Chapter 1

Introduction

The increasing use of software is giving rise to development of highly complex software systems. Further, software systems are required to be of high quality as a defect can have catastrophic effect on business as well as human life. Software development life cycle includes testing with high priority to ensure production of quality software. Testing is defined as the process of executing a program with the intension of finding error[Mye79]. Software testing is an expensive process of the software development life cycle consuming nearly 50% of development cost. Testing is required to verify and validate customer requirements, to validate incomplete/abstract/changing requirements, to uncover errors and build faster and better quality software[Bei02]. There are two strategies to testing, white box and black box. In white box testing, test cases are developed to exercise internal paths and structure of the Software under Test(SUT) thereby ensuring code coverage. Black box testing is done based on requirements and specification. Hence, test cases are built to ensure that software meets user specifications.

Specification refers to the intended behaviour of a system. Specification provides information for testing by focusing on aspects of the system that have to be implemented. Advantage of testing based on specification are better understanding of the objectives and behaviour of the system, early detection of defects, reducing effort and cost in debugging and rectification and easy verification of the system(traceability). Specification of a system can be captured using formal/informal techniques. Formal techniques involve using a formal language like B and Z. The advantage of a formal technique is the mathematical foundation offering means to reason on specification; but, it is difficult for professionals with less incli-

nation on mathematics. Probably, that's the reason why the technique is less used in comparison to UML: the Unified Modeling Language with graphical notations consisting of various diagrams to capture requirements.

UML is a widely accepted standard for modeling software systems. It consists of a set of modeling concepts(primitives) to support an object oriented approach to software development. UML consists of a set of diagrams that model both static and dynamic behaviour of a system. Various aspects of the system are elaborated at different levels of abstraction using diagrams like use case diagram, class diagram, activity diagram, sequence diagram and state diagrams.

The objective of this work is to investigate testing of software systems developed from UML specifications. Smith and Robson [SR92] discuss four levels of testing object oriented systems: the algorithm level, to test methods, the class level, to test interactions between class attributes and functions, the cluster level, to test interactions between classes, and system level that tests the entire system. Jorgensen and Erickson[JE94] in their work divide testing into unit, integration and system level. This work focusses on testing a system to ensure conformance of system requirements.

Testing of software system is critical as well as expensive. There has been attempts to reduce test efforts without compromising the quality of software. Specification based testing is viewed in terms of testing scenarios, where a scenario is a sequence of activities. This work focusses on scenario generation and selection of representative scenarios for testing so that test effort is reduced at the same time not compromising on quality. In the next section, problems investigated and reported in the thesis are detailed.

1.1 Research Problem

Problem 1: An approach to ensure consistency of specification : The inherent overlapping of elements of different UML diagrams used for system specification may give rise to specification inconsistency. Ensuring consistency in UML diagrams is pivotal since the quality of a system is highly dependent on its specification. This problem is being investigated and the resultant method is supported by a tool. The research reported here envisions a rule based approach for consistency checking and a prototype tool is developed and the viability is demonstrated

with case studies.

Problem 2: A technique for test scenario generation from UML activity diagrams : Activity diagrams elaborate scenarios related to each use case. A scenario is defined as the sequence of activities starting from the start activity to the end activity. The number of scenarios generated from activity diagrams using automated generation is exhaustive, especially in case of concurrent activities. Hence, there is need to generate scenarios in an optimized way.

Problem 3: Technique to prioritize scenarios : The order in which test cases are executed has bearing on both testing time as well as test coverage. An ineffective test case ordering could be time consuming to reveal critical faults. This motivates the study of techniques to introduce an ordering of test cases that reveal defects as early as possible while widening test coverage.

Problem 4: Technique for test scenario selection : Given constraints of cost and time, it may not be always possible to run the entire set of test cases. Hence, there is need to introduce techniques that provide a selection of effective subset of scenarios for testing.

Problem 5: Approach for test management : Given the size of software systems, and the constraints of cost and time, there is need for better test management. This thesis proposes an ontology based test scenario management for querying on test scenario repository.

