

# Resource Reviews

## SOFTWARE QUALITY MANAGEMENT

### Information Systems: Achieving Success by Avoiding Failure

Joyce Fortune and Geoff Peters.

2006. John Wiley & Sons Ltd (<http://www.wiley.com>).

234 pages.

ISBN: 0-470-86255-6

CSQE Body of Knowledge areas: Software Quality Management; Program and Project Management

*Reviewed by Noreen Dertinger*  
[noreen.dertinger@cognos.com](mailto:noreen.dertinger@cognos.com)

Information systems are an important component of many businesses today. A major systems failure could have serious consequences for business, ranging from loss of revenue up to and including threatening the viability of the business and even, in the case of more catastrophic failures, loss of life. Information systems, however, continue to be prone to failure, and the same types of problems tend to be repeated over and over again.

In *Information Systems: Achieving Success by Avoiding Failure* Joyce Fortune and Geoff Peters strive to help the reader avoid failure by bringing to light lessons learned and making those involved more apt to share what they learned from failures. The authors' goal is to provide a well-tested approach to analyzing the failures in a manner that one can understand and learn from.

The authors, U.K.-based academics specializing in research of

systems failures, begin with an overview based mostly on British case studies. They define success as:

"The system achieved what was intended of it; it was operational at the time and cost that were planned; the project team and the users are pleased with the result and they continue to be satisfied afterwards."

More detailed case studies make up the body of the book. While some case studies illustrate successful projects, most illustrate projects that failed. Most important, each case is accompanied by an analysis of the issues that contribute to the success or failure.

*Information Systems: Achieving Success by Avoiding Failure* is a concise handbook for anyone interested in successful systems initiatives, but especially for managers responsible for development and quality control. While many of the references are British, the lessons learned and the various tables and figures can provide readers anywhere with fundamental criteria that can be customized and applied according to the nature of their specific business needs.

The material is clearly written and presented, making it accessible to individuals at any level of systems development or quality teams. Any systems person should be able to glean useful information from the material that will assist him or her in planning and testing systems. This is a "must read" handbook for any systems manager.

Noreen Dertinger is a software quality analyst with Cognos Inc. (Canada). She has 17 years of experience in the IT field. During her career she has worked on software development and configuration management, and she currently works on software testing.

### Professional Issues in Information Technology

Frank Bott.

2005. British Computer Society (<http://www.bcs.org>). 249 pages.

ISBN: 1-902505-65-4

CSQE Body of Knowledge area: Software Quality Management

*Reviewed by Scott Duncan*  
[sduncan@computer.org](mailto:sduncan@computer.org)

The title suggests a book about professional behavior/practice in light of legal and business issues arising in information technology (IT). The book is actually a preparation text for the British Computer Society's (BCS) "compulsory paper entitled 'Professional Issues in Information Technology.'" This means that most material is in terms of U.K. law and business practices, though reference is made to U.S. law and legislation as well as to differences in other countries (for example, Singapore, India, Sri Lanka). It is noted, however, that even in the U.K., English and Welsh law is not exactly the same as in Scotland, Northern Ireland, and the Isle of Man. Despite this, the book does provide an outline of professional topics and, in this regard, could be used to understand the scope covered by such issues.

Topics covered include:

- The basics of law and government
- The nature of a profession and professional organizations
- Financial issues (for example, start-up costs, accounting, management/project costs, investment)

- Organizational structure and management, including human resource issues
- Legal issues in business (for example, antidiscrimination, contracts, liability, intellectual property)
- Data protection/privacy
- Internet use and computer misuse

The appendices cover the BCS Code of Conduct and sample contract and agreement forms. Some chapters end with sample questions. From these, one can see that the BCS "paper" is an essay, not a multiple choice examination like ASQ's certification exams.

If readers are not interested in the BCS exam or U.K. law bearing on IT, then this book may not be of interest, since there are resources on the Internet that summarize the same topics (for example, check out Wikipedia). There are even slides from a university course on many of the topics covered in this book that may provide a useful overview of these issues: [http://www.dcs.shef.ac.uk/~rod/COM2040\\_2006\\_RHS/COM2040\\_Professional\\_Issues\\_Rod\\_2006-7.html](http://www.dcs.shef.ac.uk/~rod/COM2040_2006_RHS/COM2040_Professional_Issues_Rod_2006-7.html).

