

# Resource Reviews

## GENERAL KNOWLEDGE

### **The Career Programmer: Guerilla Tactics for an Imperfect World**

Christopher Duncan.

2006. Apress (<http://www.apress.com>). 280 pages.

ISBN: 1590596242

CSQE Body of Knowledge  
area: General Knowledge

*Reviewed by Scott Duncan  
sduncan@computer.org*

This is a book about personal experiences in pursuing a career as a programmer. As the author notes, "Managing a successful career as a programmer has almost nothing to do with writing code." Thus, "This is not a language or technology book." There is not a line of code (nor a bibliographical reference) in it. This book is about "overcoming the obstacles you face on the job." It is "a field manual for the software developer grunts . . ." The author indicates his audience includes those for whom global outsourcing has threatened their jobs because he has "seen the numbers" that show how people "working on high-profile projects in other countries . . . make less than the local fry cook [in the United States] gets for flipping burgers" and, "in some countries, that's extremely good money." Thus, "the biggest issues we face today are all career related," not those of technology, at least not for the typical career programmer.

To address programming as a career, the author has divided the book into three parts:

- Part 1 addresses the realities of "life in a corporate world" in

three chapters covering issues of corporate bureaucracy and politics, common development team challenges, and management problems (which overwhelm issues of coding).

- Part 2 addresses specific survival skills in eight chapters covering unrealistic deadlines, requirements "scope creep," design methods, estimation techniques, quality assurance practices, coding standards, configuration control, communications, managing your management, and dealing with political games.
- Part 3 addresses "finding and keeping a job" in five chapters covering looking after your own career, getting a job, considering a move away from coding into other areas, entrepreneurship, and people skills for job security.

Overall, the author's approach to career survival can be summarized by his statements at the beginning of Chapter 3: "We've already identified a large number of culprits that appear to be responsible for the problems we encounter, but, when it all comes down to the bottom line, *it's your fault*. . . . If you sit on your hands and do nothing, then you're part of the problem when you could be part of the solution." When reading this, I was reminded of Tom DeMarco in his book, *Slack*, where he says, "Dilbert . . . is no hero" because he "and his ilk . . . make stupid management possible" by keeping "his head down," never objecting, never making waves, and never putting "his job on the line." While DeMarco is more blunt about his urging people to be leaders against dysfunctional behavior, Chris Duncan does ask that, if you

take such a risk, you have to believe there is something in it for you.

His answers:

- "Consistently meeting your deadlines" will "lower your stress level and make you less likely to be harassed by management," allowing you to "avoid pointless overtime."
- Regain control, spend "less time putting out fires," and enjoy the coding work more because it will be the kind you like to do, not have to do.
- "Work on the cool projects," which "come to those who make an effective effort to get them."
- Earn "the high regard" of managers and conduct oneself "in an ethical fashion."
- "Make better money," which will require "more than just your technical prowess," including being effective in meetings, possessing "good organizational and navigational skills," balancing technical leadership (that is, managerial) and coding responsibilities, controlling your time, and knowing what you want in order to get it.

If it seems, so far, that the book is about "soft stuff," there's a good bit of that. But Part 2 imparts some good practical advice. For example, Duncan offers a checklist of questions related to getting effective requirements that focus on "what" rather than "how." He also addresses approaches to design, especially when there is not much time for it. Then there is advice about estimation and justifying efforts to assure quality in the product, including the relationship developers should have with the test staff.

Overall, this is an interesting book that takes many familiar

# Reviews

---

concepts and suggests how you might go about trying to perform well when the environment around you appears to work against you. I would say, though, that Duncan's implicit suggestion is to use the advice in the book to get yourself into a position where you can choose your work situations and not have them thrust upon you or feel forced to take them on regardless of their unreasonableness and your potential unhappiness doing them. It is important to remember that Part 3 of the book is about how to look for and find those situations and that, though Duncan decided to become (and still is) an independent, contract programmer, his final advice about job security, like his early advice, is that "you—*are* in complete control of your career."

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.

## SOFTWARE QUALITY MANAGEMENT

### Joel On Software

Joel Spolsky.

2004. Apress (<http://www.apress.com>). 362 pages.

ISBN: 1590593898

### The Best Software Writing I

Joel Spolsky, ed.

