International Journal of Recent Trends in Engineering & Research
ISSN(ONLINE) : 2455-1457
IMPACT FACTOR : 3.344

SOFTWARE TESTING WITH DIFFERENT PHASES OF SDLC

Preeti Yadav[1], Ajay Kumar[2]
[1]*Asst. Prof in Govt college Gurgoan(Haryana)*
[2]*Asst. Prof. in IGU Meerpur Rewari(Haryana)*

Abstract— Software Testing is the process of executing a program or system with the intent of finding errors. Or, it involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Software is not unlike other physical processes where inputs are received and outputs are produced. Where software differs is in the manner in which it fails. Most physical systems fail in a fixed (and reasonably small) set of ways. By contrast, software can fail in many bizarre ways. Detecting all of the different failure modes for software is generally infeasible. Software testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results. Although crucial to software quality and widely deployed by programmers and testers, software testing still remains an art, due to limited understanding of the principles of software. The difficulty in software testing stems from the complexity of software: we cannot completely test a program with moderate complexity. Testing is more than just debugging. The purpose of testing can be quality assurance, verification and validation or reliability estimation. Testing can be used as a generic metric as well. Correctness testing and reliability testing are two major areas of testing. Software testing is a trade-off between budget, time and quality.
Keywords— Binary analysis, Software testing, Quality assurance, Debugging, Vera code.

I. INTRODUCTION

The process of manufacturing software systems. A software system consists of executable computer code and the supporting code and the supported documents needed to manufactured, use, and maintain the code. For example a word processing system consists of an executable program (the word processor), user manuals and the documents, such as requirements and designs, needed to produce the executable program and manuals. Software engineering is ever more important as larger, more complex and life critical software systems proliferate. The rapid decline in costs of the computer hardware means that the software in a typical system often costs more than the hardware it runs on. Large software system may be the most complex things ever built. This places great demands on the software engineering process, which must be disciplined and controlled.

II. CRISIS

Software crisis was a term used in the early days of computing science for the difficulty of writing useful and efficient computer programs in the required time. The software crisis was due to the rapid increases in computer power and the complexity of the problems that could be tackled. With the increase in the complexity of the software, many software problems arose because existing methods were neither sufficient nor up to the mark. The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem. The main cause is that improvements in computing power had outpaced the ability of programmers to effectively utilize those capabilities. Various methodologies have been developed over the last few

decades to improve software quality management such as procedural programming and object-oriented programming. However software projects that are large, complicated, poorly specified, and involve unfamiliar aspects, are still vulnerable to large, unanticipated problems.
The causes of the software crisis were linked to the overall complexity of hardware and the software development process. The crisis manifested itself in several ways:

- Projects running over-budget.
- Projects running over-time.
- Software was very inefficient.
- Software was of low quality.
- Software often did not meet requirements.
- Projects were unmanageable and code difficult to maintain.
- Software was never delivered.

III. DIFFERENCE BETWEEN TRADITIONAL SOFTWARE ENGINEERING AND SOFTWARE ENGINEERING

1. Software engineering is based on computer science, information science and discrete mathematics whereas traditional engineering is based on mathematics, science and empirical knowledge.
2. Traditional engineers construct real artifacts and software engineers construct non-real (abstract) artifacts'.
3. In traditional engineering, two main concerns for a product are cost of production and reliability measured by time to failure whereas in software engineering two main concerns are cost of development and reliability measured by the no. of errors per thousand lines of source code.
4. Both disciplines requires maintenance but in different ways.
5. Software engineers often apply new and untested elements in software projects while traditional software engineers generally try to apply known and tested principles and limit the use of untested innovations to only those necessary to create a product that meets its requirements.
6. Software engineers emphasize projects that will live for years or decades whereas some traditional engineers solve long-ranged problems that ensure for centuries.
7. Software engineering is about 50 years old whereas traditional engineering as a whole is thousands of years old.
8. Software engineering is often busy with researching the unknown (e.g. to drive an algorithm) right in the middle of a project whereas traditional engineering normally separates these activities. A project is supposed to apply research results in known or new clever ways to build the desired result.
9. Software engineering has first recently started to codify and teach best practice in the form of design pattern whereas in traditional, some engineering discipline have thousands of years of best practice experience handed over from generation to generation via a field's literature , standards, rules and regulations.

