ISSN 1512-1232

THE IMPACT OF FORMAL APPROACHES TO SOFTWARE QUALITY ASSURANCE

Dr.S.S.Riaz Ahamed

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

Abstract

Software quality is a composite work and can be enriched by high level analysis, design structure, encoding, testing to be followed with a properly structured FTR and corrective action. This has become a conventional practice. New voices are being heard that quality principles can be factored into a model and that can turn out a broad spectrum of indices and metrics. In turn these indices and metrics can measure either explicitly or implicitly the quality of the software. The quality assurance activity is the process of verifying that these standards are being applied. In small projects this could be done by the development team, but in large projects specific staff should be allocated to the role.

Keywords: *Software Reliability Engineering (SRE), Software Quality Assurance Plan (SQAP)*

1 INTRODUCTION

Software Reliability is the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems. "Software Reliability Engineering (SRE) is a standard, proven best practice that makes testing more reliable, faster, and cheaper. It can be applied to any system using software and to frequently-used members of software component libraries."

Measures of Reliability

In measuring reliability of software the old timers applied the principles of measuring hardware reliability. They transplanted the hardware reliability measurement standards on to the software. But that was a mismatch. Though a mismatch, there is a good deal of discussion still going on about the applicability of hardware reliability techniques to software. Hardware failures are prompted more by wear and tear and the impact of environmental factors like heat, dust etc. and less by design faults. The obverse is true in the case of software

It has been stated that in a compute system the measure of reliability is MTBF and

MTBF = MTTF + MTTR
Where MBTF = Mean Time Between Failure
MTTF =Mean Time To Failure
MTTR = Mean Time To Repair

Rate of Reliability.

A point worth noting in the field of software reliability is that a software program may contain within its ambit several errors. The end-user is not interested in the total count of the number of errors and cannot feel satisfied because the number of errors say is only 5. Each time he runs the program if the error produces failure the utility is nil. In other words each error does not produce the same failure rate. Some errors may remain dormant because that part of the program containing

ISSN 1512-1232

errors is not used quite often and hence looks to the user to be failure free. Only on the rare occasions when it is used it will end up in failure. The existence of errors and the rate of failure therefore do not carry a pro-rata link. Such errors may take longer MTBF to surface; perhaps several decades.

Consider that an error of this type triggers a failure once in 30 years. You are moving heaven and earth to eradicate all these types of errors. Obviously that will not improve MTBF in any sizable way.

Rate of Availability

Is the software available when you want it? Or is it failing when you need it badly? The measure of availability is stated to be

Availability =[MTTF/(MTTF+MTTR)] * 100%

This supplies us with the information that if the MTTR is low the software can be maintained by consuming little repair time – the maintainability is easy.

Software Safety.

This is a quality that tries to locate the potential hazards that affect the software adversely and prompts an entire system to go dead. Such risks are major and critical. For example a computer-controlled aircraft landing system may

- Not respond to command for gradual reduction in height as the ground level is being approached
- Not reduce the speed after contact is established with the ground by the tyres of the aircraft
- Not decelerate when needed but instead may accelerate at that moment.

When these top system level performance hazards are identified, they must be dealt with the respect they deserve. Even though these may occur very occasionally, the intensity of the impact on life is patent. The software is not safe and triggers mishap. Thus software safety is not the same thing as software reliability. In reliability you can statistically determine or calculate the probability of the software turning mischievous and not obeying commands. The system fails to perform or deliver goods. That is all. There is no danger or mishap. But in software safety, a failure turns into a mishap and produces loss of life or property as in the example.

2 SQA

Software Quality Assurance (SQA) is `a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements' (ANSI/IEEE Std 730.1-1989). Software Quality Assurance is synonymous with Software `Product Assurance' (PA) and the terms are used interchangeably in these Standards.

The Software Quality Assurance Plan (SQAP) defines how adherence to these standards will be monitored. The SQAP contents list is a checklist for activities that have to be carried out to assure the quality of the product. For each activity, those with responsibility for SQA should describe the plans for monitoring it.

Activities

Objective evidence of adherence to these standards should be sought during all phases of the life cycle. Documents called for by this standard should be obtained and examined. Source code should be checked for adherence to coding standards. Where possible, aspects of quality (e.g.

ISSN 1512-1232

complexity, reliability, maintainability, safety, number of defects, number of problems, number of RIDs) should be measured quantitatively, using well-established metrics.

Subsequent sections list activities derived from ANSI/IEEE Std 730.1-1989 that are necessary if a software item is to be fit for its purpose. Each section discusses how the activity can be verified.