2005. Apress (<http://www.apress.com>). 305 pages.

ISBN: 1590595009

CSQE Body of Knowledge areas: Software Quality Management, systems documentation

Reviewed by Ray Schneider  
[rschneid@bridgewater.edu](mailto:rschneid@bridgewater.edu)

If you've been locked in your cubicle by the pointy-haired boss from Dilbert and keyboarding your fingers to the bone for the last seven years then you might not have noticed Joel Spolsky. I stumbled on his Web site, [www.joelonsoftware.com](http://www.joelonsoftware.com), a couple of years ago when I was surfing around for something on Unicode and found Spolsky's article on what you have to know about Unicode. It was so interesting and entertaining that I found myself putting a link to his site on my college Web page, and I signed up to get e-mail from him. He hasn't disappointed.

Here are two books that are both entertaining and instructive. The first, which takes the name of his Web site, *Joel On Software*, is a series of articles that began life on his Web site as something like a blog of day-to-day insights. The book has the major advantage of organizing these into a cohesive thread, while reading them in the original sequence is rather more like jumping on a pogo stick.

The second book, *The Best Software Writing I*, is a series of 29 eclectic writings by folks nominated by the Web site's "faithful readers," as Spolsky describes them in the Introduction. *The Best Software Writing I* is as entertaining and instructive to read as *Joel On Software*. The biggest problem readers might have is feeling guilty about having fun reading a software book. Aren't such books supposed to instruct and not entertain? Spolsky's retort would be, "The software development world desperately needs better writing." Who can disagree with that? When was the last time you read a riveting software specification?

*Joel On Software* is an opinionated book. Spolsky says in the Introduction that he's had to "leave out the words 'in my opinion' from the beginning of each sentence . . ." Readers might well find themselves disagreeing with Spolsky on any number of matters, but they won't

be bored. He makes his case with examples drawn from experience in the trenches at Microsoft where he was a program manager on the Excel team responsible for converting Excel from a macro-driven to an embedded language, which became Visual Basic for Applications. From there he went on to Viacom Interactive Services and then Juno Online Services before starting his own company, Fog Creek Software, with a friend in 2000. This is grassroots experience not academic theorizing.

*Joel On Software* is divided into three main parts and two smaller parts. The first part is on the practices of programming. As a CS professor that came up the old way, machine language first, Spolsky's theme that one must drill down to the lowest level distinctly resonated with me. If you work for one of the small or even large companies that think writing software doesn't require process control, then Chapter 3, "The Joel Test: 12 Steps to Better Code," is a must read. Readers can rank their company's software coding efforts on Spolsky's 12-point scale.

There are also four chapters on painless functional specifications that even folks who think they know about specifications should read. Spolsky ornaments his thoughts with colorful examples and shows how to write specs that are both instructive and entertaining.

Part 2 talks about managing developers. Spolsky explores staffing, the dysfunctional aspects of incentive pay, why testers are essential, why multi-tasking people doesn't work, and more. Every topic is penetrating and entertaining. Part 3 is about random thoughts mostly on strategic themes. Much of it is about conventional wisdom that isn't. One piece I found particularly interesting explored when and why open source software works. Another focused on Microsoft's Internet strategy and why Spolsky thinks Microsoft has lost the API war.

---

Software developers, especially newbies, should have *Joel On Software* on their bookshelf. It probably takes a decade of experience to see why Spolsky is right, but as entertaining as the book is, perhaps readers can get some of that experience vicariously and save all that time.

*Best Software Writing I* is equally opinionated. Each of the 29 pieces starts with an introduction by Spolsky of various lengths as he makes his own points about the author or the issue. Readers are introduced to the 25 coauthors with brief bios at the beginning of the book. The essays and articles vary in length from a couple of pages to 20 pages or so, some in multiple parts. They are generally as entertaining as Spolsky but with different voices and sometimes different points of view.

Food for thought includes both parts of Clay Shirky's articles on social software. Readers might want to spend a week thinking about where the future of group interaction software is actually heading after reading that pair. There is too much in *Best Software Writing I* to go through everything.

