Faces of Testing Strategies: Why &When?

Sanjana[1], Yashveer Singh[2], Sagar Choudhary[3]

[1,3]Assistant professor, Dept of Computer Science and Engineering, Roorkee College of Engineering, Roorkee

---***---

***Abstract**: Software Testing is an activity that focuses accessing the capability of a program and dictates that it truly meets the quality results. Software Testing has been considered as the most significant stage of software development. Software Testing is the process of quality checking about the software system, conduct the software testing engineer and stakeholders with testing about the quality of software product. About 60% of the resources and money are cast-off for the testing of software. Whenever we think of developing software, we always concentrate on making the software bug free and reliable. In that case, Testing plays an vital role and the quality of software is assured by testing the software development application and its progression [1].This paper includes the need of testing, when it is required and how it will be done during software development process.*

1. INTRODUCTION

Testing is an essential activity in software engineering. In the simplest terms, it amounts to observing the execution of a software system to validate whether it behaves as intended and identify potential malfunctions. Testing is widely used in industry for quality assurance, indeed, by directly scrutinizing the software in execution, it provides a realistic feedback of its behavior and as such it remains the inescapable complement to other analysis techniques.

Software testing is a technique aimed at evaluating an attribute or capability of a program or product and determining that it meets its quality [2]. Software testing is also used to test the software for other software quality factors like reliability, usability, integrity, security, capability, efficiency, portability, maintainability, compatibility etc. According to Dale Emery and Elisabeth Hendrickson, Testing is defined as "It is a process of gathering information by making observations and comparing them to the expectations"[3].

Why Testing?

For any company developing software, at some point pressure to reach the deadline in order to release the product on time will come into play. Additional pressure from project stakeholders, such as 'Marketing' will not want to delay the release date as significant effort and money may have already been spent on an expected release date.

Fig 1: Testing process to find bug

Quite often, planned time to test the software will become reduced so as not to impact the release date. From a pure business perspective, this can be seen as a positive step as the product is reaching the intended customers on time. Careful consideration should be taken though as to the overall impact of a customer finding a 'bug' in the released product. Maybe the bug is buried deep within a very obscure functional area of the software product, and as the impact only results in a typo within a seldom-used report, the level of impact is very low.

In this case, the effect on the business for this software company would probably be insignificant. But what if the bug resulted in the program crashing and losing data? Maybe this software product is used within an air traffic control system? As you can imagine, the impact of this type of bug could be incredibly high and may result in loss of life and destroying the entire company responsible. So basically, the level of risk of a bug being found and what is the effect of the bug prove to be critical in how much software testing is performed prior to a products release.

Software testing and or Quality Assurance is still a kind of art, mainly due to a limited understanding of the complexities of modern software. Recent years has seen the development of software testing certification such as ISEB and ISTQB. This is good news for the software industry as a whole, as the more experienced a software tester is then the level of quality of the software they are testing can onlyincrease[4].

2. Defining Done

2.1 Finding Defects

Testing is very unlikely to find certain types of defects. Even the combination of several different styles of testing is unlikely to find more than 75% of the total defects. Each individual type of testing, even when carefully executed with a high degree of skill, is unlikely to find more than 50% of the defects. Testing is relatively inefficient at finding defects, even assuming we could find all of them. We will potentially spend a very, very long time trying to find all the defects, especially the last few. We have no way of knowing in advance if all the defects have been found, or whether more remain in the system.

So in practice we need to abandon the idea of finding all the defects and use a different approach. Instead, evaluate the likelihood of finding more defects based on the effort you have already put in and based on the results we have obtained[5].If this likelihood is too low then we are done testing, additionally testing would not provide a sufficient return on investment in terms of new defects found compared to the effort expended. Some points to consider when making this evaluation:

- If defect is just found, this is a signal to keep testing. It may seem counter-intuitive, but in general the more defects we find, the more likely it is that there are additional defects.
- If we have only exercised a small portion of the overall functionality and already found defects, then this is a signal to continue testing.
- If we have been testing a particular piece of functionality for a while and are not finding new defects, then this is a signal to stop testing.
- If we are struggling to come up with new tests that are meaningful, then this is a signal that testing is done. Another sign of this is when the defects that are found to have low relevance to users and the decision is made not to fix them.
- If the tests we are performing are becoming more and more complicated and taking significantly more effort, but if we are only occasionally finding defects, then this is a signal to stop.
- When testing a larger set of functionality like an entire application, the question of whether to stop testing can and should be applied at the level of individual features or components. One reason for this is that defects tend to cluster. This commonly leads to a system having a few error-prone components that account for up to 80% of the total number of defects, while other components can be significantly higher quality. Once you have identified a component that appears error-prone, focusing additional testing on it is quite likely to find more defects.

