ISSN 1512-1232

SOFTWARE QUALITY PATHWAYS: ISSUES AND STRATEGIES

Dr.S.S.Riaz Ahamed

Principal, Sathak Institute of Technology, Ramanathapuram,India.
Email:ssriaz@ieee.org, drriaz@gmail.com

Abstract
Quality is the ongoing process of building and sustaining relationships by assessing, anticipating, and fulfilling stated and implied needs. Quality represents the properties of products and/or services that are valued by the consumer. Quality is a momentary perception that occurs when something in the environment interacts with human factor, in the pre-intellectual awareness that comes before rational thought takes over and begins establishing order. Judgment of the resulting order is then reported as good or bad quality value. A product or process that is Reliable, and that performs its intended function is said to be a quality product.

Keywords: *Total Quality Management (TQM), Formal Technical Review (FTR),*

1. INTRODUCTION

The quality of design and the quality of conformance must be blended. If the design is fine but its implementation slack, the final product oozes low quality.

Two types of software quality can be distinguished:

- External Quality and
- Internal Quality.

External quality is that which can be seen by customers and which is traditionally tested. Bad external quality is what can be seen: system crashes, unexpected behavior, data corruption, slow performance. Internal quality is the hidden part of the iceberg, i.e. program structure, coding practices, maintainability, and domain expertise. Bad internal quality will result in lost development time, fixes are likely to introduce new problems and therefore require lengthy retesting. From a business point of view, this will invariably result in loss of competitiveness and reputation. External quality is a symptom whereas the root problem is internal quality. Poor internal quality leads to high maintenance costs. In order to improve software quality, internal quality must be improved. If software carries out the functional requirements and performance needs and works within the development standards that are documented, then that software is of high quality. If the software departs from a defined and understood development benchmark, low quality software will show up. If the software does these two functions but in the process of applying the software is user-unfriendly or its upkeep is complex and out of the way, that software has only a low-grade. These are unwritten rules compliance with which, carry high value in the minds of the users.

2. SOFTWARE QUALITY AND QUALITY ASSURANCE.

Total Quality Management is applicable for computer software. TQM calls for continuous process improvement. The aim is to make the process transparent and visible and thereby to dispel the sense of secrecy that may surround the development of the software. TQM expects that the processes that are under development must be repeatable and should not be for a one-time make. The process must be in some measurable form and so needs some sort of calibration.

ISSN 1512-1232

In the second phase of TQM the software Quality Process is subjected to an overall analysis and specifically looks at the intangibles that are related to it. Here the idea is to optimize the impact of these intangibles in the process. Continuity and stability of the organizational structure can bring about substantial improvements in the quality of the software. You can easily guess that a dedicated team can produce reliable software products whereas a team that is changing quite often due to exit and entry of staff can produce results of lower quality and that too by taking longer time.

This paper previews a two-stage approach to transforming quality control to quality assurance:

- *Quality Assessment* (Stage 1) provides objective quantitative evaluation of quality while identifying and fixing software defects early in the development cycle, before they get to the customer and even before testing;
- *Quality Assurance* (Stage 2) helps prevent defects from entering the code base by enforcing coding standards during, and by preventing new defects from occurring during times of change through accurate impact analysis.

STAGE 1: Quality Assessment

Establish an information model

Establishing an information model requires the following steps:

1- Build and deliver the database that captures all software entities, their attributes and inter-relationships, together with software artifacts such as tests and documentation.
2- Synchronize the model build process with the software build process itself.

Establish standards for software development: standard quality filter sets (QFS)

Identify and define industry-standards for a variety of categories such as *programming constructs, software structure, adherence to globalization standards, portability, statistics, metrics,* time and date constraints, etc.