Frankly, it isn't often that one can absorb rich experience from the grassroots and enjoy every minute of it. I can enthusiastically endorse both of these books. They will be rewarding for both the novice and the experienced software developer. These are definitely two keepers.

Ray Schneider holds a bachelor's degree in physics, a master's degree in engineering science, and a doctorate in information technology. He is currently an assistant professor in the Mathematics and Computer Science Department of Bridgewater College in Bridgewater, VA.

## SOFTWARE ENGINEERING PROCESS

**Agile Software Development,  
Second Edition—The  
Cooperative Game**

Alistair Cockburn.  
2007. Addison-Wesley  
Professional ([http://www.  
awprofessional.com](http://www.awprofessional.com)).  
467 pages.  
ISBN: 0-321-48275-1

CSBE Body of Knowledge  
area: Software Engineering  
Processes

*Reviewed by Carolyn Rodda  
Lincoln  
Carolyn.Lincoln@l-3com.com*

The second edition of *Agile Software Development* is a 2007 update to the original published in 2001. Since then, agile development has become more popular and widely used. The author, Alistair Cockburn, has been at the forefront of agile development since the beginning; he was one of the framers of the Agile Manifesto and co-edits an Addison-Wesley series on agile. He also has created the family of Crystal methodologies for agile development.

The theme of the book is that software development is a cooperative game of invention and communication with a primary goal of delivering useful, working software and a secondary goal of setting up for maintenance or further development of the system. The seven chapters explain those principles and then address how to develop an agile methodology. A methodology includes roles, skills, teams, techniques, activities, process, work products, milestones, standards, quality, and team values. There is a very high-level description of the Crystal methodologies to illustrate agile methodologies with differing levels of weight and ceremony.

*Agile Software Development* describes the philosophical and practical underpinnings of agile and also includes justifications and answers to objections posed by critics. Example topics are

the three levels of learning and communication, Goldratt's theory of constraints, lean development, and methodology design principles and errors. All of the illustrations are for developing software, but Cockburn says agile principles can be applied to many other areas of life, such as constructing a house.

If the reader is interested in the theory and philosophy of agile development, this book is an archaeological dig into the thought processes of the agile movement. One result is that its style is rambling. Each chapter has the original version plus a separate addendum for the second edition. The same topics were addressed in several places but from a different angle. For example, the idea of collating developers must have been mentioned dozens of times. There doesn't appear to be a clear line of reasoning or direction throughout the book.

This book will be most meaningful to someone who is familiar with agile methodologies and wants to understand them on a theoretical basis. It will not be helpful to someone who wants descriptions of specific methodologies. Extreme programming, Scrum, the Dynamic System Development Method, and others are mentioned by name, but they are not described in enough detail to use. Other books listed in the appendices may be consulted for that purpose.

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 in quality assurance and measurement for L-3 Communications Titan Corporation in Reston, VA. She holds bachelor's and master's degrees in math.

**CMMI Survival Guide/Just  
Enough Process Improvement**  
Suzanne Garcia and  
Richard Turner.  
2006. Addison-Wesley

# Reviews

---

Professional (<http://www.awprofessional.com>).  
299 pages.  
ISBN: 0-321-42277-5

CSQE Body of Knowledge  
area: Software Engineering  
Process

*Reviewed by Uma Reddi*  
[Reddi1@llnl.gov](mailto:Reddi1@llnl.gov)

This book is very readable and filled with many anecdotes. It brings down the formidable topic of Capability Maturity Model Integration (CMMI) into nonprofessional terms without sacrificing the rigor. The book is general enough, provides material for varied readers from the uninitiated to the experienced professional, and can be useful for small organizations to large conglomerates.

The book is divided into five parts, each dealing with some aspect of process improvement. In Part 1, the book introduces continuous process improvement by briefly introducing various paradigms. These include the plan, do, check, act (PDCA) cycle, the IDEAL cycle, Six Sigma and its framework of design, measure, analyze, improve, and control (DMAIC), NASA's QIP, and others. Several reference models such as CMMI, CMMI\_DEV, ISO, ITIL, COBIT, and others are discussed, giving the reader a broader choice. Several common pitfalls of process improvement initiatives are also addressed.

