Implementing Software Quality Assurance (SQA) in UiTM Pahang: The Preliminary Study of Formal Methods in Software Development Life Cycle

Noraini Mohamed
Mahfudzah Othman
Nursyahidah Alias

ABSTRACT

This paper is aimed to study the method of Software Quality Assurance (SQA) that can be implemented in Universiti Teknologi MARA (UiTM) Pahang. Software quality as defined by Institute of Electronics and Electrical Engineers (IEEE) is the degree to which a system component or process meets specified requirements and user needs or expectations. Whereas, Software Quality Assurance (SQA) is a planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. In this study, we will investigate the use of formal methods that is integrated in software project life cycle development to ensure the quality assurance of the software. Focus is given to a number of methods only, which are the B Method, SDL, VDM, Z Language and Cleanroom Software Development. This paper will compare these techniques to study the use of formal methods in the Software Development Life Cycle (SDLC).

Keywords: *formal methods, Software Quality, Software Quality Assurance (SQA)*

Introduction

Software can be defined as computer programme, procedures and possibly associated documentation and data pertaining to the operation of a computer system (Galin, 2004). It also can be described as series of instructions or programme that tells a computer what to do and how to do it (Vermaat, 2008). As mentioned by Galin (2004), software is developed by many individuals and in different situations to fulfill a variety of needs. For instance, software development professionals such as system analysts and programmers develop software products as a professional career objective prior working in the software houses or in software development and maintenance units of large or small scale organizations.

Those people who participate in software development activities will be facing the software quality problems. Poor quality can result from inaccurate requirements, design problems, coding errors, faulty documentation and ineffective testing (Rosenblatt, 2001). However, quality problems in their most severe form govern the professional software development (Galin, 2004). Therefore, to ensure the quality of the software, it is important for the software development processes and methods used to be monitored using software quality assurance (SQA). The main objective of quality assurance is to avoid problems, or to detect them as soon as possible.

Motivation for Software Quality Assurance

Universiti Teknology MARA (UiTM) Pahang was accredited with ISO9001:2000 in 2005. This is a result of Quality Movement that has been practiced here since the establishment of Pahang campus. All aspects of teaching and learning, organization of administration and daily operations are covered by this movement. As a result, a number of software was produced in order to assist the Quality work that has been practiced. The development of the software is also to further sustain the accreditation obtained so that the quality may be retained if not improved. But one has to bear in mind that the Software itself is subjected to a number of quality issues. This is because only quality software will perform quality task designed for it to perform.

A set of criteria of a quality software among them are reliable, robust, correct and timeliness. In addition, the ISO 9126-1 software quality model identifies six main quality characteristics, namely:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Though the criteria mentioned seems to be very easily achieved, in reality it is not so, especially for an academic institution where the software is developed by the lecturer whose primary task is teaching and coaching. Another reason is that the quality of the software is important to the customer and in UiTM Pahang the customer happened to be the developer of the software itself. Therefore, there is a need to dissolve the quality issues regarding software since the software is part of a tool used to achieve the overall quality. The place that the quality of the software should start is at the process that produced the software as suggested by Huges & Cotterel (2006).

Software Quality Assurance (SQA) in Practice

The Need for SQA

Software production, developed or engineered via a software project need a careful planning since a lot of resources, which may be classified as very scarce are involved, such as man power, skill, time and cost. This careful planning and managing of software process is defined as SQA which encompasses the entire software development process. The processes are designing, coding, source code controlling, code reviewing, changing management, configuration management and releasing management (Wikipedia, 2008). In other word, it can be described as a control of processes that produced the software. SQA at a fundamental level, specifies all those process-related activities that are undertaken in the pursuit of achieving software quality. It involves careful attention to the specification of the requirements to be satisfied, to ensure they accurately capture what is wanted or intended (validation), and the formulation of the test cases that can be used to demonstrate their eventual satisfaction in code (verification).

SQA plan carried out thoroughly by quality assurance professionals will grant certain benefits such as improved customer satisfaction and reduced cost of development and maintenance. For instance, internal failures are the costs related to software defects that are found before shipping products to customers. Internal failures activities include correcting flaws in requirements and design; reviewing requirements and design changes; correcting flaws in purchased software; retesting purchased software corrections; and most importantly, reporting, tracking and fixing defects in the software being developed. Most organizations fail to recognize that each additional iteration of integration and system testing acts to the cost of internal failures (Cognence Inc., 2005).

Fletcher Buckley had developed IEEE standard 730 for software quality assurance, which was completed in 1979. The purpose of IEEE Standard 730 was to provide uniform, minimum acceptable requirements for preparation and content of software quality assurance plans (Buckley, 1984). This standard was influential in completing the developing standards in the following topics: configuration management, software testing, software requirements, software design, and software verification and validation.

