to architecture is not overtly discussed.

I cannot recommend this book. It skips across many topics at a high level. There are better books that cover most of the material in this book more effectively. I found the book a somewhat dissatisfying summary of the material.