2.2 Assessing readiness

If your goal in testing is to assess whether the software is ready to be promoted to the next level of testing or released into production, then you are done testing once you have obtained a sufficiently high level of confidence in your assessment. If your assessment is a 'no-go' - do not proceed with the promotion / release - then it is likely that issues you found will need to fixed and trigger additional testing in order to obtain your 'go' recommendation.

Factors to consider in performing this assessment are:

- What level of quality is required? Is the system life-critical, mission-critical, or only for casual personal use? This quality level relates directly to the level of confidence you need to have that the system is ready.
- It is far easier to determine that the system is not ready. Finding a critical defect or finding even a few serious defects in critical functionality is usually sufficient to warrant a no-go assessment. In contrast, achieving sufficient confidence that the system is good for release usually takes more work.
- How much of the system's functionality have you tested? If there are significant features that are mostly or entirely not tested, then you likely will not be prepared to recommend it is good to go. So frequently you should favor testing critical functionality broadly across the entire application versus focusing in detail on a single component.

- This conflicts to some degree with the approach suggested for the prior goal of finding defects where you might choose to focus your testing on an error-prone module.
- False confidence is a very real danger. This is especially true for developers - I have seen or heard of many who have an unrealistically high level of confidence that their code works, in some cases without any testing whatsoever. For Example, I have heard developers state that if their code compiles, it is good enough to be promoted to acceptance testing.This is one reason to have on development teams one or more testers with a testing mindset who will ensure for themselves that the system will work rather than assume it.
- Testing should be designed as much as possible to find defects of the highest severity and highest relevance to users, as these are the kinds of defects most likely to warrant a no-go assessment. You might be able to find lots of cosmetic defects like spelling mistakes in the application's help documentation, but this does not really contribute much towards an assessment of readiness.

3. Faces of Testing?

Testing embraces a variety of activities, techniques and actors, and poses many complex challenges. Indeed, with the complexity, pervasiveness and criticality of software growing ceaselessly, ensuring that it behaves according to the desired levels of quality and dependability becomes more crucial, and increasingly difficult and expensive. It is estimated that testing can consume fifty percent, or even more, of the development costs [6], and a recent detailed survey in the United States [7] quantifies the high economic impacts of an inadequate software testing infrastructure.

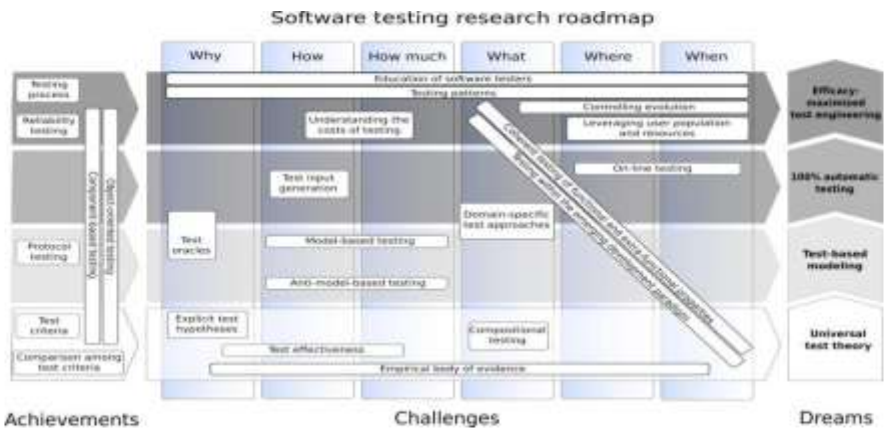

Fig 2: Software Testing Research: Challenges and Achievements

Correspondingly, novel research challenges arise, such as for instance how to conciliate model-based derivation of test cases with modern dynamically evolving systems, or how to effectively select and use runtime data collected from real usage after deployment. These newly emerging challenges go to augment longstanding open problems, such as how to qualify and evaluate the effectiveness of testing criteria, or how to minimize the amount of retesting after the software is modified.

Starting from this very general view, we can then concretize different instances, by distinguishing the specific aspects that can characterize the sample of observations:

3.1 Challenges in Software Testing

WHY: why is it that we make the observations? This question concerns the *test objective*, e.g.: are we looking for faults? or, do we need to decide whether the product can be released? or rather do we need to evaluate the usability of the User Interface?

HOW: which sample do we observe, and how do we choose it? This is the problem of *test selection*, which can be done ad hoc, at random, or in systematic way by applying some algorithmic or statistical technique. It has inspired much research, which is understandable not only because it is intellectually attractive, but also because how the test cases are selected -the test criterion- greatly influences test efficacy.