Scott Duncan has been a quality/process improvement consultant since 1994. He has been involved in all facets of software development since 1972, including publishing, state government, agricultural R&D, COTS, telecom, and financial systems.

### The Road Map to Software Engineering: A Standards-Based Guide

James W. Moore.

2006. IEEE Computer Society and John Wiley & Sons, Inc (<http://www.wiley.com>).

357 pages.

ISBN: 13-978-0-471-68362-9 and 10-0-471-683-62-0

CSQE Body of Knowledge area: Software Quality Management

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

This excellent book is obviously written by someone who has been immersed in standards work for many years. It is not an easy task to write a book that describes the history of standards and the relationships of the various standards-related organizations, and is also a useful reference document. Yet this book accomplishes each of these with aplomb. Whether the reader is looking for standards related to a particular knowledge area (for example, software quality) or for standards related to a life-cycle process (for example, software life cycle or system life cycle) this book identifies and briefly describes the applicable standard(s). The author identifies his personal goal—that this book provides an overview of the entire software engineering discipline by providing an entrée to the IEEE Computer Society series of books devoted to software engineering standards in different areas. This goal is deftly accomplished.

There are numerous reasons for an organization to adapt software engineering standards, and the author recognizes that not all organizations will adapt software engineering standards (one or many) for the same reasons. An organization may want to be more competitive by producing software that is of a higher quality and better able to satisfy the customer's needs, while another organization may need a "uniform framework for defining the relationship and the respective responsibilities of the acquirer and the supplier." The need to evaluate products emanating from its software engineering activities through criteria, processes, and measures is another reason for adapting software engineering standards. Finally, an organization may need to assure high integrity levels for its software where safety and dependability are important to protect lives or property.

The layout of the book makes using it easy. If readers merely want to learn about the history of the current standards they only need to read Chapter 2, "Standards-Makers" and Chapter 3, "Principles of the S2ESC Collection." If they want to have their own collection of standards they can read Chapter 4, "Organizing a Standards Collection." These chapters are included in Part 1.

Part 2 of the book is organized to follow the Guide to the Software Engineering Body of Knowledge (SWEBOK) and identifies the standards used for each of the knowledge areas. If readers are interested in standards that relate to the knowledge areas of software engineering then they need look no further than these chapters. Each chapter provides a description of the knowledge area and identifies the applicable standards. The knowledge areas include:

- *Software requirements*: The elicitation, analysis, specification, and validation of software requirements—those properties that must be exhibited to solve a real-world problem.
- *Software design*: The process and result of software architectural design, describing the system's top-level structure and organization, and identifying its components—and software detailed design—describing each component sufficiently to allow for its construction.
- *Software construction*: The detailed creation of working, meaningful software through a combination of coding, validation, unit testing, integration testing, and debugging.
- *Software testing*: The dynamic verification of the behavior of a program on a finite set of test cases, suitably selected from the usually infinite executions domain, against the expected behavior.
- *Software maintenance*: The totality of activities—both pre- and post-

delivery—to provide cost-effective support to the software system throughout the life cycle.

- *Software configuration management*: The discipline of identifying and controlling the configuration of a system and its software components to maintain integrity and traceability.
- *Software engineering management*: The application of management activities—including measurement—to ensure that the development and maintenance of software is systematic, disciplined, and quantified.
- *Software engineering process*: The definition, implementation, measurement, management, change, and improvement of software engineering processes.
- *Software engineering tools and methods*: Software development tools and environments and methods to develop software.
- *Software quality*: Software quality considerations—the degree to which a set of inherent characteristics fulfills requirements—that transcend the individual life-cycle processes of the software.

Those interested in learning about related disciplines need only turn to Chapter 16 for standards applicable to project management, quality management, and systems engineering. Part 3, which includes chapters 19 and 20, covers standards regarding the processes of software or systems engineering.

Perhaps the best part of this book is that the author astutely answers the questions that the reader is about to ask. For example, “What do I need to know in order to be successful in undertaking requirements analysis or requirements specification?” This approach of helpfulness continues throughout the book and into Appendix A, which lists in numerical order each of the standards described in the book.

