Creation of Algorithms for Making Test Cases, and Implementing using JAVA which can be Applied to Each Phase of SDLC to Ensure Quality Assurance

Supriya Shrivastava

Shri Ram Institute of Technology, Rajiv Gandhi Technical University, Bhopal (M.P) India

Abstract: *Creation of algorithms for making test cases and implementing using high level language such as JAVA which can be applied to each phase of SDLC to ensure quality assurance". These are test-like programs that automatically check whether an implementation conforms to a specific rule. These rules are implemented directly in the target programming language in the form of tests.*

Keywords: Software quality Assurance (SQA). Capability Maturity Model Integration (CMMI), Software Development Life Cycle (SDLC), International Organization for Standardization (ISO), Rational Unified Process (RUP)

1. Introduction

The current software development practice is still accident-prone. For instance, projects are completed too late; they exceed their budgets; or they require substantially more resources than expected. The intrinsic reason is that we have inadequate understanding of our objective world and its characteristics. Those problems are addressed as the software crisis.

As software is integrated more frequently into every aspect of our lives, as it grows more quickly in size and function, as its failure in operations causes increasingly devastating consequences, and as schedules and budgets are continually reduced despite the need for high-quality, reliable, and secure software, advanced and innovative technologies must be developed to achieve software quality assurance more effectively and efficiently. It is also critical for the industry and academia to work together to conduct cooperative research to reduce the gap between state-of-the-art analyses and practice applications.

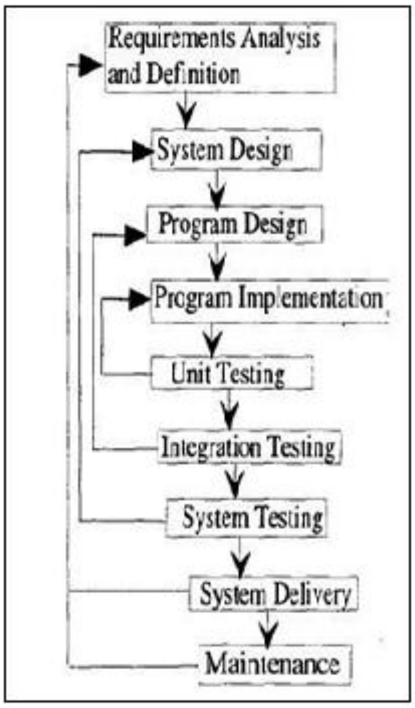
Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI.

Software quality assurance (SQA) contains, different ways of having a continuous check on software engineering processes, and methods used to certify quality.

Software Quality Assurance of large size software involves checking and assuring if many aspects like the design, coding, testing and performance of the software are as per the specifications. There are many attempts done all over the world to quantify the quality of the software and many quality metrics have been evolved. The quality attributes are measured in various ways. Two of these ways in which performance can be gauged are -

- By measuring the time required for execution of the functions and
- By the number of problems found during testing.

There are many QA methodologies like the standard Waterfall SDL based on the standards laid down by the IEEE, CMMI (Capability Maturity Model Integration) based on the guideline of the SEI, V-model, Incremental model, RUP (Rational Unified Process), Agile Method of Software Development, Test-based development method, Rapid Action Development model etc.



Software Development Life Cycle

2. Quality Assurance Techniques

Basic quality assurance techniques are:

- Application of Standards

- Definition and delivery of end products
- Traceability

The competitive market place today demands the best of everything - Quality, Cost and Schedule. The on time delivery of an error-free product at minimal cost is standards that demanding customers expect and good suppliers continually strive to meet. It is no easy task to strike an effective balance where quality is accomplished without sacrificing schedules and incurring unplanned costs - and to do so consistently, release after release.

In a software development project, errors can be injected at any stage during development. For each phase, there are different techniques for detecting and eliminating errors that originate in that phase. However, no technique is perfect, and it is expected that some of the errors of the earlier phases will finally manifest themselves in the code. This is particularly proved because in the earlier phases most of the verification techniques are manual because no executable code exists. Ultimately these remaining errors will be reflected in the code. Hence the code developed during the coding activity, it is likely to have some requirements errors and design errors, in addition to the errors introduced during the coding activity. Because the code is frequently the only product that can be executed and whose actual behavior can be observed, testing is the phase where the errors remaining from all the previous phases must be detected. Of the development cost, an example distribution of effort with the different phases is shown in following table:

Table 1: Effort Distribution

Requirements	*10%*
Design	20%
Coding	50%
Testing	50%

The exact numbers will differ with organization and the nature of the process.