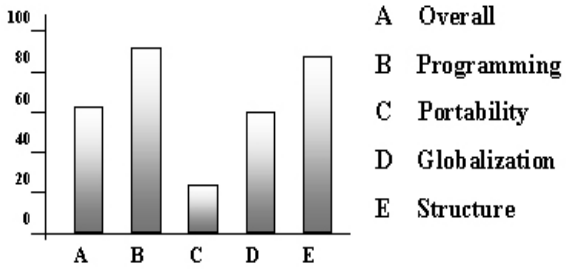
As an example of standard, we consider Programming Constructs, which measure the use of questionable programming constructs that may be unsafe or may conflict with site-specific programming guidelines. Use of questionable programming constructs adds to overall complexity of the code base, compromises security and makes the code less manageable and modular than intended. Examples include *Empty statement bodies, functions using global data, functions that return a pointer to a local stack, potential memory leak,* etc.

Assess initial quality and identify defects

These standards may now be applied to create a quantitative assessment of an application in statistical and graphical form plus identify a list of construct violations. The Quantitative baseline may now be used to measure improvements in product quality and for comparison to other software development organizations.

Quality assessment can be represented graphically using a chart of quality indexes. The Figure shows the overall quality index together with the quality indexes of four areas of examination: programming constructs, portability, globalization and structure. Quality indexes approaching 100 are representative of near ideal quality whereas low quality indexes are representative of dangerously low software quality. The higher the software quality the lower the risk, the sooner the products will be out to market, and the fewer the defects.

ISSN 1512-1232



Defect repair

Once the defects have been detected, their repair is facilitated by quick identification of defect location. DIS systems provide easy navigation and querying of the code base.

Monitor quality

Monitoring of software quality is enabled by regularly generated quality assessments and custom trend chart generation, providing assessment of trends over time.

STAGE 2: Quality Assurance

Impact analysis

The understanding of the complex relationships between all the entities in the code base requires not only skill but also extensive experience with the specific source code. Even the best and most experienced engineers can make mistakes in their analysis of changes to the source code. For example, if an engineer changes only a single line of source in a function, there may be 15 other places in the code base that must be examined and may require related changes. Even if the developer remembers 14 of the 15 other places in the code to check, it is the one instance overlooked that can cause the code submission to fail. Such a failure impacts not only the individual developer but may very well impact the entire engineering team, or even the entire company, waiting on a successful build or bug-free release.

Submission check

Checking code for design flaws and coding defects before it is submitted to the shared source base is widely acknowledged as a good thing to do. The promise of properly conducted code reviews is that they can be effective in improving quality of software. Poor coding constructs can be identified and eliminated before they add complexity to the product. Coding defects can be found and eliminated before they enter and pollute the shared source base.

However, the reality of code reviews usually does not match their promise. Because code reviews are labor intensive and require scarce senior developers, they are viewed by many developers as painful and a waste of time. Code reviews are often not done with the level of attention and energy to be thorough and complete. Because of this reality, code reviews are done on an irregular basis and are frequently ineffective.

SQA work

There are two sets of work groups who are at the center of SQA. Naturally the first is the team of software engineers whose job is to lay solid technical foundations for the success of the software product. They conduct periodical reviews, tests and improve the techniques.

ISSN 1512-1232

The second is an independent but related group (Group) who assist in very many ways to enable the software team (Team) come out with quality products. They look at what the Team has planned and examine whether their plan of work fits in the standards that govern the making of the software. For instance the Group finds whether the planning for quality is right, how records can be kept, how the work in progress can be monitored or supervised and how the deficiencies or shortcomings can be conveyed to the Team. The Team in the main is concerned with how to deliver a product that satisfies the requirements set before it. The Group is concerned with ensuring that the product is made in accordance with the generally accepted standards (ISO 9001) and is in alignment with the internal policies of the firm and the external standards of the industry.

Software Reviews

Software engineering needs periodical reviews that act as filters and remove the dirt from the engineering process. When an external examination is conducted of the so-far-developed programme, it may be possible to detect some errors. It is natural that a person overlooks the errors he has made and locates the errors others have made. That is why some external examination is required. Review stages help to know whether any modifications or turning points are needed in the software work in progress. Furthermore reviews help in determining whether the software work is going on in a wholesomely uniform standard and that parts of it are not low grade. Quality maintenance can be improved by reviews.

Format Technical Review

The Formal Technical Reviews: A formal technical review (FTR) is a software quality activity that is performed by software engineers. The objectives of FTR are

(1) to cover errors in function, logic, or implemenation for any representation of the software;
(2) to verify that the software under review meets its requirements;
(3) to ensure that the software has been represented according to predefined standards;
(4) to achieve software that is developed in a uniform manner
(5) to make projects more manageable.