Throughout the book the author has successfully interwoven the needs of software engineering with the standards that fill those needs while comparing and complementing the standards with each other. If there were anything else the reader would want from this book, I cannot contemplate what that might be.

Eva Freund is a subject-matter expert in independent verification and validation (IV&V). She performed IV&V activities for the National Archives ERA Project and was formerly the acceptance test manager and acting project manager for the IV&V Task for the FBI's IAFIS Program. Freund received her bachelor's degree from Fairleigh Dickinson University and her master's degree from Goddard College.

### **Process Improvement Essentials—CMMI, ISO 9001, Six Sigma**

James R. Persse.  
2006. O'Reilly (<http://www.oreilly.com>). 334 pages.  
ISBN: 978-0-596-10217-3

CSQE Body of Knowledge  
area: Software Quality  
Management

*Reviewed by Carolyn Rodda  
Lincoln  
carolyn.lincoln@  
catapulttechnology.com*

*Process Improvement Essentials* is an introduction to three popular process improvement models: Capability Maturity Model Integration (CMMI), ISO 9001, and Six Sigma. It is targeted to process managers and executives who are new to process improvement and want the 30,000-foot view. It does not provide enough information to actually implement a model, but it does outline an approach so that a newly appointed manager could decide what model to pursue and generally how to do it.

The book consists of eight chapters in two parts. Part 1 describes

how to establish and maintain a process improvement (PI) program. The author provides an excellent explanation of why one might want to launch a PI program, and also why one might not want to. He also explains and debunks six PI myths. Some hints for launching a PI program are to capitalize on strengths, focus on targeted improvements, and train and provide support.

Part 2 contains about 60 pages on each of the models. The reader will be able to understand the differences between the models themselves. For example, ISO and Six Sigma have broad applicability, and CMMI focuses on systems work. Since ISO is such a short standard, the author describes each paragraph. CMMI is much longer, so there is a paragraph about each goal. Six Sigma doesn't have a standard, so the author explains the define, measure, analyze, improve, control (DMAIC) and design for Six Sigma (DFSS) methodologies.

In addition to information on the content of each model, the author describes their infrastructure and history. For example, CMMI and ISO have owners and governing bodies, while Six Sigma does not. The final chapter of the book lists many areas with specific comparisons of the models, for example, required knowledge and skills, and proven effectiveness.

*Process Improvement Essentials* does an excellent job of accomplishing its stated purpose. It provides an overview of the challenges and promises of process improvement. It should be the first book read by someone new to process improvement, especially if one is in the information technology field. Author James Persse includes enough anecdotes and practical tips to make it an enjoyable literary experience. The book is also recommended for those who are familiar with one model and want to learn about the other two. It should be on a handy bookshelf for all process managers!

---

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 manager for Catapult Technology in Bethesda, Md. She holds bachelor's and master's degrees in math and was previously a programmer and project manager.

## SOFTWARE ENGINEERING PROCESS

### Software Specification and Design: An Engineering Approach

John C. Munson.

2006. Auerbach Publications (<http://www.auerbach-publications.com>). 377 pages. ISBN: 0-8493-1992-7

### Management of the Object-Oriented Development Process

Liping Liu and  
Borislav Roussev.

2006. IGI Publishing (<http://www.igi-pub.com>). 372 pages. ISBN: 1-59140-605-6

CSQE Body of Knowledge  
area: Software Engineering  
Processes

*Reviewed by Ray Schneider  
rschneid@bridgewater.edu*

Throughout my professional life stretching back to 1962 there has been the constant mantra of "the software crisis" with the subtext that software professionals just weren't doing it right. In the early days, the chief cry was, "If this goes on, software will soon cost more than hardware." That point was reached long ago and passed without much hoopla.

A sporadic stream of solutions has been proposed. Concepts and

methods were proposed to solve the problem. But software continues to be late, costly, and prone to errors. The flurry of activity that generates new methodologies, buzz words, and instant "silver bullets" goes on unabated. The two books in this review, if not solutions, are valuable contributions to this search for the perfect solution.