Part 2 of the book provides glimpses on how to get started with CMMI. Several choices that resolve around whether CMMI should be used are discussed. A chapter is devoted to decision-based life cycle for improvement.

In Part 3, a fictional story is presented making a case for process improvements. Several tools that help in moving forward with process improvements are discussed.

Part 4 contains the meat of the book. It discusses in several chapters

challenges facing process improvements and the techniques to tackle them. This part essentially discusses how to go about implementing CMMI in an organization.

Part 5 gives a list of extensive resources that are available for anyone willing to take this process improvement journey.

This is an excellent reference book that packs a lot of information into 299 pages. All things related to process improvements in general, and CMMI in particular, are introduced and discussed briefly. This book is a must read for anyone contemplating how to start or continuously improve processes for delivering better quality products.

**Uma Reddi** is a software quality assurance manager with Lawrence Livermore National Laboratory. He has two bachelor's degrees (one in electrical engineering and the other in math), and a master's degree in electrical engineering. He has worked for nearly 40 years in all facets of software product development in major U.S. computer companies (IBM and Hewlett-Packard). He is a Life Senior member of IEEE and a Senior member in ASQ.

## PROJECT AND PROCESS MANAGEMENT

### **Managing Iterative Software Development Projects**

Kurt Bittner and Ian Spence.  
2007. Addison-Wesley  
Professional (<http://www.awprofessional.com>).  
672 pages.  
ISBN: 0-321-26889-X

CSQE Body of Knowledge  
area: Program and Project  
Management

*Reviewed by Pieter Botman*  
[p.botman@ieee.org](mailto:p.botman@ieee.org)

Anyone reading the popular literature on agile development methods will have a basic appreciation of

the concept of iterative development. This concept is common and fundamental to all of the agile methodologies.

The first part of the book defines an iteration from the perspective of a development/project manager, and the close players (roles such as customer representative, business analyst, developer, and sponsor). The authors introduce the many forces shaping the project and its iterations: business value, risk, resources, and quality. They discuss the meaning of project success, and follow this by introducing various measurements that are critical to the project manager, such as verified requirements, quality/defect density, architectural risk/stability, and estimate to complete.

The authors do not insist on the use of the Unified Process (UP) or its variants. However, they use terms from the UP and use the UP life-cycle phases as a framework or starting point for discussing iterations. They do not describe the entire UP in detail, but introduce at some length the UP life-cycle phases (inception, elaboration, construction, and transition), the life-cycle phase objectives, and the UP's strong risk management approach. The many technical practices of UP, along with most UP documents and work products, are not central to this explanation.

The second part of the book deals with "Planning and Managing an Iterative Project." Bittner and Spence approach planning hierarchically, on three levels: the level of the iteration, the level of the development (or "evolution," the term used to describe one pass through the UP phases), and the level of the overall project. Each of these levels has objectives, milestones, and deliverables described in some form of plan, yet the development manager must synthesize and continually harmonize the plans at all three levels.

Many guidelines are included in these chapters, including rules of thumb for iteration lengths, team

---

sizes, and relative effort/duration of each UP phase. Chapter 7 includes a section that attempts to generalize some aspects of iterations and phases, using pattern terminology. One example is a list of the forces shaping the extension of phases through added iterations. Four patterns are presented that address the organization and planning of an "evolution" in terms of iterations. These patterns address cases where the standard UP phase model is adapted by the development manager, an example being the "Immediate Construction" pattern, in which no iterations in the inception and elaboration phases are required. The discussion of these variant applications of the iteration within the UP life cycle is valuable, both for readers with a UP background and for those agilists who may be unfamiliar with the significance of the respective phases.

The second part of the book also contains two chapters that go beyond the fundamentals of managing within a single iteration, and these are worth noting. Chapter 10 discusses scaling up the management approach to handle larger projects and programs. The themes of risk management, breaking down of systems into coherent components or subprojects, incremental delivery of business value, and coordinated planning at several levels are reiterated and applied to the problem, although the treatment here does not attempt to tackle all of the technical work and infrastructure needed to support the management of large, scaled-up development projects. Chapter 11 contains advice on how a project manager can get started with iterative project management. This too is an interesting topic, and even inexperienced project leaders will gain some insight from the discussion. However, such a short chapter cannot do justice to the large topic of process improvement and organizational change.