The FTR is actually a class of reviews that include walkthroughs, inspections, round-robin reviews, and other small group technical assessments of software. Each FTR is conducted as a meeting and will be successful only if it properly planned, controlled and attended.

An informal review takes place when two or three staff talk about the software techniques. A formalized structural presentation to users, management and others is another type of review. We term this formal technical review (FTR). FTR pinpoints defects and faults. You may have already gone through some distance in the software work. FTR in essence locates mistakes and errors so that they do not spill into the next stage with magnification and become defects or faults.

Obviously FTR is able to find out and correct errors early. The eradication of a large body of errors at an early stage of FTR therefore enhances the quality. FTR is primarily meant to detect to find mistakes in function logic or other connected activities. So one of the essentials of the FTR is that it must have software engineers and personnel from other disciplines. Apart from these the FTR meeting can determine whether the preplanned designing is being adhered to or not. Overall it can contribute to the manageability of the software work in progress.

An FTR with about 3/5 staff members and others, conducted with adequate preparation beforehand, and lasting a reasonable time is what is wished. Or else the FTR may be mere formality and without producing results. FTRs should not lapse into this category.

ISSN 1512-1232

An FTR is needed when a unit or an assembly or part of the software packet is complete. The product shaper should inform the pith and substance of the product and should welcome comments so as to improve the quality. Surely there must be minutes of the meeting for review incorporating the subject matter of the review, who reviewed it and the final findings and conclusions. A one-page summary can be prepared with annexes of details for a quick look. This report can be distributed to the software engineers so that they will profit by it for their future design plans. The FTR must be followed by corrective action as otherwise the purpose of FTR gets muted.

At the end of the review, they must decide whether to: 1) accept the work product without further modification, (2) reject the work product due to severe errors (once corrected, another review must be performed) or 3) accept the work product provisionally (minor errors have been encountered and must be corrected, but no additional review must be performed). The decision made, all FTR attendees complete a sign-off, indicating their participation in the review and their concurrence with the review team findings.

Review Reporting and Record keeping: During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting and a review issues list is produced. In addition, a simple review summary report is completed. A review summary report answers three questions:

- What was reviewed?
- Who reviewed it?
- What were the findings and conclusions?

Guidelines

FTR meetings are to go on in alliance with the guidelines that are agreed upon. If a meeting is to go on smoothly there must be some focus and a method.

1. It has to be borne in mind that the subject matter of the review is the product and not the person who created it or shaped it. Products that are presently considered to be excellent have been produced only by gradual improvements and additions made in response to review reports.
2. The atmosphere in the meeting should be pleasant cooperative and contributory. Errors and mistakes should not be bluntly stated so as to spoil the soft tenor of the meeting.
3. There must be an agenda drawn up so that the participants in the meeting can stick to the point at hand and not talk randomly out of the context.
4. When points become contestable and points and counterpoints are canvassed care should be taken to see that the meeting does not slip into a controversy. If discussions indicate animosity, such discussions should be ended or at any rate limited by tact. It is quite possible that the problems are detected but solutions to the problem are not offered. The errors and defects can be solved only by discussions by the programmer with a couple of others. That solution may come about later when the problem pointed out lingers in the mind of the software engineers.
5. In the meeting the number of persons participating should be reasonable. While nobody denies that every body can contribute to the quality enrichment and help fixing errors, if the number of participants in the review meeting increases beyond a reasonable figure, the quality of the meeting itself gets diluted. Instead of addressing the particular, the focus would slide into the general. It should be made clear that adequate pre-meeting preparation is a must for the participants because technical problems cannot be presented and resolved off hand.
6. A checklist or pro-forma for each product or integral part of a product is to be drawn up, as that will help the channeling of discussions in the desired direction. Such a checklist can reduce unwanted collateral impact.
7. An FTR meeting will suggested modifications, amendments and course changes. There must be adequate time reserved for carrying out these changes. So fixation of deadlines for delivery of the software product to the end-user must take into account the expenditure of such extra time.