John Munson has written a challenging and confrontational book in *Software Specification and Design (SSD)*. He rejects current practices and challenges the computer science community to develop a true software engineering process. UML, he says, is a "debacle" and the object-oriented "religion" creates "incredible operating overhead bloat" so that "... systems developed using this metaphor literally suck out all the juice from the user's computer." He likens current practices to "building pyramids" that lose structural integrity when they get too large piling "code modules on top of one another until we reach the functional limits of the language metaphor."

In Chapter 1 he says, "The major problem confronting the computing industry today is not the speed of the hardware; it is the colossal inefficiencies in the software systems." The skeptical part of me said, "Just another software crisis, now what's his silver bullet?" Munson's answer to the problem is to propose a highly structured engineering process based on the Space Shuttle Primary Avionics Software System (PASS).

The key is to escape the idea that software is about code and put the software evolution process under life-cycle configuration control. A series of modeling steps develop weakly coupled, nearly independent "isolated steps" defining operational, functional, and modular specifications that precisely describe the system. Munson's claim: "In essence the software specification methodology developed in this book will permit software to be engineered and not crafted."

The book proceeds to guide the reader through this process in 15 chapters that describe each step and illustrate it using a simple calculator example. OxFxMxC summarizes this process of machine modeling. Three machine models capture the specification process. First the operational machine that defines the user interface is modeled, then the functional machine is captured by modeling the "functionalities of an underlying abstract software environment" and that is in turn converted to an architectural machine model that will implement the low-level design. Those are the OxFxM stages described in detail in the book. The last two levels involve code (language and real machine levels), the first is the mapping of M (architectural machine) to a virtual machine, which is how Munson terms the programming language level, a C-machine, an Ada-machine, a flight qualified HAL/S machine, and so on. The last step is the mapping of the virtual to the real onto the hardware of an actual computer—a real machine. These are the five machines that Munson says, "... figure into the complete specification and implementation of a software system."

*Software Specification and Design* is liberally laced with Munson's insights into what is wrong with existing development practices. He talks about software security, critiques existing practices, and often uses hyperbole. Putting operational and functional specifications under the kind of configuration control commonly reserved for code he calls "an astonishingly good idea" and characterizing design solutions that start with a specific language as Procrustean solutions forces the design to fit the language and not the other way around.

I rarely communicate with an author when reading a book and writing a review, but in this case I sent John Munson an e-mail. He was

kind enough to reply. He said, "From my work in the area, I can assert that the overwhelming majority of faults that wind up in the code are not coding problems at all. They are design problems, or they are specification problems. It only makes sense to isolate these two software development steps from the coding process."

Turning now to Liping Liu and Borislav Roussev's book, *Management of the Object-Oriented Development Process* (MOOP), is to turn back to the very world that Munson excoriates, the world of UML and object-oriented (OO) programming. It is profoundly ironic that there is, in fact, a kind of deep unity between the two books even though SSD is revolutionary and MOOP evolutionary. They both stress modeling up front and limit their attention to code as the implementation of the model. The 15 chapters of this book consist of a collection of presentations looking at a broad spectrum of technical, business, and social dimensions of OO development from the viewpoints of 21 authors drawn from the ranks of consultants, researchers, project and program managers, and educators.

Each chapter of the book has a similar layout: it begins with an introduction that has a description of what is coming, is followed by a development of the ideas in the chapter, and finishes up with conclusion, references, and endnotes sections. The individual chapters largely stand alone; however, there is an attempt at continuity exploring a broad range of topics on aspects of OO project management in such a way that earlier topics often illuminate later ones.

Chapter 1 sketches the history of OO programming and links it to the need to develop software engineering methods as a solution to the software crisis. Despite the continual maturation of OO technologies only 30 percent of software companies rely on them. The evolution of OO modeling methods culminating in

UML is described as is the seminal patterns work of the gang of four. The promise of reuse has not developed at the object level according to the author because "objects turned out to be disproportionately small" leading to increased interest "in things bigger than objects." These larger entities include components, domains, and product lines. The reader will find an informative introduction to UML2 providing a great introduction to the balance of the chapters.

In Chapter 2 three ways of producing software were described: code-driven, model-based, and model-driven. The model-driven architecture (MDA) uses Executable UML (xUML). A mature MDA methodology does not need to be programmed at all, instead a modeling language with executable semantics allows the models to be simulated and debugged.