1.2 Contribution

The contributions of this thesis can be summarized as below:
Consistency checker to deal with Problem 1. A methodology that follows transformational approach is proposed for consistency checking. Well Formedness Rules(WFRs), following structural relationship among UML diagram elements are extracted and applied on UML diagram repository for checking. The usage of this method is demonstrated with a prototype tool developed for the purpose.

The benefits of the proposed approach are two fold. One, consistency rules

are defined as Well Formedness Rules which are generic in nature. Second, the transformational approach is a common format to store model elements. Also, the relational model has inherent mechanism to check for inconsistency through primary and secondary keys as well as triggers and assertions which can be used to enforce consistency.

Test Generation technique to deal with Problem 2. Functional requirements are recorded using use case diagrams. Each use case is elaborated using activity diagrams. Thus, each path in an activity diagram from the start activity to the end activity constitutes a scenario. A modified Depth First Traversal algorithm is proposed to generate scenarios from activity diagrams given that the number of scenarios generated could be large, especially in cases where concurrent activities are involved. Dependency information among activities are used to reduce the number of scenarios generated. Concurrent activities are annotated with either priority or level information. So, activities having higher priority are to be performed before activities having a lower priority. Alternatively, all activities at a higher level have to be completed before activities at a lower level.

The benefit of the technique is that based on dependency information, the number of scenarios generated can be reduced. This is especially effective in case of concurrent activities represented using fork/join constructs in an activity diagram.

Test Prioritization technique to deal with Problem 3. Given a large number of test scenarios(test cases), the order of execution gains importance given constraints of cost and time. Defects have to be detected early in the testing process so that feedback can be given at the earliest for bug fixing. In this work, two types of priority are considered: user defined and structural complexity. In the former, use cases are prioritized by the customer according to business needs. Secondly, structural priority due to diagram structure is obtained by considering interactions between primitives of use case diagrams, due to include, extend and generalization features. In case of scenarios, priority of a scenario is calculated by considering the primitives of the activity diagram, namely, activity, fork/join, branch/merge and concurrent activities. The combined priority of the use case and scenario is taken as the priority of the scenario.

The advantage of the technique is that besides customer input on priority of

use case, priority calculation is automated as it is based on the primitives of the UML use case and activity diagrams. It's observed that the priority obtained by the proposed method is realistic as it considers inputs from user as well as domain complexity reflected through specification diagrams.

Test Selection technique to deal with Problem 4. Prioritization provides an ordering of scenarios. However, given constraints of cost and time it may not be possible to test all scenarios. Hence, there is need for selecting scenarios for testing. A close look at scenarios reveal the commonality among scenarios so that testing of common scenarios should be avoided and more dissimilar scenarios are to be selected for speeding the test process and for revealing critical defects early.

In this work, three techniques that use distance measures to determine similarity are presented. The first technique is based on Levenshtein distance[Lev65] as a measure of dissimilarity between scenarios. For each pair, the Levenshtein distance is calculated. The least value indicates scenario pairs that are least dissimilar. The second technique for test scenario selection is based on Longest Common Subsequence. A subscenario is a contiguous set of activities within a scenario. The technique looks at similarity between scenarios in terms of its length and position of the common subscenarios in the scenario. The third technique is based on clustering of scenarios based on their similarity computed by a distance metric like Levenshtein distance. Agglomerative Hierarchical Clustering is used to cluster scenarios based on a certain degree of similarity defined by a metric.

The proposed techniques are comprehensive, well defined by distance metric and algorithmic, so that these are computable and can be automated with ease. They provide a clear cut guideline to choose representative scenarios for testing.

Ontology for Test Management to deal with Problem 5. Management of test scenarios involves ordering and selecting a set of scenarios for testing with the objective of fulfilling a criteria like maximizing coverage or discovering defects as early as possible. For this, there is a need to maintain knowledge on the main activities in the domain and the interactions between them. Ontologies provide a mechanism to share and reason on knowledge that is captured. Ontologies are used in this work to aid in test management.