ISSN 1512-1232

8. Reviewers are software professions and persons from other specialised fields. All should have a grounding in the psychology of meetings. These can be imparted by suitable briefing and that can greatly enhance the utility of the FTRs.
9. If possible the modus of the FTR itself can be discussed and a norm established. Having established a framework if changed situations warrant it has to be improved upon thus providing dynamism.

3. CONCLUSION

By quality of conformance we mean the extent to which the designed specifications are adhered to during making the product. In other words, if the design specifications are set high, but in practice, those set are breached the product performance is sure to suffer thus leading to low quality because the quality of conformance is low. Quality is a key measure of project success. It is what a customer remembers in the long run. High-quality products result in customer satisfaction, while poor quality results in customer dissatisfaction. Software quality factors cannot be measured because of their vague definitions. It is necessary to find measurements, or metrics, which can be used to quantify them as non-functional requirements. For example, reliability is a software quality factor, but cannot be evaluated in its own right. However, there are related attributes to reliability, which can indeed be measured. Some such attributes are mean time to failure, rate of failure occurrence, and availability of the system. Similarly, an attribute of portability is the number of target-dependent statements in a program. To produce a good-quality product, it is essential to clearly define the quality requirements of a product. In addition, we need to plan and perform a systematic set of activities called software quality assurance (SQA) and use ‘quality filters’ such as formal technical reviews (FTRs) for detecting errors.

ISSN 1512-1232

4. REFERENCES

1) David I. Cleland, Roland Gareis (2006). Global project management handbook. McGraw-Hill Professional, 2006. ISBN 0071460454. Pp.1-4.
2) Martin Stevens (2002). Project Management Pathways. Association for Project Management. APM Publishing Limited, 2002.
3) Morgen Witzel (2003). Fifty key figures in management. Routledge, 2003. ISBN 0415369770. Pp. 96-101.
4) Bjarne Kousholt (2007). Project Management –. Theory and practice.. Nyt Teknisk Forlag. ISBN 8757126038. p.59.
5) F. L. Harrison, Dennis Lock (2004). Advanced project management: a structured approach. Gower Publishing, Ltd., 2004. ISBN 0566078228. p.34.
6) Stellman, Andrew; Greene, Jennifer (2005). Applied Software Project Management. O'Reilly Media. ISBN 978-0-596-00948-9.
7) Diomidis Spinellis. Code Quality: The Open Source Perspective. Addison Wesley, Boston, MA, 2006.
8) Ho-Won Jung, Seung-Gweon Kim, and Chang-Sin Chung. Measuring software product quality: A survey of ISO/IEC 9126. IEEE Software, 21(5):10–13, September/October 2004.
9) Stephen H. Kan. Metrics and Models in Software Quality Engineering. Addison-Wesley, Boston, MA, second edition, 2002.
10) Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, IEEE Computer Society Press, 2005, ISBN: 0471713457.
11) Musa, J.D, A. Iannino, and K. Okumoto, Engineering and Managing Software with Reliability Measures, McGraw-Hill, 1987
12) Pressman, Scott (2005), Software Engineering: A Practitioner's Approach (Sixth, International ed.), McGraw-Hill Education.
13) Albert Hamilton (2004). Handbook of Project Management Procedures. TTL Publishing, Ltd. ISBN 07277-3258-7.
14) Edward Kit, Software testing in the real world, Addison-Wesley publications, 2000, ed.1
15) Richard Fairly, Software engineering concepts, McGraw-Hill Inc.,1985
16) Myers, Glenford J. (1979). The Art of Software Testing. John Wiley and Sons. p. 145-146. ISBN 0-471-04328-1.
17) Barry W. Boehm, Software Engineering Economics, Prentice-Hall Inc., 1981.
18) Dustin, Elfriede (2002). Effective software Testing. Addison Wesley. p. 3. ISBN 0-20179-429-2
19) Pankaj Jalote, An integrated approach to software engineering, Narosa publications, 1997, ed. 2
20) Shari Lawrence Peleeger, software engineering theory and practice, Pearson education, 2001, ed. 2

Article received: 2011-01-16