Future of Software Testing: Novel Perspective, Challenges

Manas Kumar Yogi[#1], G.Kumari[*2], Himatej S R Y[#3] , Ch V S G Manikanta[#4]

[1,2]*Asst. Prof. CSE Department ,Pragati Engineering College(Autonomous)*
[3,4]*B.Tech. III Year Students CSE Department ,Pragati Engineering College(Autonomous)*
Surampalem, Dist-East Godavari ,A.P., India

Abstract *Software testing is part of the software development which ensures software functions in the intended way of the client. Software testing depends on how well we are practicing the principles of software testing currently. Formation of testing in Modern era is becoming popular day by day. Therefore, software testing engineers are trying to efficiently convert the manual test effort into automation test report. This challenge is difficult due to various factors which we are presenting in this paper. This paper is a novel afford to get software testing practitioners regarding path they follow to overcome the challenges of software testing. Our paper is a sincere advice to envision the future of software testing which is highly dependent on current software testing practices.*

Keywords - *testing, risk, automation, exploratory*

I. Introduction

Future of software testing depends on perspective of people who involve themselves in a conception of a software till its complete manufacturing along with users who use the software. The future of software testing depends on current testing principles. Concisely, future testing depends on current testing. Present on average software developers have a cautious approach towards development process because of the fact that to obtain a high quality software. The testing should be robust. In the current scenario, it is found under popular observation that the reliance on manual testing is the top technical challenge in application development. So it amounts to nearly 80% of effort expended in testing. Test automation also needs Skillet developers amounting create 20% of testing effort. The 80% manual test effort again can be broken down to manual test case creation execution maintenance but this also includes manual test data localization preparation during execution just a small fracture goes into the automated testing between 15 to 20% automation is carried out on the level of UI to user interfaces. .

As we can observe in the diagram above test automation is regarded as developers discipline and open it becomes perfectly clear then when we use as a manual tester depends tell on manual testing

principles creating and maintaining a test automation solution becomes challenging so for survival of manual test engineer the Automation Testing rate should be more than 85 to 90% or even about 95% the next challenge is teacher testing should focus on API testing it refers to automation driven by uses of application programming interface is like service for example yet another challenge decrease the effort of manual testing dramatically also needs to transform into exploratory testing now that's the future of testing in a nutshell. In subsequent sections of this paper we will tell in deep about each of the challenges mentioned above.

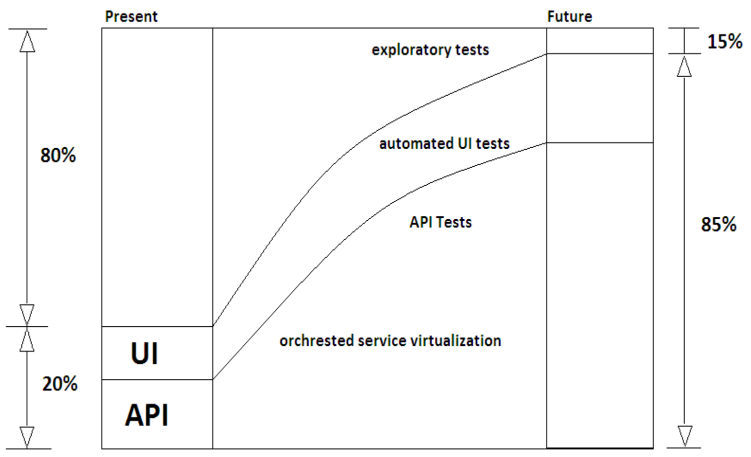

Fig. 1. Current scenario of software testing approach

II. API Testing:

In the graph which follows below we can observe the degree of completion of UI and API is usually evolved over a certain period of time let's see from the beginning of certain iteration to its ending. When it comes to UIs we can observe the curve which drops low unevenly in initial development duration but rises sharply during later stages. UI revisions takes time, and API testing results in greater efficiency than UI testing. To be precise according to recent research work in this area, it has been found that there is gain of 4 times in test creation for API testing over UI testing gain of 6 times in test case maintenance gain of 20 times in test case execution. The main reason is UI iterations are difficult to manage get in time to test whereas APIs much early available in development phase and so we can start testing early by using those APIs to shift test automation in left side in the curve, thereby increasing efficiency of API testing. The main challenge in the API testing is that most of the time APIs which are to be integrated into a system may be under maintenance. This creates a hell of test environment as we cannot repeat our test execution without interruption. So we need to have the ability to virtualize that means simulate the communication between your real system and surrounding systems. This happens in enterprise end to end testing. The greatest benefit is that we can identify a lot of critical defects atleast one step earlier in our development phase, the main reason being able to verify the system functionality from any integration perspective much earlier.