1.3 Terminology

The terms related to software testing used in this dissertation comply with the Standard Glossary of Terms used in Software Testing V.2.0, Dec, 2nd 2007 produced by the Glossary Working Party International Software Testing Qualifications Board. Some of the most used terms are presented in this section.

code coverage	An analysis method that determines which parts of the software have been executed(covered) by the test suite and which parts have not been executed, e.g. statement coverage, decision coverage.
priority	The level of (business) importance assigned to an item, e.g. defect.
system testing	The process of testing an integrated system to verify that it meets specified requirements.
test scenario	A document specifying a sequence of actions for the execution of a test. Also known as test script or manual test script.
test suite	A set of test cases for a system under test.
test coverage	The degree, expressed as a percentage, to which a specified coverage item has been exercised by a test suite.

Regarding UML and its related terms, the definitions used in OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2 are adopted:

Actor	An actor specifies a role played by a user or any other system that interacts with the subject.
Use case	A use case is the specification of a set of actions performed by a system, which yields an observable result that is of value for one or actors or stakeholders of the system.
Activity	An activity execution is the execution of an activity, ultimately including the executions of actions within it.
Scenario	A sequence of activities from an activity diagram starting from the start activity to the end activity constitutes a scenario.

1.4 Thesis Structure

Organization of the thesis is shown in Figure 1.1. Chapter 2 surveys recent work on software testing with particular reference to consistency checking, test scenario generation, prioritization, selection and ontology generation within the context

of using UML diagrams for specification. Chapter 3 discusses a transformational approach to consistency checking. Techniques for test scenario generation, prioritization and selection is discussed in Chapter 4. In Chapter 5, generation of an ontology for test management is the focus. Chapter 6 presents the tool and experiments conducted to evaluate the performance of the test scenario generation, prioritization and selection technqiues. Finally, Chapter 7 concludes the work and discusses directions for future work.

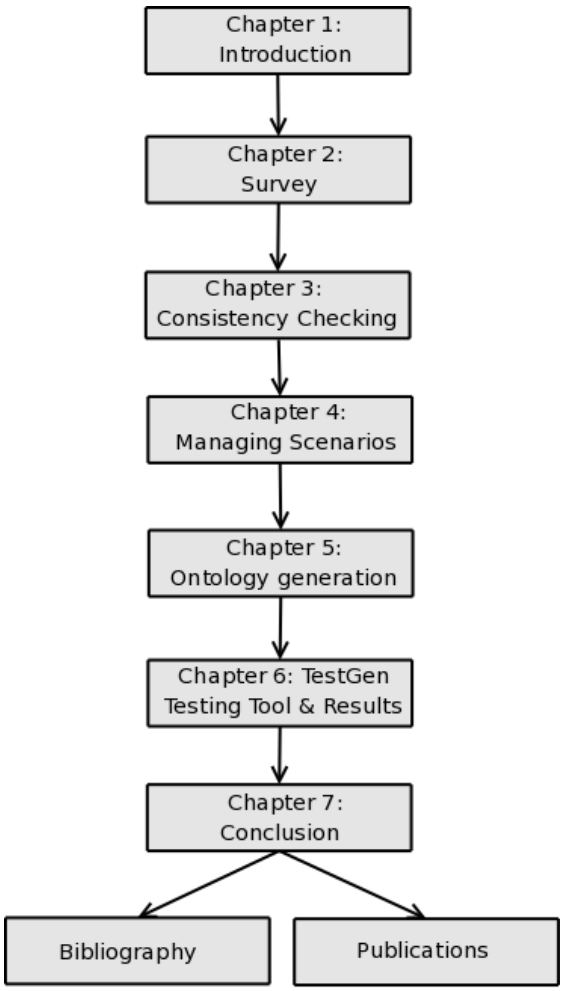

Figure 1.1: Outline of Thesis