A chapter-by-chapter, blow-by-blow description would expose some weak chapters and other strong ones. I particularly liked Boris Roussev's contributions, fully a third of the book, including one on xUML describing the use of finite state machines, action language, and object constraint language. He also provides a chapter on capturing user requirements and another on outsourcing to India using Agile methods, trying to reengineer large-scale outsourcing projects to benefit from Agile's "sweet spot." In keeping with Munson's skepticism one should be cautious to establish that Agile really does have a sweet spot before trying to capitalize on it.

I didn't plan it this way, but reading these interesting books together turned out to be a valuable experience in the clash of ideas. I came away from the experience with the conviction that there is a deep unity between these two very different model-driven ideas. Experience tells me that both are probably too optimistic, but the convergence suggests that something meaning-

ful is happening. Perhaps it really is time to stop thinking about software development as predominantly an exercise in code writing and elevate requirements and design to the lofty position they deserve. I highly recommend both of these books to developers and managers, especially in combination. They will stretch the reader's mind and give him or her the jump on the exciting future of highly productive software development through model-driven methods.

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 with more than 35 years of product development and applied research and development experience working for government, defense industry, and small business. He is currently an assistant professor in the Mathematics and Computer Science Department of Bridgewater College in Bridgewater, Va.

## **Practical Support for ISO 9001 Software Project Documentation Using IEEE Software Engineering Standards**

Susan K. Land and John W. Walz.

2006. Wiley-InterScience (<http://www.wiley.com/ieeecs>). 419 pages.

CSQE Body of Knowledge areas: ISO and Software Process

*Review by Linda McInnis  
lindamcinnis@yahoo.com*

This book is not only a book about the practical processes necessary for ISO 9001 certification but it can readily be used for most software development organizations as they try to mature their processes whether for ISO, Capability Matu-

rity Model (CMM), or other formal process levels.

The book contains good solid best practice for development organization. One of the daunting tasks in improvement initiatives is where to begin and how to document progress. The authors of practical support provide a starting point and ease the transition with straightforward templates on the enclosed CD-ROM based on IEEE standards.

What I particularly like about the book is the straightforward style that explains the basic concept and then highlights those areas that are particularly relevant to ISO implementation. For example, Chapter 4 is the implementation guidance; it starts the reader's effort out on a solid ground of reviewing what type of life cycle he or she currently has. Then one must build sponsorship for this activity and start by performing a GAP analysis of his or her current and desired states.

Another benefit of the book is that it can be used as a training tool for one's team and to help management understand what is being proposed. An especially helpful chapter is ISO 9001 for small projects, which explains how to work on an ISO project with a minimal team—challenging yes, but doable.

A further feature of the book is a well-done mapping of ISO items to IEEE standards, which is really helpful so that teams don't have to constantly invent forms and policy, thus speeding adoption.

On the whole, this book is definitely a worthwhile addition to a quality library providing great value in templates alone.

Linda McInnis has been a quality engineer for more than 20 years in bleeding edge technology, pharmaceutical informatics, and medical devices. She is a frequent speaker at local and national conferences.

## SOFTWARE METRICS, MEASUREMENT, AND ANALYTICAL METHODS

### **EMP III, Using Imperfect Data**

Donald J. Wheeler.

2006. SPC Press (<http://www.spcpress.com>). 315 pages. ISBN: 0-945320-67-1

CSQE Body of Knowledge areas: Software Metrics, Measurement, and Analytical Methods

*Reviewed by Scott Duncan*  
[sduncan@computer.org](mailto:sduncan@computer.org)

The title of this book would not make much sense unless one was familiar with Wheeler's prior book on *Evaluating the Measurement Process*. In his preface, Wheeler notes he was working on a revision of the second edition of this earlier book, but as it got larger and larger with more being added, he decided to write this completely new book. Thus, EMP stands for "Evaluating the Measurement Process" and the "III" is an acknowledgment that this would have been a third edition of the original book. As Wheeler notes, this book is a "book on characterizing, evaluating, and using imperfect data," retaining its emphasis on measurement systems and the nature of measurement data.