However, there are some observations we can make. First is that coding consumes only a small percentage of the development effort. This is against the common naïve notion that developing software is largely concerned with writing programs and that programming is the major activity.

The second important observation from the data about effort distribution with phases is that testing consumes the most resources during development.

Overall, we can say that the goal of the process should not be to reduce the effort of design and coding, but to reduce the cost of testing and maintenance. Both testing and maintenance depend heavily on the design and coding of software, and these costs can be considerably reduced if the software is designed and coded to make testing and maintenance easier.

Question arises, what are the sources of the defects?

The source of the defects can be many: oversight wrong assumptions use of inappropriate technology, communication gap among the project engineers, etc. These defects usually get detected much later in the life cycle. For example, a design defect might go unnoticed till we reach the coding or testing phase.

Once a defect is detected, the engineers need to go back to the phase where the defect had occurred and redo some of the work done during that phase and the subsequent phases to correct the defect and its effect on the later phases. It ultimately increases the cost, because for example, the defect was occurred in the coding, so the development team needs to work again on this. It will also unnecessarily increase the delay in schedule.

3. Early Defect Removal and Defect Prevention

Table 2: Effort Distribution

Requirements	*20%*
Design	30%
Coding	50%

As we can see, errors occur throughout the development process. However the cost of correcting errors of different phases is not the same and depends on when the error is detected and corrected.

The main moral of this section is that we should attempt to detect errors that occur in a phase during that phase itself and should not wait until testing to detect errors. Detecting errors soon after they have been introduced is clearly an objective that should be supported by the process. However, even better is to provide support for defect prevention. It is generally agreed that all the defect removal methods that exist today are limited in their capability and cannot detect all the defects that are introduced. Furthermore, the cost of defect removal is generally high, particularly if they are not detected for a long time. Clearly, then, to reduce the total number of residual defects that exist in a system at the time of delivery and to reduce the cost of defect removal, an approach is to prevent defects from getting introduced and stop moving those defects to carry forward to next phase of SDLC.

References

[1] Jo˜ao Brunet, Dalton Guerrero, Jorge Figueiredo, "Design Tests: An Approach to Programmatically Check your Code Against Design Rules" published in IEEE ICSE'09, May Vancouver, Canada 978-1-4244-3494-7
[2] Yuri Kharmov, "The Cost of Code Quality" published in IEEE 2006 0-7695-2562-8/06
[3] Naeem Seliya Taghi M. Khoshgoftaar and Shi Zhong, "Analyzing Software Quality with Limited Fault-Proneness Defect Data" published in IEEE 2005 1530-2059/05
[4] Luigi Benedicenti, Victor Wei Wang, Raman Paranjape, "A Quality Assessment Model for Java Code" published in IEEE 2002 0-7803 -75 14-9/02

[5] Xuefen Fang, "Using A Coding Standard to Improve Program Quality" SRA Key Technology Laboratory, Inc, Published in IEEE 2001
[6] Charles H. Wells Russell Brand and Lawrence Markosian, "Customized Tools for Software Quality Assurance and Reengineering" Published in IEEE 1995 0-8186-7111-4/95
[7] Per Runeson and Peter Isacsson, "Software Quality Assurance-Concepts and Misconceptions" published in IEEE 1998 1089-6503/98
[8] S.Murugesa, "Attitude Towards Testing: A Key Contributor to Software Quality" 0-7803-2608-3
[9] Raymond Day and Thomas McVey, "A Survey of Software Quality Assurance" published in IEEE AES Magazine, November 1986
[10] FLETCHER J. BUCKLEY and ROBERT POSTON, "Software Quality Assurance" published in IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. SE-10, NO. 1, JANUARY 1984
[11] Rathna K. Prasad, "Towards A Zero-Defect Product - The End-To-End Test Process" AT&T Bell Laboratories 0-7803-2608-3
[12] Pankaj Jalote, "An Integrated Approach to Software Engineering" 81-7319-271-5
[13] Sterling J. McCullough, "SOFTWARE QUALITY ASSURANCE METHODOLOGY EVALUATION AND SELECTION" published in IEEE CH2984-319110000-0364 1991

Author Profile

Supriya Shrivastava is an Assistant Professor in IT branch in Shri Ram College Jabalpur (M.P. India) since 2010. Had completed MCA in 2006 from Jabalpur Engineering College, Jabalpur.