Management

Analysis of the managerial structure that influences and controls the quality of the software is an SQA activity. The existence of an appropriate organisational structure should be verified. It should be confirmed that the individuals defined in that structure have defined tasks and responsibilities. The organisation, tasks and responsibilities will have been defined in the SPMP.

Documentation

The documentation plan that has been defined in the SPMP should be analyzed. Any departures from the documentation plan defined in these standards should be scrutinized and discussed with project management.

Standards, practices, conventions and metrics

Adherence to all standards, practices and conventions should be monitored.

Reviews and audits

These Standards call for reviews of the URD, the SRD, the ADD, the DDD, the SVVP and the SCMP. It also calls for the review and audit of the code during production. The review and audit arrangements described in the SVVP should be examined. Many kinds of reviews are possible (e.g. technical, inspection and walkthrough). It should be verified that the review mechanisms appropriate for the type of project. SQA personnel should participate in the review process.

Testing activities

Unit, integration, system and acceptance testing of executable software is essential to assure its quality. Test plans, test designs, test case, test procedures and test reports are described in the SVVP. These should be reviewed by SQA personnel. They should monitor the testing activities carried out by the development team, including test execution. Additionally, other tests may be proposed in the SQAP. These may be carried out by SQA personnel.

Problem reporting and corrective action

The problem handling procedure described in these standards is designed to report and track problems from identification until solution. SQA personnel should monitor the execution of the procedures, described in the SCMP, and examine trends in problem occurrence.

Tools, techniques and methods

These Standards call for a tools, techniques and methods for software production to be defined at the project level. It is an SQA activity to check that appropriate tools, techniques and methods are selected and to monitor their correct application.

SQA personnel may decide that additional tools, techniques and methods are required to support their monitoring activity. These should be described in the SQAP.

Code and media control

These Standards require that the procedures for the methods and facilities used to maintain, store, secure and document controlled versions of the identified software, be defined in the SCMP. SQA personnel should check that appropriate procedures have been defined in the SCMP and carried out.

ISSN 1512-1232

Supplier control

Software items acquired from external suppliers must always be checked against the standards for the project. An SQAP shall be produced by each contractor developing software. An SQAP is not required for commercial software.

Records collection, maintenance and retention

These standards define a set of documents that must be produced in any project. Additional documents, for example minutes of meetings and review records, may also be produced. SQA personnel should check that appropriate methods and facilities are used to assemble, safeguard, and maintain all this documentation for at least the life of the project. Documentation control procedures are defined in the SCMP.

Training

SQA personnel should check that development staff are properly trained for their tasks and identify any training that is necessary. Training plans are documented in the SPMP.

Risk management

All projects must identify the factors that are critical to their success and control these factors. This is called `risk management'. Project management must always analyse the risks that affect the project. Their findings are documented in the SPMP. SQA personnel should monitor the risk management activity, and advise project management on the methods and procedures to identify, assess, monitor, and control areas of risk.

3 THE SOFTWARE QUALITY ASSURANCE PLAN

Software Quality Assurance Plan for the SR phase

1 Purpose
2 Reference Documents
3 Management
4 Documentation
5 Standards, practices, conventions and metrics
  5.1 Documentation standards
  5.2 Design standards
  5.3 Coding standards
  5.4 Commentary standards
  5.5 Testing standards and practices
  5.6 Selected software quality assurance metrics
  5.7 Statement of how compliance is to be monitored
6 Review and audits
  6.1 Purpose
  6.2 Minimum requirements
7 Test
8 Problem reporting and corrective action
9 Tools, techniques and methods
10 Code control
11 Media control
12 Supplier control
13 Records collection, maintenance and retention
14 Training
15 Risk Management

ISSN 1512-1232

16 Outline of the rest of the project

4 FORMAL APPROACHES TO SQA

Proof of Correctness

- Treat a program as a mathematical object.
- Developed with a language with a rigorous syntax.
- Are attempts at developing a rigorous approach to specification of software requirements.
- With both can attempt to develop a mathematical proof that a program conforms exactly to its specification.
- In the code, can at selected statements formulate assertions on the set of correct values for program variables.
- Can then show that the statements between these assertions do the correct transformation of the values in the assertions.

Statistical Quality Assurance

In statistical quality assurance:

1. Information about software defects is collected and categorised.
2. An attempt is made to trace each defect to its underlying cause (e.g., not conforming to the specification, design error, violation of standards, poor communication with customer ...)
3. Using the 'Pareto principle' (80% of defects can be traced to 20% of all possible causes), isolate the 20% of causes (the "vital few").
4. Once the "vital few" causes have been identified, correct the problems that have caused the defects.

The Cleanroom Process

- Use statistical quality control and formal program verification.
- Attempt to prevent defects rather than find defects.
- In projects attempted so far with this method (size between 1000 and 50,000 LOC), 90% of all defects were found before the first execution tests were conducted.
- Has not been widely applied in industry.
- Requires significant change in both management and technical approaches to software development.