As such, the book deals with the uses of measurement, types of measurement studies, measurement system analysis, and using data. It is a reasonably detailed and mathematically rigorous treatment of manufacturing measurement practice and data analysis. As such, the more detailed and applied the material becomes, the more it drifts from what software practitioners, even software measurement specialists, would find useful. Since the issue of data imperfection has to do with errors and problems in measurement (of physical things),

its applicability to software is limited, given that software mostly "measures" by counting rather than by applying measurement to any objective attributes of the things being measured. For example, one can count the number of defects of a certain severity, but it is judgment, not measurement, that is used to assign defects to severity categories. A physical measurement equivalent might be assigning bolts to categories of small, medium, and large by deciding where individual ones go based on looking at them and deciding what is small, medium, or large rather than measuring them in some objective way for example, by length, width, or weight) and having ranges of results for each category.)

Not surprisingly, the earlier edition of this book is among those named in ASQ's Statistics Division's topical reference listing on their Web site. In formal metrology practice, this book would be quite useful, but for software practitioners, it would be of limited value.

## SOFTWARE VERIFICATION AND VALIDATION

### **Understanding Variation: The Key to Managing Chaos**

Donald J. Wheeler.

2000. SPC Press (<http://www.spcpress.com>). 162 pages. ISBN: 0-945320-53-1

CSQE Body of Knowledge area: Software Verification and Validation

*Reviewed by Scott Duncan*  
[sduncan@computer.org](mailto:sduncan@computer.org)

Donald Wheeler's books have long been recognized as doing a fine job covering the statistics and data landscape. Unlike some of his other books, *Understanding Variation* is not laden with formulae and is intended to raise awareness for how

# Reviews

---

to convert data into knowledge. Wheeler's main approach to this is to urge management (and those who prepare data for management) to look for the patterns in data. As he says in his introduction, "numerical naivete is not addressed by the traditional courses in . . . [public] schools, nor is it addressed by advanced courses in mathematics."

In his first two chapters, Wheeler provides two "principles for understanding data" based on: 1) knowing the context of any data, and 2) finding and filtering out the "noise" in the data.

Wheeler provides many examples of how to work with data using mostly run charts, that is, line graphs over time. More formally applied and computed, run charts eventually become controls charts, the heart of SPC. In this book, however, Wheeler stays away from too much mention to the math required to use formal SPC tools. This is not to say one should not use more formal tools, just that this book is not about going to extensive mathematical lengths in data analysis.

Indeed, it is this latter point that some may use to find fault with this book as being too light, perhaps, on the subject of variation. But Wheeler seems to have intended the book for management more than quality technicians and engineers who might be expected to do a lot of it. In this regard, from a software perspective, the book may be more useful than more detailed ones on SPC since it focuses, generally, on how to analyze process data to see patterns that provoke a better understanding of what is happening rather than to formally "control" an otherwise defined process.

## **Understanding Statistical Process Control**

Donald J. Wheeler and David S. Chambers.  
1992. SPC Press (<http://www.spepress.com>). 411 pages.  
ISBN: 0-945320-13-2

CSQE Body of Knowledge  
area: Software Verification  
and Validation

*Reviewed by Scott Duncan*  
[sduncan@computer.org](mailto:sduncan@computer.org)

In a sense, this book picks up where Wheeler's book, *Understanding Variation*, leaves off since the latter is a less mathematically rigorous treatment of variation and this book is far more detailed and comprehensive. On the other hand, where Wheeler's *EMP III, Using Imperfect Data* has limited value for software practitioners, this book is more useful because it provides a very good overview of statistical process control (SPC) regardless of one's domain of interest. And, again, it is among the books listed by ASQ's Statistics Division in its topical reference listing.

The book's foreword discusses the two types of variation from Shewhart: uncontrolled (special) and controlled (common) or, as some characterize them, process problems that recur regularly, balance out one another, and cannot be eliminated, and patterns of problems from individual, unique, mistakes, which can be eliminated. Ideally, one would like to properly assign all variation to the correct category, never assigning a problem to the wrong one. As the author notes, given that this is not possible, Shewhart hit upon SPC as a way "to achieve minimum economic loss from both mistakes" when we occasionally make them.

