CHAPTER 11

Conclusion and Future Work

This thesis presents few methodologies and approaches for improving the effectiveness of software testing. The main objective of this research has been to look at the regression testing, performance testing, software testing measurements and software testing effort estimation.

In this dissertation, we dealt with prioritization of regression test suite and arrived at a strategy of prioritization of the regression test suite based on the cost of risk associated with each test case. The test cases are executed based on the priority derived from the cost of risk in the given duration of time. Risk exposure is calculated by considering the probability of fault occurrence and cost of fault. Our future work includes using more components for case studies, performing additional empirical studies to evaluate the effectiveness of our technique, applying metrics (for example, specification traceability and coverage reports) to guide when to stop regression testing, and running pilot project implementing our approach in a production test environment.

We also proposed a method for estimating the cost of fault from developer’s perspective using Neurofuzzy approach. The model gives an integrated approach for calculation of cost of a failure to the developer. This model combines the effect of different parameters so that effective impact can be calculated in spite of giving a biased view with respect to individual parameters. Neurofuzzy systems were considered as a candidate for this work because of the transparency and therefore the validatory nature of the models they produce. The neurofuzzy
system was used for cost estimation because of its transparency, accuracy and consistency; these quantitative results strongly suggest they can be used for cost determination and therefore merit further research. This model was trained with limited set of data and it needs to be rigorously tested and validated with more data.

Performance testing should be considered as an activity that needs to run in parallel to the design and development activities. All these need proper planning to be done and necessary resources allocated for performance engineering right at the beginning of the SDLC. Currently there is no methodology available for estimating the effort required for Performance testing of web applications This theses arrived at a methodology for estimating ball-park % of total SDLC effort required for performance testing.

During performance testing, team face the challenges in Workload Modeling, Load generator and simulation of the user behavior?, Load generator and simulation of the required number of users?, How to ensure that system can support more users? & User Education. This thesis provides answers to these questions by reflecting on case studies of distributed applications and highlighting aspects that should be considered while answering these questions. In this thesis we examined all these performance-testing issues and arrived at an approach to address these issues for certain classes of software systems.

This thesis proposes and details certain quantitative techniques that can be used in verifying performance and scalability of the web application. The proposed techniques are validated by following them during the performance testing of .NET architecture for a complex
application with thousands of users. Some of the performance optimization techniques are also discussed.

Software testing has taken a new dimension in the recent past due to increasing complexity and functionality of software. In critical applications, the effort spent on testing can be as high as 35% to 45% of project effort. Hence it is necessary to define precise processes and process metrics for testing and manage the testing projects objectively. This thesis discusses how process definition and process metrics help to consistently execute testing, measure quality and productivity of testing in order to improve the process.

In practice, time and resources are always limited. When to stop testing is a problem since the first day. We want to use time and resources in the most effective way and gain enough confidence about the quality of the system under test. As a phase of project development, sometime we need to stop testing when we feel that system quality is good enough, and leave time and resources to other tasks. Metrics are useful tools to help make the decision. One example matrix is specification coverage report. When a specific specification coverage target has been reached, we can stop testing with enough confidence.

For large software systems, testing phase seems to have profound effect on the overall acceptability and quality of the final product. A good measure for testability can better manage the testing effort and time. Metrics like Average Cyclomatic Complexity (ACC), Response for a Class (RFC), Coupling Between Objects (CBO) and Depth Inheritance Tree (DIT) are used in measurement of object-oriented testability but none of them is alone sufficient to give an overall reflection of software testability. Thus an integrated measure
considering the effect of all these measures is required to well define the testability. The thesis combines OO software metric values into a single overall value (called Testability Index) that can be used to calculate the testability of a class.

The fuzzy approach is used to integrate these four inputs and arrive at Testability Index for a class. The empirical data of unit testing time for Java classes has been collected and results have been validated. A strong correlation exists between unit testing time and output value of this model i.e. Testability Index. In most software projects managers feel challenge in estimating effort that should be spent on testing. Other side to look at the same problem is to find out if enough rigor has gone into the testing activity and hence what is the confidence on software quality? To be confident on software quality and at the same time to ensure software gets delivered within cost and schedule – Testability Index can be used to estimate testability of software unit (in OO world class). We foresee the following extension of our work. Our experimental basis should be extended. It is desirable to extend our findings to a large number of systems, developed by all sorts of teams using different development methodologies.

The thesis also proposed a single unified measure of software complexity of Object-Oriented software. The proposed model measures the complexity of object-oriented systems based on the four important object oriented metrics – Weighted Methods per Class (WMC), Response for a Class (RFC), Coupling Between Objects (CBO) and Depth Inheritance Tree (DIT). This model integrates the effects of different parameters of object-oriented systems, which can affect the complexity and is not biased towards one particular aspect of the system. This model can be useful in deciding about the order of complexities of different classes.
which in further can help in determining the risk associated with the classes of different complexity level and we can prioritize the classes on the basis of risk associated with them for regression testing. Future work includes an exhaustive empirical investigation of the single unified measure proposed in this thesis.

Underestimating a testing project leads to under-staffing it (resulting in staff burnout), underscoping the quality assurance effort (running the risk of low quality deliverables), and setting too short a schedule (resulting in loss of credibility as deadlines are missed).

The approach proposed in the thesis emphasizes on key testing factors that determine the complexity of the entire testing cycle and gives us a way of translating test creation efforts to test execution efforts. Once the size of the application to be tested is determined in terms of test case points, multiplying the same with productivity figures can arrive at the effort required to execute the same.

The work on this thesis shows that more experience with the use test case point method is needed in order to establish the general usefulness of the method. The method should be applied to very large projects developing real-time applications, and systems that are rich in user interface and outputs and low in algorithmic complexity. One of the difficulties of the method is converting test case points to man-hours. More research is needed to find reliable solutions to this problem.