Book review

Fundamental Concepts for the Software Quality Engineer
Edited by Taz Daughtrey

American Society for Quality -- Quality Press, 2002
ISBN: 0-87389-521-5
Cover price: US$50.00
Hardcover, 288 Pages

Fundamental Concepts for the Software Quality Engineer is a collection of twenty excellent articles, reprinted from *Software Quality Professional* magazine and the proceedings of recent conferences. The range of topics is wide, so readers with a variety of interests will find relevant and useful articles. Overall, the book is a good way for someone with a basic knowledge of software quality engineering methods to gain a deeper understanding of various techniques and processes, and to learn practical lessons from case studies. Many of the articles will be as interesting to software project managers and test managers as they are to quality engineers.

The structure of the book reflects that of the American Society for Quality's Software Quality Engineering Body of Knowledge, a list of topics that engineers must master in order to pass the ASQ Certified Software Quality Engineer (CSQE)[1] test. However, this is not intended as an exam preparation textbook. Rather, it is a selection of the best two or three recent articles on each topic, so that the book as a whole presents a broad picture of current developments in the field. The eight sections of the book are:

- Standards, Principles, and Ethics
- Quality Management
- Software Processes
- Project Management
- Measurement
- Inspection and Testing
- Audits

- Configuration Management

I'll briefly describe each of the chapters in each section so that readers can pick and choose those that align with their interests.

Standards, principles, and ethics

In this section, Watts Humphrey's article "The Software Quality Profile" describes how deploying the Personal Software Process and the Team Software Process can improve software quality and team productivity. "Choice and Application of a Software Quality Model" by Dave Miller summarizes several well-known methods of measuring software quality -- Boehm's model, Cavano and McCall's model, FURPS+, Dimensions of Quality, ISO 9126, and the SEI model -- and compares them, with an eye to helping readers choose the model that best fits their application. "Risk Management: Supporting Quality Management of Software Acquisition Projects" by Gerard Getto discusses ways to identify and manage risk when buying software and identifies key risks and how to address them.

Quality management

This section includes John Elliott's "Achieving Customer Satisfaction Using Evolutionary Processes," which presents the results of an experiment using the Dynamic Systems Development Method to ensure that a software project would meet customer requirements and achieve a high level of customer satisfaction. "People Management and Development Process" by Giovanni Evangelisti, Emilia Peciola, and Cosimo Zotti is a case study of how a company improved its management practices, which led to happier employees and lower turnover. The lessons apply not just to software, but to any sort of management situation.

Software processes

This section begins with "Risk Identification Techniques for Defect Reduction and Quality Improvement" by Jeff Tian. He presents a collection of risk identification techniques ranging from traditional -- such as correlation analysis -- to cutting-edge, such as artificial neural networks. In "Cost of Software Quality: Justifying Software Process Improvement to Managers," Dan Houston supplies ammunition for quality engineers trying to convince their managers to invest in process improvements. Applying the Cost of Quality technique developed decades ago by J. M. Juran for industrial manufacturing, Houston derives a method for calculating the financial impact of implementing software development process improvements -- or of failing to implement them. This is followed by Craig Smith's "Defect Prevention: The Road Less Traveled," a useful case study of one company's successful attempt to improve product quality by creating a culture of defect prevention.

Project management

In the introduction to this section, Editor Taz Daughtrey proposes that next to the Statue of Liberty should stand a companion Statue of Responsibility, as a reminder of the balance between freedom and accountability. In the software industry, as in other fields, he maintains, there is now general agreement about what constitutes acceptable practice, and therefore also about what represents deviation from acceptable practice, or malpractice. The idea of software malpractice claims is sobering, and reminds us of how far we are from a state of perfection.

"Initial Experiences in Software Process Modeling" by Ray Madachy and Denton Tarbet presents examples of how a project manager used metrics and mathematical models to make decisions about the optimal size for his project team, whether to share people among different tasks, how much code to reuse, and so on. On the flip side, Alan Weimer and R. Jack Munyan remind us how important it is to pay attention to "people issues" on a software project in "Recipe for a Successful System: Human Elements in System Development." Different though these two articles are, both have much practical advice to offer managers, and this is one of the most valuable sections in the book.

Measurement

William Florac and Anita Carleton's "Using Statistical Process Control to Measure Software Processes" begins this section by applying W. E. Deming's statistical quality control methods to software development in order to ensure software stability and capability. For example, Shewart control charts that are commonly used to record and analyze metrics about the physical characteristics of manufactured objects can be used equally well to represent defect discovery rates and resolution time in a software development project. "Managing with Metrics: Theory into Practice" by Denis Meredith is a case study of a large data-scanning project that used metrics to ensure a successful result, despite such challenges as a short schedule and a ban on maintenance releases once the software was deployed. Particularly useful are specific examples of metrics and charts the project manager used. Next, "Experiences Implementing a Software Project Measurement Methodology" by Beth Layman and Sharon Rohde describes the US Department of Defense's Practical Software Measurement program, explaining how it helps identify critical issues on a project and collects metrics to see how aspects of the project affect those issues. It includes examples of how the information was used in decision making and discusses lessons learned.