This book explains the history, theory, and application of SPC in great theoretical and practical detail by explaining what control charts are, how they are calculated, why they work, and how to "read" them to make use of the information they provide through the data points displayed as a result of sampling a process in action. Thus, on a control chart, uncontrolled variation is the oscillation around the central line, showing that variation exists, but understood in terms of the accept-

able level of the process at that time. Controlled variation is shown by deviations from regular oscillation around the central line such as data points that exceed upper and lower limits, a certain number of sequential data points above or below the central line, and so on.

On the applied side, examples come from basic manufacturing situations, so despite the comprehensive coverage of SPC given in the book, its application to specific software situations remains an exercise for the reader. Indeed, there seem to be no truly good books addressing the application of statistical methods to software development. Though there are many books on statistical methods, none has addressed software in a truly useful way for practitioners. It is not clear whether this is because SPC cannot be applied to software anywhere near as rigorously or in a statistically acceptable manner or because it simply isn't being done in software (as so many other engineering techniques are not being applied), so the experiences and knowledge is simply not there to draw upon.

Given the aforementioned limitation in a software context, this is a very good book on SPC and variation.

## **Software Testing: A Craftsman's Approach, 2nd edition**

Paul C. Jorgensen.  
2002. CRC Press (<http://www.crcpress.com>). 359 pages.  
ISBN: 0849308097

CSQE Body of Knowledge  
area: Software Verification  
and Validation

*Reviewed by Noreen Dertinger*  
[noreen.dertinger@cognos.com](mailto:noreen.dertinger@cognos.com)

*Software Testing: A Craftsman's Approach* by Paul C. Jorgensen is a classic that should be read by, if not on the shelf of, software testers

---

at all levels along with Glenford Myer's classic, *The Art of Software Testing*. The author brings many years of experience to software testing, having worked the first 20 years of his first career developing, supporting, and testing telephone switching systems. An initial version of this book was published in 1995. In this second edition, Jorgensen has significantly changed the content of the five chapters in Part 5 to address the dominance of UML at the time of writing. He has also replaced the now outdated Pascal examples found in the first edition with more language-neutral pseudo code. According to the Amazon (ca and com) Web sites, a third edition of this book is scheduled to be released in November 2007, and although no details about this new edition's content were available at the time of this review, it is assumed that the author will have made further significant changes to keep up with current developments in the software world.

Jorgensen has written *Software Testing: A Craftsman's Approach* in a very clear, accessible style. He covers most of a software tester's knowledge base of testing techniques, beginning with a perspective of testing "why do we test?" and continuing with areas such as boundary value testing, decision table-based testing, equivalence class testing, and others.

This book is largely based on mathematical principals but both the prose and the supporting examples are clear. One does require a certain level of math skills to follow them, but an advanced degree in mathematics is not required to understand the text. Parts 2 and 3 are based on three examples to illustrate various unit-testing methods. The first is a triangle problem, which has been greatly discussed in testing literature, where a triangle program accepts three integers, a, b, and c, as input (sides of the triangle) and the output of the program is the type of

triangle as determined by the three sides: Equilateral, Isosceles, Scalene, or NotATriangle. This problem is sometimes extended to accept right triangles and this is done in some of the exercises. The second example is a logically complex function, NextDate, which is used to illustrate logical relationships among the input variables. There are two sources of complexity for this problem, the first being the complexity of the input domain and the second being the rule determining when a year is a leap year. The third example is a commission problem that contains a mix of computation and decision problems thus leading to interesting testing questions.

Part 4 consists of a discussion of integration and system testing and uses three other examples:

- A simplified version of the automated teller machine containing a variety of functionality and interactions typifying client-server systems
- A currency converter to illustrate a typical event driven GUI interface
- The windshield wiper control device from the Saturn automobile to illustrate the discussion of interaction testing in Chapter 15

In Part 5, an object-oriented version of the NextDate problem is provided to illustrate the aspects of testing object-oriented software. In this section, Jorgensen examines how traditional testing can be extended to address the issues associated with the characteristics of object-oriented software—inheritance, polymorphism, and encapsulation. The remaining chapters of Part 5 cover class testing, object-oriented integration testing, GUI testing, and object-oriented system testing.