The third part of the book is devoted to appendices. Appendix A contains an overview of use cases, and summarizes their application through the various UP phases. This chapter contains useful material, but seems a bit disjointed from the main text. It makes the point that use cases are an important basis for planning iterations, and drive subsequent development activities throughout the life cycle. But the important theme of selecting and bundling appropriate chunks of functionality for an iteration is introduced in Chapter 4, and revisited in Chapter 7.

It would be asking too much for this book to compare and contrast all of the agile and iterative development methods. However, the concept of iteration is at the heart of all the agile methodologies. Readers (especially those coming from a background of XP or SCRUM) would have benefited from an expanded discussion in the main text of the commonalities and differences between the various methodologies with respect to iterations and their packaging of functional units ("stories," use cases, and so on).

Appendix B provides sample templates, document outlines, and checklists. These cover artifacts directly related to the planning and control at the project and the phase level, but it is the checklists at the iteration level of detail that prove to be the most interesting for those already familiar with the UP. Appendix C contains four sample documents from a specific development project, and here again it is the two sample documents at the iteration level (the iteration plan and the iteration assessment) that provide the most value to readers already familiar with the UP.

The authors have succeeded in presenting a reasonable approach to the organization and management of iterative development projects. They provide a great deal

of information in the forms of rules of thumb, and some useful guidance to the project manager in making the many tradeoff decisions that come with iterative development. Readers familiar with the philosophy and structure of the UP will find that this book adds depth to their understanding of planning and management of iterations within that structure. Readers familiar only with SCRUM or XP development may balk at the structure and the detailed nature of planning and control, but a scalable approach demands these.

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.

### **Implementing Lean Software Development: From Concept to Cash**

Mary and Tom Poppendieck. 2006. Addison-Wesley Professional (<http://www.awprofessional.com>). 304 pages. ISBN: 0-321-43738-1

CISQE Body of Knowledge area: Project Management; Software Engineering Processes

*Reviewed by Robert Ferguson*  
[rxf@sei.cmu.edu](mailto:rxf@sei.cmu.edu)

Mary Poppendieck is widely known for her talks on lean software development. She and her husband, Tom, also published the book, *Lean Software Development*, in 2003. The new volume is a set of vignettes that describe examples of lean principles taken from their experience in product development. They have cast

# Reviews

---

a wide net for this volume. It includes tales from automobile manufacturing, ready-to-wear clothing, and software development firms. The tales are interesting and do a good job of highlighting what they have described as "lean principles."

After a brief history, in Chapter 2 the Poppendiecks identify their "Seven Principles of Lean Software Development." These are

1. Eliminate waste
2. Build quality in
3. Create knowledge
4. Defer commitment
5. Deliver fast
6. Respect people
7. Optimize the whole

The remaining chapters are mostly stories that show how different firms have implemented one or more of these principles. In some cases, the stories come from Toyota (the originator of many lean principles), or are selected to demonstrate one of W. Edwards Deming's 14 principles. The Poppendiecks use this history to show the strength of their seven principles.

The stories are an appealing way to describe how each lean principle has been enacted in an organizational context. A few of the stories are particularly good reminders. "Respect people" is sometimes forgotten today. Deming and the Poppendiecks remind readers that people want to be able to take pride in their work. They also remind us that the assembly line is often dehumanizing. The idea of the assembly line is to make the working person a completely replaceable entity—the task is designed so the skill can be taught quickly. The demonstration of assembly line mentality in product development is seen in companies that determine it is better to hire skills than it is to train employees. This shows little respect for people and implies that training is a waste of money.

The book cites the open source movement as an example demonstrating what happens when workers have pride in their product. Open source software is often a labor of love. Many people may contribute development, testing, and documentation to open source products. Their contribution, for the most part, is not compensated in any way except recognition by their peers. The remarkable observation, however, is that the quality of open source software is very high. Few commercial products achieve the same levels of quality.

Automation, say the Poppendiecks, should enhance the skills of the worker rather than replace their skills. An example of enhancing their skills is demonstrated in "test-driven development." Writing the test cases and automating them eliminates the need to have developers devise and run tests many times. The attempts to "test quality in" at the end of the development cycle nearly always fail anyway. The testers usually run out of time long before the desired level of quality is reached.

