CHAPTER 3

SYSTEM ARCHITECTURE FOR SAFE COMPOSABLE TESTING MODEL

3.1 INTRODUCTION

Testing consumes lot of time and lot of challenges for different applications. Testing for safety critical system is very tedious and involves extensive risk if proper testing is not done. Formal specification is done to check the properties of the system modelled for safety properties. Testing actually aims at showing the deviation between expected and actual behaviour of the system. Main goal of testing is detecting failure in identifying the observable differences between implementation of what is needed on the basis of specification. Model based testing is a complete variant of testing for modern applications that actually encode the expected behaviour of a system and the behaviour of its environment. Input and output pairs of the system are taken as the test cases for the model which is interpreted for the system under test.

A complete testing model where verification assists validation is specified in system architecture which is important to safety constraints by formal specification and composing the requirements into a component model thus analysing the properties in the system for which the model is being built. A complete system with component model is built as a simulated model with timed automata and Petri Nets and model checking is done with temporal logic. The complete verified system is tested for the system under test with
test case specification generating test cases and scripts and ensuring complete code and transition coverage. A complete model for verification and validation is modelled as safe composable testing model which is represented in system architecture as in Figure 3.1.

Figure 3.1 System Architecture for safe composable testing model
3.2 ARCHITECTURE DESCRIPTION

Architecture for safe composable testing model consists of the following functions as represented in Figure 3.1.

1) Requirement Generation
2) Verification
3) Extended Finite State Machine and Petri Net
4) Test Selection Criteria
5) Test Case Specification
6) Test Cases
7) Test Script
8) Test Coverage

3.2.1 Requirement Generation

System Engineering is a comprehensive discipline which consists of requirement engineering, design and specification, coding, testing and deployment. Requirement is the first step towards the development of any software engineering project. Gathering requirements is very tedious and time consuming. Requirements gathered from analyst have to undergo vigorous review to avoid confusion, consistency, repetition and extra requirements. Requirements gathered are the main document in continuing with the next phases of software development life cycle. The requirements gathered are documented in software requirement specification (SRS) manual. The requirements are documented in terms of functional and non-functional requirements which are not specified explicitly. A requirement is a statement about an intended product that specifies what it should do or how it should be
done. For requirements to be effectively implemented and measured, they must be specific, unambiguous and clear.

![Diagram of Types of Requirements](image)

**Figure 3.2 Types of Requirements**

Different types of requirement are specified in Figure 3.2 which contains three types of requirements namely 1) Business requirements 2) User requirements 3) System requirements. Business requirement is established with the scope of project and business objectives. User requirements are established with user point of view and user goals. System requirements consists of functional and non-functional requirements which describes the functional aspects and non-functional aspects like security, safety, performance, reliability, scalability and many other aspects.

### 3.2.2 Verification

Verification starts at the initial stages of the development. Verification can be established with models checking for properties of the system. Verification deals with checking the system phases for any deviation from product specification. The act of reviewing, inspecting and testing a product with documentation to establish whether a product, service or system
meets regulatory or technical standards constitute verification in software engineering.

### 3.2.3 Component Model

A component model is built with basic building blocks which constitute the connection among other components depending on the dependency between components. When a model is being built, care has to be taken in composing the components with common interfaces, input and functionality. Design of components has to be done in such a way that it ensures low coupling and high cohesion for an effective design.

### 3.2.4 Extended Finite State Machine and Petri Net

Verification of system can be done with model driven engineering. With the complexity of modern system, it is very crucial to provide methods that enable debugging and testing of system prior to implementation and deployment. One way to approach the challenge is to build an executable model of the system. Developing a model and simulation of the model leads to many new insights into the design and operation of the system and results in a simpler design. An executable model leads to a more complete specification involving systematic investigation of scenarios which significantly reduces errors in development. An executable model can be built with extended finite state machine and Petri Net which are both simulation models.

### 3.2.5 Test Selection Criteria

Explicit graphs can be developed containing nodes and arcs for transition based models where graph coverage criteria can be used to control test generation. The coverage criteria commonly used are all nodes, all
transitions, transition pairs and cycles. The main goal of the system under test is to find faults in the system. One of the most common fault-based criteria is mutation coverage. This technique involves mutating the faults and generating tests that distinguishes the mutated model and original model. There is a correlation between faults in the model and mutated faults. Structural testing involves combination of many Boolean decisions within the model.

3.2.6 Test Case Specification

Test processes determine whether the development products of a given activity conform to the requirements of that activity and whether the system and/or software satisfy its intended use and user needs. The scope of testing encompasses software-based systems, computer software, hardware, and their interfaces. The test case specification is developed in the development phase by the organization responsible for the formal testing of the application. A test case specification describes the purpose of a specific test, identifies the required inputs and expected results, provides step-by-step procedures for executing the test, and outlines the pass/fail criteria for determining acceptance.

3.2.7 Test Cases

Test cases are a set of conditions or variables under which a tester will determine if a requirement upon an application is partially or fully satisfied.

- There must be at least one test case for each requirement for traceability.
- Each test case will have a known input and an expected output, worked out before the test.
3.2.8 Test Scripts

Test Scripts are a set of instructions that will be performed on the system to test that the system functions as expected. These steps can be executed manually or automatically.

- Can either be written using a special automated functional GUI test tool or in a well known programming language?

- Automated test tools can test many areas of system functionality such as the user interface, performance of the system, the system code and the requirements.

- Automated testing has the advantage over manual testing in that it is easily repeatable, and thus is favoured when doing regression testing.

3.2.9 Test Coverage

Test Coverage is an important part in software testing and software maintenance and it is the measure of the effectiveness of the testing by providing data on different items.

Test coverage analysis is done for the following reasons.

1. To find the areas in specified requirement which is not covered by the test scenarios and cases.

2. By determining the test coverage we can create more test cases to increase our test coverage.
3. By performing the test coverage we can measure how much testing is covered. This indirectly means the check on quality of the application.

4. We can also identify some useless test cases which don’t have meaning in being executed and thus omit them.

5. Testing life becomes smooth by managing the risk based testing approach.

6. Traceability between the requirements and the test cases can be achieved by this technique.

7. Impact analysis and change tracking can be determined if we have proper test coverage.

3.3 SUMMARY

Software architecture for safe composable model testing involves on the whole an efficient methodology for generating component model from enormous requirements. The requirements are simulated by generating a model which is checked for testing by choosing the right characteristics of model based testing. Next Chapter deals with the design of timed automata with the architecture model.