My only complaint about this book lies not with the content but with the quality of the binding. The copy I have, although hardcover, appears to be fragile and prone to fall apart without much stress being placed on the book. Hopefully,

the quality of the binding of the forthcoming edition will be better and more durable so that the book lasts its lifetime to the next edition without falling apart.

### **System Testing with an Attitude: An Approach That Nurtures Front-Loaded Software Quality**

Nathan Petschenik.

2005. Dorset House Publishing (<http://www.dorsethouse.com>). 350 pages. ISBN: 0-932633-46-3

CSQE Body of Knowledge area: Software Verification and Validation

*Reviewed by M. Glenn Newkirk  
[glenn\\_newkirk@infosentry.com](mailto:glenn_newkirk@infosentry.com)*

Did you hear the one about the software tester who walked into a bar with an ostrich? Probably not. But you probably heard the one about the lawyer, priest, rabbi, minister, doctor, nurse, or person of hair color who did. Generally, software testers have not been taken seriously enough to warrant many good jokes about them. Very few ambitious information technology (IT) professionals aspire to work as software testers. They want to do almost anything but manage software testing. It is one part of the IT profession that has guaranteed obscurity. Business managers and even project managers ignore software testers, viewing them largely as project costs. They are roadblocks to meeting deadlines. Technical whiz-kids dislike testers who find fault in otherwise elegant program designs and code.

Typically, it is in regulated industries and certain life-or-death environments, such as the space industry, certain financial organizations, and public safety

# Reviews

---

organizations, that software testers have had prominent roles in project teams. The system project often could not move forward until the project developers satisfied the testers' quality metrics. Nathan Petschenik's *System Testing with an Attitude* offers a useful perspective that can help change the dynamics between software testers and other development team members. The book's target audience includes software project managers, development team leaders, quality assurance groups, and system testers.

The author's passion for system testing and its role in the development process is evident in the book's title and in every chapter. The "attitude" that permeates the book is expressed on the front cover: one must design in and build in quality because one cannot test in quality. *System Testing with an Attitude* provides solid coverage of the entire system development life cycle. It supports an understanding of overall project management. The book explains thoroughly the roles for specific test techniques throughout the life cycle. Petschenik bases his argument for "frontloading" test steps on both other researchers' findings and his own in-depth experience in system testing.

The author divides the book into two sections. The first section addresses many issues often associated with system testing in small and large projects alike. Anyone who

has been involved in a real system development project will recognize most, if not all, of these issues. How much testing should be in the project? What is the real role of a system tester? How should a project handle the important change management issues associated with system testing? How should teams use testing to avoid unpleasant surprises for users?

The book's second section provides solutions to these and other system testing issues. It also provides a way of addressing these questions and projects. That is perhaps a more valuable contribution. This section really consists of four subsections on solutions to many of the issues raised in the first section. One subsection provides a solid review of systems testing methodology, techniques, and tools. The remaining three sections focus on practical project management and organizational tactics for providing the "attitude" mentioned in the book's title. One of the most useful, though very short, chapters discusses how to organize and structure a test team.

So-called "extreme programming" practitioners probably will not spend much time with this book. It gives an altogether too solid roadmap for proper testing and argues for sufficient times for testing. Petschenik describes a kind of role for testing that many extreme programmers might view as mere overhead if not outright

obstructionism to rapid, dynamic application delivery.

This book can provide great benefit to its target audiences. It also would provide tremendous understanding to much wider audiences that probably will never see the book. Those audiences include business division managers, financial officers, and organizational executives. Also, academics and software accreditation mills that teach new programmers would do well to require their beginning students to read *System Testing with an Attitude* before those programmers write their first line of code. Perhaps they would understand from the beginning of their career what real testing is. Perhaps they would understand how testers can help them succeed.

One will know when business managers, corporate executives, academics, and new programmers have taken this book seriously. Then there might be some good jokes or some good comic strips about software testers. It will be the ultimate sign of respect. If not, one can at least hope that project managers, quality assurance team members, and testers will read the book and put its attitudes and techniques into practice.

M. Glenn Newkirk is president of InfoSENTRY Services, Inc. in Raleigh, N.C. He is a certified software project manager and a Certified Business Continuity Professional. He has managed large information networks and information systems implementation projects for 30 years.