I like the stories and believe each can be a constructive way of thinking about one of the principles and its implementation, although I have a couple of warnings about attempting to adopt a book like this without some help. First, the suggested role definitions may be difficult to implement if the change in role attempts to create a new organizational structure. Toyota uses someone called a "chief engineer" to work between the customer and the engineers (product development personnel). The chief engineer must work to understand the customer's value proposition and help to translate this into the product architecture and specifications for the development work. Using the term "chief engineer" may call up different images and conflicts in other organizations. Whatever the role is called, the key is the initial stage of the product. The customer's values

have to be understood and tested against the product architecture. Whether this is one role or two and how they are named is not the issue.

Second, some methods are difficult to implement. Refactoring, for example, seems to be hard for some organizations to implement. I believe this happens because the organization has adopted "local optimization." The system feels that stopping to remake something that already works is considered a waste. Failure to refactor has shortened the life of many products. Linux would already be shelfware had not it been refactored for version 2. The Poppendiecks understand this difficulty and describe it briefly.

In summary, the book is an easy read. The stories mostly stand alone, so it is not necessary to adopt every strategy at one time in order to benefit from lean methods. The Poppendiecks have included a lot of wisdom so that an individual new to lean concepts can actually get something from the book.

Robert W. Ferguson is a Senior member of the technical staff at the Software Engineering Institute where he works in software engineering measurement and analysis. He has a master's degree in mathematics and about 30 years of experience in software development and project management.

## VERIFICATION AND VALIDATION

### Modern Software Review: Techniques and Technologies

Yuk Kuen Wong.

2006. IRM Press (<http://www.irma-international.org>)

331 pages.

ISBN: 1-59904-013-1

CSQE Body of Knowledge area: Verification and Validation

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

There has been a great deal published over the years about reviews. Most people are likely familiar with the Fagan, Freedman and Weinberg, Gilb, and Weigers books and articles on the subject. So the immediate question when faced with another book on the subject would be, "What's new?" The author of this book answers by stating: "Readers will gain a deeper understanding of current software review literature and theoretical models for analysis [of] software review performance. More specifically, this helps readers to understand the critical input and process factors that drive software review performance."

What struck me first in reading this book, unfortunately, was the weak editing job. There are missing words, atypical grammatical structures, mismatched pronoun references, and incomplete sentences at various points in the book. The author may not be a native English speaker, and it may

be that the editing was not done by a native English speaker, either.

That being said, I would classify this book more as a study/survey of work and ideas on software reviews rather than a practitioner's book. The book addresses many models and research perspectives, including discussion of how to design an industry survey on reviews. It also features the author's own "conceptual EIIO model for software review." Despite the claim of offering "practical guidelines for software reviews," most of the book is a summary of some methods and tools plus coverage of research and survey design approaches. Indeed, the author positions her EIIO model as "a sound theory to drive empirical research in software review and a different perspective to existing literature."

The second to the last chapter is entitled "Recommendations for Conducting Software Reviews." Within this chapter is a section on recommendations for practitioners.

The author says to focus consideration when planning reviews on:

1. Decision on the selection of inputs.
2. A determination of the level of review meetings required.
3. Identifying the measurement metrics.

However, the author devotes less than three pages to these three items followed by a "Summary of Key Points for Conducting [a] Software Review," which is a bullet list of suggestions.

While interesting as a discussion of the literature on software reviews, I believe a practitioner would not find this book valuable compared to those by Freedman and Weinberg, Gilb, or Weigers.

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



Software  
Division

Members of the Software Division are software quality professionals, software engineers, and others interested in applying quality principles to the field of software development. The Software Division offers newsletters, journals, conferences, a Web site, certification, and networking activities. A Software Division-only membership is just \$29 per year!

In upcoming issues of *Software Quality Professional* we will present the value of belonging to and participating in the Software Division of ASQ. There are many ways to benefit from membership from general awareness to specific skill learning opportunities available to members. We will discuss these benefits and how to turn them into positive results in your personal professional growth.

**Call 800-248-1946 or 414-272-8575, or join us online now at <http://www.asq.org/software> to exchange knowledge and experiences!**