HOW MUCH: how big of a sample? Dual to the ques- tion of how do we pick the sample observations (test se- lection), is that of how many of them do we take (*test ad- equacy*, or stopping rule). Coverage analysis or reliability measures constitute two "classical" approaches to answer such question.

WHAT: what is it that we execute? Given the system under test, we can observe its execution either taking it as a whole, or focusing only on a part of it, which can be more or less big , more or less defined, this aspect gives rise to the various *levels of testing*, and to the necessary scaffolding to permit test execution of a part of a larger system.

WHERE: where do we perform the observation? Strictly related to what do we execute, is the question whether this is done in house, in a simulated environment or in the target final context. This question assumes the highest relevance when it comes to the testing of embedded systems.

WHEN: when is it in the product lifecycle that we perform the observations? The conventional argument is that the earliest, the most convenient, since the cost of fault removal increases as the lifecycle proceeds. But, some observations, in particular those that depend on the surrounding context cannot always be anticipated in the laboratory, and we cannot carry on any meaningful observation until the system is deployed and in operation.

3.2 Achievements

Testing process: Indeed, much research in the early years has matured into techniques and tools which help make such "test-design thinking" more systematic and in- corporate it within the development process. Several test process models have been proposed for industrial adoption, among which probably the "V model" is the most popular. All of its many variants share the distinction of at least the Unit, Integration and System levels for testing.

More recently, the V model implication of a phased and formally documented test process has been argued by some as being inefficient and unnecessarily bureaucratic, and in contrast more *agile* processes have been advocated. Concerning testing in particular, a different model gaining attention is *test-driven development* [8], one of the core *extreme programming* practices. The establishment of a suitable process for testing was listed in FOSE2000 among the fundamental research topics and indeed this remains an active research today.

Test criteria. Extremely rich is the set of test criteria devised by past research to help the systematic identification of test cases. Traditionally these have been distinguished between white-box and black-box, depending on whether or not the source code is exploited in driving the testing. A more refined classification can be laid according to the source from which the test cases are derived [8], and many textbooks and survey articles exist that provide comprehensive descriptions of existing criteria. Indeed, so many criteria among which to choose now exist, that the real challenge becomes the capability to make a justified choice, or rather to under- stand how they can be most efficiently combined. In recent years the greatest attention has been turned to model-based testing.

Comparison among test criteria. In parallel with the investigation of criteria for test selection and for test adequacy, lot of research has addressed the evaluation of the relative effectiveness of the various test criteria, and especially of the factors which make one technique better than another at fault finding. Past studies have included several analytical comparisons between different techniques [9]). This have permitted to establish a hierarchy of relative thoroughness between comparable criteria, and to understand the factors influencing the probability of finding faults, focusing more in particular on comparing partition (i.e., systematic) against random testing. "Demonstrating effectiveness of testing techniques" was in fact identified as a fundamental research challenge in FOSE2000, and still today this objective calls for further research, whereby the emphasis is now on empirical assessment.

4. Conclusion

Software testing is and will continue to be a fundamental activity of software engineering: notwithstanding the revolutionary advances in the way it is built and employed, the software will always need to be eventually tried and monitored. A necessary concluding remark concerns the many fruitful relations between software testing and other research areas. By focusing on the specific problems of software testing, we have in fact overlooked many interesting opportunities arising at the border between testing and other disciplines.

Software testing, synonymous with Quality Assurance itself can have many different purposes like quality assurance, validation, performance etc. This is a key decision when planning the QA /software testing, as not testing enough or testing in the wrong areas will inevitably result in missed bugs.

5. References

[1] Syedroohullahjan, Syedtauhidullah shah, Yasin shah, Fazlullahkham "Department of Computer Science", An innovative to Investigate Various Software Testing Techniques and Strategies.

[2] http://selab.netlab.uky.edu/homepage/sw-test-roadmap-bertolino.pdf

[3] Himanshi Babbar "Software Testing Techniques and test cases" Chandigarh Group of Colleges, International Journals of research in computer applications and Robotics,ISSN 2320-7345

[4] https://www.testingexcellence.com/why-do-we-need-software-testing/

[5] http://www.basilv.com/psd/blog/2011/when-is-testing-done

[6] B. Beizer. *Software Testing Techniques (2nd ed.)*. Van Nos- trand Reinhold Co., New York, NY, USA, 1990.

[7] NIST, The economic impacts of inadequate infrastructure for software testing, May 2002.

[8] D. Janzen, H. Saiedian, and L. Simex. Test-driven develop- ment concepts, taxonomy, and future direction.*Computer*, 38(9):43–50, Sept. 2005.

[9]P. Frankl and E. Weyuker. Provable improvements on branch testing. *IEEE Trans. Softw. Eng.*, 19(10):962–975, 1993.