Inspection and testing

This section begins with an interesting but somewhat inaccessible article: Tom Gilb's "Planning to Get the Most Out of Inspection." He assumes that the reader has already read his book (I had not) in discussing document inspection as a way to prevent defects in a product. In other words, by discovering defects in project documents -- requirements specifications, design documents, and so forth -- you can prevent people from implementing these problems in the software. The next article, "A Testing Maturity Model for Software Test Process Assessment and Improvement," is on my short list of favorites because it adds so much to our collective body of practical engineering knowledge. Most software engineers are familiar with the Software Capability Maturity Model (CMM), but quality specialists recognize that this model does not address some of the processes and practices specific to testing. A complement to the CMM called the Testing Maturity Model (TMM) is proposed here by Ilene Burnstein, Ariya Homyen, Taratip Suwanassart, Gary Saxena, and Rob Grom. The TMM defines testing-related maturity goals, activities, tasks, and responsibilities that correspond to each of the CMM's five levels. Burnstein *et al.* also propose a TMM Assessment Model to measure the TMM level of one's own project. The final article in this section, "Choosing a Tool to Automate Software Testing" by Mark Fewster and Dorothy Graham, looks at ways to decide what software to buy. Anyone charged with acquiring a testing tool will find this a very good resource for learning what questions to ask, what to test and look for in software tools, how to predict how well a tool will work in-house, and how to work with vendors, get the most out of demos and trials, and make a final decision.

Audits

In this section, Daughtrey points out that, although many books cover the subject of auditing, most of them are not specific to *software* audits. "Quality Evaluation of Software Products" by Jørgen Bøegh picks up where the previous article left off: instead of evaluating a tool for purchase, Bøegh's task is to systematically evaluate the quality of a product to determine whether or not it fulfills specified quality requirements. As Bøegh points out, software is increasingly used for such life-critical systems as traffic control, robotics, and medical systems, in which software defects can have extreme consequences. He lists the relevant ISO and IEC standards for software product evaluation and testing, goes on to present several methods for objectively evaluating software quality, and concludes with a real-world example of evaluating software for a fire alarm system. The second article on auditing, "Practical Quality Assurance for Embedded Software" by Erik P .W. M. van Veenendaal, is a case study of how inspections successfully detected and prevented defects in software for a television. By supplementing his company's ISO-9001 procedures with nine person-weeks of inspection time, his team found more than 1,400 defects and was then able to ship a high-quality product. The chapter includes detailed, clear instructions on how to conduct and apply the results of such inspections.

Configuration management

This final section of the book begins with "Software Configuration Management for Project Leaders" by Tim Kasse and Patricia A. McQuaid. They take a broad view of configuration management (CM), defining it not only as source code version control but also as a methodology for controlling change in all software project artifacts -- from design specifications and data files to test procedures and user documentation. CM, they contend, is a critically important process for producing and delivering software in a controlled manner, and they provide many specific details about CM implementation. The final article in the book, "Applying Quantitative Methods to Software Maintenance" by Ed Weller, explains how to use measurement results to make good decisions, and also holds up some commonly-held beliefs for quantitative scrutiny. On one three-year project, for example, the engineers on his team believed that if they were to push bug fixes out faster, they would introduce more *new* bugs in the process. But when Weller compared the recidivism rate (percent of new bugs) with time spent on fixing bugs, he found no correlation. Throughout the project he improved his group's processes by conducting similar quantitative analyses, and this chapter presents some good lessons for both project managers and process engineers.

Daughtrey ends the book with a reminder that there are triumphs as well as problems in software development projects: Everyone hears about a few well-publicized failures, but many successes go unnoticed. For those of us who spend our days looking for hidden problems in every apparently-functioning bit of software, this is a good reminder. The goal of all our efforts, after all, is to produce something that works.

On the whole, this book presented many useful ideas and techniques; rarely did I find myself muttering "No way, not in my project!" as I read. The collection of articles also accurately reflects the state of the art of software quality engineering. Note that I am not using *state-of-the-art* in the colloquial sense, as a descriptor for bleeding-edge innovation, but rather to refer to the true state of our current practices, warts and all. Software quality engineering is a discipline that is still in the process of maturing. And I recommend this book for anyone who wants to see a compact but detailed snapshot of what is going on in that particular corner of the software development world.

-**Susan McVey**
Software Quality Engineer
Rational Software
IBM Software Group

Notes

1 A copy of the CSQE Body of Knowledge is included as an Appendix to the book. Readers who are interested in software quality engineering certification can find out more at http://www.asq.org.

For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!