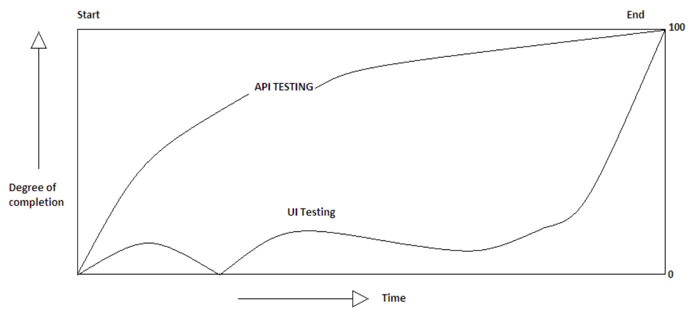

Fig. 2.UI Testing Versus API Testing

3.Risk Coverage Optimization: For a typical software development cycle, in some sprints we add more test cases and in some sprints we add less test cases nevertheless number of test cases goes on increasing, thereby introducing a scope of test redundancy in test suite. Over some time of testing we hit limit a critical limit. The critical limit indicate that we do not have the time the resources in the budget to execute and to maintain all the design test cases at some point in time. Then we have to make a decision, regarding which test cases have to be executed first in order as much as possible risk out of the system as early as possible. To be more radical even ask ourselves which test case needed to keep pace with the Agile development. Risk based approach allows us to categorize the test case portfolios into a high medium low risk area. For example to carry out light testing in low risk areas and heavy testing in high risk areas. The figure below depicts the idea.

The objective is to achieve maximum risk coverage which small number of test cases. In test driven software development the number of test cases grows exponentially over time, so efficient methodological test case design is a challenge to be faced in future. In reality risks in software development consists of functional risk, usability risk, reliability risk, performance risk, security risk, coherence risk etc. To meet all this risk is single minded approach may not be a strong option. There by the advocate a hybrid strategy of specification based testing along with features of exploratory testing.

4.Hybrid Test Strategy: With specification based testing we will be able to cover the most important risk but cannot cover all of them due to constraints on the test environment. This left us with the risk factor which we can accept before software is released. Subsequently exploratory testing along with risk based testing helps in creating new testing ideas whenever they are required. It actually diversifies testing which makes testing intellectually rich, productive. More critical bugs can be uncovered by such approaches. Also it provides fast error detection rates rapid feedback for comparison of test cases pass results versus fail result. Hybrid strategy enriches already existing test case portfolios.

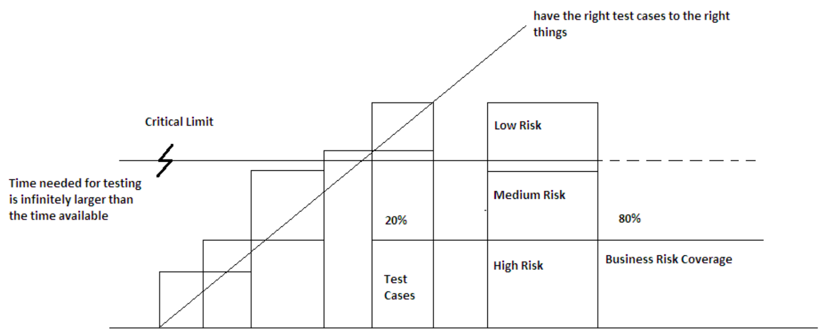
have the right test cases to the right things
Critical Limit
Time needed for testing is infinitely larger than the time available
Low Risk
Medium Risk
High Risk
20%
80%
Test Cases
Business Risk Coverage

Fig. 3. Risk Coverage versus Number Of Test Cases

CONCLUSIONS:

This paper brings out a clear, crisp vision of how classical testing which is carried out now will be outperformed we think broadly to envision automation testing to hundred percent extent. Our work discusses the prevalent challenges encountered during automation testing and how it effects software delivery pipeline. We throw light on various factors to be considered minimizing the effects of risk faced by the software functionality. We conclude wisely that a single approach to testing software cannot help us so the need of amalgamation of test approaches are inevitable. By hybrid test design we indicate a mixture of different type of meta-heuristics to optimize the business risks of the software. To have a foolproof approach in future we need to start testing the software using hybrid testing techniques in current era to make the future of software testing bright.

REFERENCES

[1] B. Korel, “Automated software test data generation,” IEEE Transactions on Software Engineering, vol. 16, no. 8, pp. 870–879, 1990.

[2] “Dynamic method for software test data generation,” Software Testing, Verification and Reliability, vol. 2, no. 4, pp. 203–213, 1992.

[3] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios, “Application of genetic algorithms to software testing (Application des algorithmes gen´ etiques ´ au test des logiciels),” in 5th International Conference on Software Engineering and its Applications, Toulouse, France, 1992, pp. 625–636.M. Wegmuller, J. P. von der Weid, P. Oberson, and N. Gisin, “High resolution fiber distributed measurements with coherent OFDR,” in *Proc. ECOC'00*, 2000, paper 11.3.4, p. 109.

[4] M. Harman, “The current state and future of search based software engineering,” in Future of Software Engineering 2007 (FOSE 2007). IEEE Computer Society, 2007, pp. 342–357.

[5] M. Harman and J. Clark, “Metrics are fitness functions too,” in International Software Metrics Symposium (METRICS 2004). IEEE Computer Society, 2004, pp. 58–69.

[6] J. Wegener and F. Mueller, “A comparison of static analysis and evolutionary testing for the verification of timing constraints,” Real-Time Systems, vol. 21, no. 3, pp. 241–268, 2001.

[7] J. Wegener, A. Baresel, and H. Sthamer, “Evolutionary test environment for automatic structural testing,” Information and Software Technology, vol. 43, no. 14, pp. 841–854, 2001.

[8] N. Tracey, J. Clark, K. Mander, and J. McDermid, “An automated framework for structural test-data generation,” in Proceedings of the International Conference on Automated Software Engineering. Hawaii, USA: IEEE Computer Society Press, 1998, pp. 285–288.