The objective of SQA is to assure sufficient planning, reporting, and controlling to affect the development of software products which meet their contractual requirements (Fischer, 1978). To implement this objective, eight quality assurance (QA) functions can be identified, which are initial quality planning, development of software standards and procedures, development of quality assurance tools, conduct of audits and reviews, inspection and surveillance of formal tests, configuration verifications, management of the discrepancy reporting system and retention of QA records.

Methods in SQA

SQA activities as defined by Boger & Lyons (1984), usually design around a number of variables such as:

a) Size of the software
b) Degree of criticality of the software
c) Platform used
d) Level of user
e) Type of release
f) Relationship with the user
g) The cost of implementing a fix after release versus the cost of doing it right the first time.

Many methods come to existence to assure the quality of software as mentioned above are all geared towards disciplined approach to meeting requirements related to the above variables. However, focus will only be given on a formal method. The reason being is that growing number of interest of research has developed rapidly lately. Table 1 illustrates the thirty-five top ranking university which is currently actively involved in formal method research (Simas, 2001).

Table 1: Top Ranking University Actively Involved in Formal Method Research

Name	Faculty	Ranking
Stanford University	4	2
Carnegie Melon University	4	3
University of Texas	4	7
University of Pensyllvania	3	24
University of California	3	34
Massachusettes Institute of Technology	2	1
University of North Carolina	2	30
State University of New York	2	31

Formal Methods

In computer science and software engineering, formal methods are particular kind of mathematically-based techniques for the specification, development and verification of software and hardware systems (Butler, 2006). The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analyses can contribute to the reliability and robustness of a design (Holloway, 1997).

Since formal methods use mathematical notations, the use of these methods might strenghten the activities encompasses in SQA. Therefore, it may offer more accurate measures of quality of the software. There are many aspects of SQA that may be benefited from formal methods, such as in test specification, project estimation metrics, operational profiles and reliability engineering, and formal specification styles as a basis for natural language specification templates.

Formal methods can be used in all phases of a system's development and present an opportunity to develop new techniques to improve software production. Formal methods used in developing computer systems provide frameworks for specifying developing, and verifying systems in a systematic manner rather than ad hoc manner. System designers use formal methods to specify desired behavioral and structural properties of a system (Iglewski & Muldner, 1997).

Iglewski & Muldner (1997) added that one tangible product of applying a formal method is a formal specification. The specification serves as a contract, a valuable piece of documentation, and a means of communication between a client, a specifier, and an implementor. Formal specifications have the additional advantage over informal specifications because they are amenable to machine analysis and manipulation. The greatest benefit of applying a formal method is that system designers gain a deeper understanding of the specified system, because they have been forced to be more abstract and precise about the desired properties. Another important application of formal specifications is that they can be used as a base to reason about the behavior of system's components. For example, if predicate logic is used, then laws of this logic can be used to reason. But it is not necessary to prove very properly or detail of a system; indeed proving small things about core properties of the system may be invaluable in the future.

Formal Methods in System Development Life Cycle (SDLC)

For each system development phase, Iglewski & Muldner (1997) claimed that there are some applications of formal specifications and the formal methods that support them might be considered. Below are the applications of formal methods that can be applied in system development life cycle:

- Requirements analysis: this step clarifies the informally stated requirements, helps clear up vague ideas, reveals contradictions (or inconsistencies), ambiguities and incompleteness;
- System design: this step is used during modular decomposition and refinement to record design decisions and assumptions. A module interface specification provides its clients with the information needed to use the module without knowledge of its implementation. At the same time, it provides the implementor the information needed to construct the modules without knowledge of its clients. The implementation can be replaced by a more efficient one, without affecting the interface and the client's code;
- System verification: this is the process of showing that a system satisfies its specification. This process is impossible without a formal specification. It is important to realize that although the entire system may never be completely verified, a smaller, critical piece often can be;
- System validation: formal methods can aid in system testing and debugging. Specifications can be used to generate complete test cases;
- System documentation: a specification serves as a description of the system. It is used for a communication between a client and a specifier, between a specifier and an implementor, and among the implementation team;
- System analysis and evaluation: to learn from the experience of building a prototype system, developers should perform a critical analysis of its functionality and performance after this prototype has been built. Recently, significant research has been carried out in specifying a system which is already built, running, and used. Some of these exercises revealed serious bugs in published algorithms and designs.

Comparisons of Formal Methods

Table 2 shows the comparisons between some of the formal method techniques. As compared to the other formal methods, Z language is chosen since it is the formal method that can be used at the early stage in SDLC, which is in Software Requirement Specifications. Much software developed failed to meet its reliability criteria because of the ambiguous specification of the problem domain. The use of Z language in particular will help in removing the ambiguity of the problem domain. This is because of the precise nature of the mathematical notation in used. Furthermore the proving of the correctness of the software is made in the early stage in SDLC to ensure the reliability of the software. This way a more reliable and cheaper cost is involved in removing the errors at the end of SDLC.