IV. SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.
SDLC Activities: SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC framework includes the following steps:

1. **Communication:** This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

2. **Requirement Gathering:** This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given -
   a. studying the existing or obsolete system and software,
   b. conducting interviews of users and developers,
   c. referring to the database or
   d. Collecting answers from the questionnaires.

3. **Feasibility Study:** After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

4. **System Analysis:** At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems

beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

5. **Software Design:** Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

6. **Coding:** This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

7. **Testing:** An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

8. **Integration:** Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

9. **Implementation:** This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

10. **Operation and Maintenance:** This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

11. **Disposition:** As time elapses, the software may decline on the performance front. It may go completely obsolete or may need intense up gradation. Hence a pressing need to eliminate a major portion of the system arises. This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

V. TESTING TECHNIQUES OF SOFTWARE ENGINEERING

1. **Black box testing:** Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.

2. **White box testing:** This testing is based on knowledge of the internal logic of an application's code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

3. **Unit testing:** Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code may require developing test driver modules or test harnesses.

4. **Incremental integration testing:** bottom up approach for testing i.e continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately done by programmers or by testers.

5. **Integration testing:** Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

6. **Functional testing:** This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

7. **Security testing:** Can system be penetrated by any hacking way. Testing how well the system protects against unauthorized internal or external access. Checked if system, database is safe from external attacks.

8. **Compatibility testing:** Testing how well software performs in a particular hardware/software/ operating system/network environment and different combination s of above.

9. **Comparison testing:** Comparison of product strengths and weaknesses with previous versions or other similar products.

10. **Alpha testing:** In house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made as a result of such testing.

11. **Beta testing:** Testing typically done by end-users or others. Final testing before releasing application for commercial purpose.

VI. CONCLUSION

Software Testing covers a wide range of areas where any verification or validation of software functionality can occur. Occasionally, non-functional aspects become less concerning over the functional aspects. They are not performed practically; simultaneously during software testing. SDLC is an acronym for **Software Development Life Cycle**. It is also sometimes referred to as **System Development Life Cycle**. In simple words it the process, methods or a set of methodologies applied to create or alter software projects. Each of these methodologies defines unique way to create a new software module or program.

VII. REFERENCE

1. KK Aggarwal, “Software Engineering”.
2. Russell Kay, “QuickStudy: System Development Life Cycle”, May 14, 2002.
3. Blanchard, B. S., & Fabrycky, W. J.(2006) Systems engineering and analysis (4th ed.).
4. Marakas, James A, O’Brien, George M. (2010). Management information systems New York: McGraw-Hill/Irwin. pp.485– 489.
5. Paul Ammann, Jeff Offutt, “ Introduction to Software Testing”(2008).
6. Van Veenendaal, Erik, “Standard glossary of terms used in Software Testing” 4 January 2013.
7. Clapp, Judith A, “Software Quality Control, Error Analysis, and Testing”(1995).
8. Jiantao Pan, “Software Testing” Carnegie Mellon University

9. Authors' Profiles:

PREETI YADAV: Preeti Yadav did her B.Tech from Dronacharya Collge of Engineering,Gurgaon Haryana (India) and M.Tech (Computer Science) from BRCM,Bahal,, Haryana (India). She has teaching experience about 4 years in India. She has UGC NET –JRF qualified. At present she has been working as Assistant Professor (Extension Faculty) in Department of Computer Science & Application, Govt.Girls College,sec-14,Gurgaon(India). She has about 8 international journal publications.

AJAY KUMAR: Ajay Kumar did his MCA from LIMAT Faridabad, Haryana (India) and M.Tech (Computer Science) from GITAM Jhajjar, Haryana (India). He has teaching experience about 4 years in India. He has UGC NET qualified. At present he has been working as Assistant Professor (Guest Faculty) in Department of Computer Science & Application, Indira Gandhi University, Meerpur, Rewari (India). His research interest in Networks Security and database management System. He has about 7 international journal publications.

