Book Review: The Economics of Software Quality

Posted by **BenLinders** on Wednesday December 14
from the read-all-about-it dept.

BenLinders writes: "The book *The Economics of Software Quality* provides solutions to quantify Software Quality, helping you to manage software development and maintenance. It contains software quality data that you can use to build a business case to improve the quality of your software, and decide upon processes and techniques that can help to implement the needed improvements in your organization.

	The Economics of Software Quality
author	Capers Jones and Olivier Bonsignour
pages	587
publisher	Addison-Wesley
rating	8/10
reviewer	Ben Linders
ISBN	978-0-13-258220-9
summary	To build your Business Case for Software Quality Improvement

Quantifying software quality is not an easy thing. Several measurements exist, for instance estimating and tracking the number of defects that are found (both within development/maintenance and from customers), measuring software quality with static analysis tools (complexity, fan in/fan out), or measuring the effectiveness of software development methods and techniques (like inspections, test, and Cost of Poor Quality). This book covers Software Quality Factors that influence the quality of software products as perceived (and believed!) by customers. An extensive list of factors is provided, where the authors have selected those factors that they consider most significant to achieve quality.

Many software development processes and techniques are covered in this book, from a quality and economic point of view. This also includes agile methods, where a body of data is available about the effects of agile techniques like user stories, Test Driven Design, Scrum Sessions, Measuring Technical Debt, and Pair Programming. For instance, about agile user stories the book states "... the user story method seems to be concise and fairly trouble-free, User stories average below 0.5 pages per function point and seem to contain fewer than 0.5 defects per function point`. This kind of information can be very helpful to build a business case for using agile methods in your organization.

Most of the data on software quality that the book provides is in "Defects per Function Point". A backfiring table is also provided, to translate language statements/lines to function points. So if you are not using function point, but programming in Java, Ruby, C++ or any other popular programming language, the data can still be used.

There is a full chapter covering defect prevention. Methods like Reuse, Formal Inspections and Quality Function Deployment are the most effective in preventing defects, and also techniques like Root Cause Analysis and PSP/TSP are claimed to be very effective. Given that the top ten techniques reduce defects with 40% - 85%, makes it interesting for many organizations to investigate the business case to improve the quality of their products, using these methods and techniques.

Additional information is provided on how to measure the effects on quality from a given method or technique. The book also provides warning for quality measurements that can be unreliable. An example is measuring cost-per-defect. When the quality of your development activities increases, for instance by improving requirements practices and implementing defect prevention for design and coding, the number of defects that testing finds will go down. Since test case preparation is a fixed cost, the cost per defect for testing will go up when the software has fewer defects. This makes such a measurement potentially unreliable. I believe that the main benefits will come when you can reduce your testing activities, based upon measurements that quantify the quality of your products before testing starts. Techniques like risk based testing can also reduce your testing hours, thus saving time and money on tests that are not needed.

Defects measurements and tracking are used in more then 55% of the military and defense software applications (using CMMI, TSP, QFD, etc), but in less then 15% of IT, commercial, web or embedded applications. Given their prevention effectiveness of -35%, and removal effectiveness of 25%, it is still surprising to me that this is not used more often. The data needed for these kinds of measurements is usually available in the defect management systems, though some addition effort is needed to classify defects and to do Root Cause Analysis. The benefits of using these kinds of measurements, combined with estimations of the expected quality at release, to decide and steer software development and prevent defects during the development and before release are significant.

The book also gets into methods to quantify structural quality issues that are not exactly “defects” but have an important impact – “Technical Debt” being one of these methods of quantification. These kind of measurements help to manage the quality of your code base, being able to see the impact on quality from changes, and take action to get quality back on the desired level when needed.

Reviews and inspections are very effective ways to remove defects before testing. Several techniques are described, both informal and formal techniques. Several of them are also usable within agile methods, supporting teams in developing better quality software. Applying these techniques effectively requires training, and arrangements within your company that enable employees to use them. The book makes clear that if you want to reduce post release defects and lower your maintenance costs, the work needs to start with early software development activities, like using better techniques for managing requirements, software modeling and design, reviews and inspections, and automatic code analysis. Testing alone is not sufficient to improve quality, and is also very costly.

The relationship between quality and risks is also explored. Many major software problems are related to the quality of the software products, e.g. outages, data loss, security issues or regulatory non compliances. Investigating such issues, for instance with Audit or Root Cause Analyses, and taking action to prevent similar problems in the future can be essential for your business. Measuring the losses and estimating potential benefits from preventive actions helps you to select the right improvements, and acquire commitment and funding to implement them.

The capabilities and skills of the staff that develops the software have significant impact on the quality. The benefits of training, skill development, and sharing of experiences to develop a learning organization can be huge. Software methods like Agile and RUP include mechanisms to continuously evaluate, learn and improve the capabilities of your staff. E.g. using retrospectives and scrum boards, to identify and follow up with improvement actions.

Overall the book covers the economic perspective of quality. The information provided can be overwhelming for some readers. If you need to improve your product quality, and are limited in time and money to do it, this book helps you to select effective quality methods and techniques, and to measure and track your progress when implementing improvements.

Ben Linders is a specialist in quality, process improvement and organizational development.