5 CLEANROOM METHODOLOGY

The objective of the Cleanroom methodology is to achieve or approach zero defects with certified reliability. As described by Hausler (1994), the Cleanroom methodology provides a complete discipline within which software personnel can plan, specify, design, verify, code, test and certify software. In a Cleanroom development, correctness verification replaces unit testing and debugging. After coding is complete, the software immediately enters system test with no debugging. All test errors are accounted for from the first execution of the program with no private testing allowed. As opposed to many development processes, the role of system testing is not to test in quality; the role of system testing is to certify the quality of the software with respect to the systems specification. This process is built upon an incremental development approach. Increment N+1 elaborates on the top down design of increment N . The Cleanroom process is built upon function theory where programs are treated as rules for mathematical functions subject to stepwise

ISSN 1512-1232

refinement and verification. Cleanroom specifications and designs are built upon box structure specifications and design. Box structure specifications begin with a black-box specification in which the expected behavior of the system is specified in terms of the system stimuli, responses and transition rules. Black boxes are then translated into state-boxes which define encapsulated state data required to satisfy black box behavior. Clear box designs are finally developed which define the procedural design of services on state data to satisfy black box behavior. Team reviews are performed to verify the correctness of every condition in the specification. During the specification stage an expected usage profile is also developed, which assigns probabilities or frequency of expected use of the system components. During system correctness testing, the system is randomly tested based on the expected usage of the system. In this process, software typically enters system test with near zero defects. The Cleanroom process places greater emphasis on design and verification rather than testing. In this process errors are detected early in the life cycle, closer to the point of insertion of the error.

6 CONCLUSION

Quality Assurance is possible only if quality products are turned out. In turn it is helped by auditing and reporting. Needless to say mere auditing and reporting will not produce quality products. Periodical and many a time continuous auditing and reporting systems are basic tools. The management has access to some data anyway. But will it enable the management to feel the standard and quality of the software product that is under process? Audit and report supply the management with what we may call as post-inspection data so that the management can feel confident that the product quality is up to the goals it is set to meet. 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.

7 REFERENCES

1. Pressman, Scott (2005), Software Engineering: A Practitioner's Approach (Sixth, International ed.), McGraw-Hill Education.
2. David I. Cleland, Roland Gareis (2006). Global project management handbook. McGraw-Hill Professional, 2006. ISBN 0071460454. Pp.1-4.
3. Martin Stevens (2002). Project Management Pathways. Association for Project Management. APM Publishing Limited, 2002.
4. Morgen Witzel (2003). Fifty key figures in management. Routledge, 2003. ISBN 0415369770. Pp. 96-101.
5. Bjarne Kousholt (2007). Project Management –. Theory and practice.. Nyt Teknisk Forlag. ISBN 8757126038. p.59.
6. F. L. Harrison, Dennis Lock (2004). Advanced project management: a structured approach. Gower Publishing, Ltd., 2004. ISBN 0566078228. p.34.
7. Stellman, Andrew; Greene, Jennifer (2005). Applied Software Project Management. O'Reilly Media. ISBN 978-0-596-00948-9.
8. Albert Hamilton (2004). Handbook of Project Management Procedures. TTL Publishing, Ltd. ISBN 07277-3258-7.
9. Edward Kit, Software testing in the real world, Addison-Wesley publications, 2000, ed.1
10. Pankaj Jalote, An integrated approach to software engineering, Narosa publications, 1997, ed. 2
11. Shari Lawrence Peleeger, software engineering theory and practice, Pearson education, 2001, ed. 2

ISSN 1512-1232

12. Richard Fairly, Software engineering concepts, McGraw-Hill Inc.,1985
13. Diomidis Spinellis. Code Quality: The Open Source Perspective. Addison Wesley, Boston, MA, 2006.
14. 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.
15. Stephen H. Kan. Metrics and Models in Software Quality Engineering. Addison-Wesley, Boston, MA, second edition, 2002.
16. Jeff Tian, Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, IEEE Computer Society Press, 2005, ISBN: 0471713457.
17. Musa, J.D, A. Iannino, and K. Okumoto, Engineering and Managing Software with Reliability Measures, McGraw-Hill, 1987
18. Myers, Glenford J. (1979). The Art of Software Testing. John Wiley and Sons. p. 145-146. ISBN 0-471-04328-1.
19. Barry W. Boehm, Software Engineering Economics, Prentice-Hall Inc., 1981.
20. Dustin, Elfriede (2002). Effective software Testing. Addison Wesley. p.3. ISBN 0-20179-429-2.

Article received: 2010-04-24