Conclusion and Further Works

As a suggestion to produce a more quality software in UiTM Pahang, the Z language technique may be applied. For the existing software we may use a reverse-engineering technique to improve the software. Whereas, for the upcoming software, the use of Z language method is to be used from the beginning. Another area that we think worth giving an attention is the use of Z language technique on a Web-Based Application since a number of software currently in use at UiTM Pahang is operating on a Web.

Table 2: The Comparisons of Formal Methods Techniques

Techniques	Descriptions
Z language	Z is a formal specification language that was developed by Oxford University in 1977 (Wikipedia, 2008). It was later being standardized in year 2002 (ISO/IEC 2002). Z is based on the standard mathematical notation used in axiomatic set theory, lambda calculus, and first-order predicate logic. Z notation is used for describing and modeling computing systems. It is targeted at the clear specification of computer programmes and the formulation of proofs about the intended programmes behavior.
B Method	B method is a tool-supported formal method based around AMN (Abstract Machine Notation). B method has been used in major safety-critical system applications in Europe such as the Paris Metro Line 14, and is attracting increasing interest in industry (Wikipedia, 2008). It has robust, commercially available tool support for specification, design and proof and code generation. Compared to Z, B is slightly more low-level and more focused on refinement to code rather than just formal specification.
Vienna Development Methodology (VDM)	The Vienna Development Method (VDM) developed by IBM's Vienna Laboratory in the 1970s is one of the longest-established formal methods for the development of computer-based systems (Dines & Jones, 1978). Support for VDM includes commercial and academic tools for analyzing models, including support for testing and proving properties of models and generating program code from validated VDM models (Wikipedia, 2008).
Specification Description Language (SDL)	Specification and Description Language (SDL) is a specification language targeted at the unambiguous specification and description of the behavior of reactive and distributed systems. Originally focused on telecommunication systems, its current areas of application include process control and real-time applications in general (Sommerville, 2007)
Cleanroom Software Development	The Cleanroom Software Engineering process is a software development process intended to produce software with a certifiable level of reliability. The Cleanroom process was originally developed by Harlan Mills and several of his colleagues including Alan Hevner at IBM (Mills et al., 1987). The focus of the Cleanroom process is on defect prevention, rather than defect removal. Cleanroom development makes use of the Box Structure Method to specify and design a software product. Verification that the design correctly implements the specification is performed through team review. Recent work on the Cleanroom process has examined fusing Cleanroom with the automated verification capabilities provided by specifications expressed in Communicating Sequential Processes (CSP) (Broadfoot & Hopcroft, 2005)

References

Boger, D.C., & Lyons N.R. (1984). The organization of the software quality assurance. *ACM Journal, 16*(**2**).

Buckley, F.J. (1984). The IEEE software engineering standards process. *ACM'84 Annual Conference: The 5th Generation Challenge.* Association of Computer Machinery.

Butler, R.W. (2006). *What is Formal Methods?* Retrieved November 20, 2008, from Langley Formal Methods. http://shemesh.larc.nasa.gov./fm/fm-what.html.

Cognence Inc. (2005). *Are software quality problems costing your company?* Denver: Cognence Inc.

Fischer, K.F. (1978). Software Quality Assurance tools: Recent experience and future requirements. *Proceedings of the software quality assurance workshop on functional & performance issues* (pp. 116-121). Association Computer Machinery.

Galin, D. (2004). *Software Quality Assurance: From theory to implementation.* Great Britain: Pearson Education Ltd.

Holloway C.M. (1997). Why engineers should consider Formal Methods? *16th Digital Avionics Systems Conference.* Wikimedia Foundation Inc.

Huges, B. & Cotterel, M. (2006*). Software project management (4th ed).* United Kingdom: Mc Graw Hill.

Iglewski, M. & Muldner, T. (1997). Comparison of formal specification methods and object-oriented paradigms. *Journal of Network Computer , 20* (4), 355-377.

methods graduate research. Retrieved November 20, from http://chart2d.sourceforge.net/jjsimas/prof/ documents/undergradsGuideToFM.pdf.

Rosenblatt, S.C. (2001). *System analysis and design (4th ed.)* Boston: Thomson Course Technology.

Simas,J.J. (2001). A synthesis of introductory information for undergraduates considering formal

Vermaat, S.C. (2008). *Discovering Computers: Fundamentals (4th ed.).* Massachusetts: Thomson Course Technology.

Wikipedia. (2008, November). *Software Quality Assurance.* Retrieved November 13, from Wikipedia: The free encyclopedia http://en.wikipedia.org/wiki/software_quality_assurance

---

NORAINI BT. MOHAMED, MAHFUDZAH BT. OTHMAN & NURSYAHIDAH BT. ALIAS. Faculty of Information Technology & Quantitative Sciences, Universiti Teknologi MARA Pahang. noraini@pahang.uitm.edu.my