Chapter 6

Results and Prototype Tool

A prototype tool that implements the concepts discussed in the previous chapters was built and the concepts were tested with case studies. The tool consists of automated scenario generation from UML diagrams, prioritization and selection of scenarios. The results obtained using the tool on the case studies are presented in this chapter.

6.1 Introduction

The ideas presented in Chapter 4, Sections 4.2, 4.3 and 4.4 are tested on case studies and results are presented in this chapter. The experiments attempt to answer the following:

- Are all scenarios preferred by the customer generated using the scenario generation algorithm proposed in the work(ScenGen, Priority-Based-ScenGen and Level-Based-ScenGen in Chapter 4, section 4.2)

- Compare performance of scenario prioritization obtained due to combination of weights assigned by the user and calculated using structure based complexity to optimal, coverage based and random prioritization.
Performance of selection techniques taking into consideration the percentage of scenarios selected vs. percentage of defects detected.

A testing tool called TestGen (Test Generation) was built to support the techniques presented in Chapter 4. The tool implements techniques to generate scenarios from UML activity diagrams, as well as techniques for test scenario prioritization and selection.

In this chapter, a prototype of a testing tool and results produced is presented. Section 6.2 gives a quick overview of the tool and the functionality of the tool is given in Section 6.3. Section 6.4 presents the case studies used to study the effectiveness of the proposed techniques. The results of applying prioritization and selection techniques is also discussed. To end, some conclusions and final remarks are made in Sections 6.5 and 6.6.

6.2 Architecture of TestGen

A tool, TestGen, was built to implement the techniques proposed in this work. The tool consists of two parts, the preprocessing part (scenario generation) and test case generation. This work focuses on the former and the latter is left for future work. The architecture of TestGen is presented in Figure 6.1. Specification of the system in the form of UML use case and activity diagrams is the input to the tool. The TestGen tool, extracts required use case and activity diagram primitives from the diagram files. First, consistency of specification is checked by the ‘Consistency Checker’. Once consistency among diagrams is ensured, the specification can be used for scenario generation. UML activity diagrams are used to generate scenarios.

The next step involves prioritization of use cases and scenarios. Customer inputs on priority of use cases is obtained. Also, structure based priority of use cases
and scenarios is calculated using an automated technique based on the primitives of the use case and activity diagram respectively. Scenario selection is essential when there is need to pick a subset of scenarios for testing while test engineers work with time limitation in contrast to high demand on quality. Techniques include selection based on distance measures calculated between scenarios (Levenshtein distance, Common subscenario approach, Clustering). The use case diagram of the tool shows the actors and the use cases they interact with (Figure 6.2).

### 6.3 Functionality supported

The TestGen tool support the following functionality:

- Extracting necessary details from UML use case and activity diagrams.
- Customer assignment of priorities for use cases.
- Automated computation of use case priorities according to primitives.
- Automated computation of scenario priority by primitives.
- Approach for selecting a subset of scenarios using distance measures (Levenshtein based, common substring based).
- Clustering based approach to scenario selection.
Figure 6.2: Use case diagram for the testing tool
The tool requires as input the folder containing all UML diagrams related to the Software Under Test (SUT). This folder contains the use case diagrams and activity diagrams related to each use case diagram (Figure 6.3).

![Figure 6.3: Testing tool: TestGen - Selecting the Software Under Test](image)

The use cases are extracted from the 'UseCase' folder. Then, for each use case the corresponding activity diagrams are selected and linked (Figure 6.4).

Figure 6.5 shows a screen shot of the tool, TestGen. There are five main windows: the system window, the model window, the details window, the operations window and the scenario window.

- **System Window.** The system window is used to represent the actors and interacting use cases in a hierarchy (tree structure). At the first level is the root node, namely, Root. At level one, are the actors of the system. Related to each actor are the use cases they interact with. Further, inter use case relationships <include>, <extend> and <generalization> relation are shown. The objective of the system tree window is for easy understanding of the relationship between the actors and the functionality of the system.
- **Model Window.** The model window shows the activity diagram(s) related to each use case.

- **Details Window.** On request on each item in System window, the 'Details Window' shows the information on the entity e.g. for a use case entity the details are the name of the entity, the type, related actors and priority information.

- **Operations Window.** Operations window gives a set of operations that can be applied on use cases. It includes, select, project, union, intersect, difference and alike. These relational operators can be applied to use cases.

- **Scenario Window.** Each use case consists of a set of scenarios. The scenario window lists all scenarios related to a use case.

Scenarios generated for each use case from corresponding activity diagrams are stored in .xml file.
Figure 6.5: Testing tool: TestGen
6.3.1 Modules of the System

The proposed system integrates the following modules:

- **Preprocessing.** This module prepares data from the UML documents. Specification captured using UML use case and activity diagrams are stored in .xmi format. In the preprocessing stage, first the use case diagrams are used to extract details about the actors, use cases and the relationship between actors and use cases as well as between use cases. The output is a .xml containing actors, use cases and the relationship between them. Second, activity diagrams are used to extract activities and the relationship between activities. Thus, the output of this process is the recording of each activity with its type (e.g. activity, start/stop activity, decision, fork/join) as well as the activity interaction pairs.

- **Scenario Generation.** The input to this step is the .xml file containing details about activities and their relations with reference to each use case. Scenarios are generated using modified Depth First Traversal. Scenarios for each use case diagram is generated from the corresponding activity diagrams as discussed in Chapter 4, Section 4.2.5.

![Figure 6.6: Obtaining priority from the Customer(default value assigned to 1, lowest priority)](image)

Figure 6.6: Obtaining priority from the Customer (default value assigned to 1, lowest priority)
• **Prioritization.** Priority of each scenario is calculated thus:

1. Customer inputs on priority is obtained for each use case. The customer gives priority to use cases on a scale of 1-10. Example is shown in Figure 6.6.

2. Structure based priority is calculated using the primitives of the use case and activity diagram. The results of structure based priority is shown in Figure 6.7. The details on calculation of structure based priority is already discussed in Chapter 4, Section 4.3.8.

3. Combined priority of use cases and activities is used to prioritize scenarios belonging to a test suite. The relative weights for consideration of customer and structure based priority are given by the test engineer and the combined priority calculated.

• **Selection.** Involves selecting a subset of scenarios to form a test suite. Dissimilar scenarios are selected based on distance measures to form a representative set of scenarios using one of three techniques as mentioned in Section 4.4.

• **Ontology Generation.** The information captured about the domain, namely, actors, use cases, scenarios and activities, are used to generate an ontology. The .owl file generated can be used with an ontology editor like Protege(discussed in Chapter 5).

### 6.4 Case Study

In this section, the results obtained on case studies used to validate the proposed methodology empirically are discussed. Two case studies have been considered, namely, the CoffeeMaker system, and a Supermarket Automation System(SAS).
Figure 6.7: Computing Structure based Priority
The goal of the case studies is to present results obtained using test generation, prioritization and selection techniques proposed in this work.

**Faults**

Faults detected during testing are associated with either real defects or defects injected manually into the software [MCHI97]. In case of manually injected faults, care was taken to try to simulate as closely as possible typical programming errors. Table 6.1 shows a sample of faults, with a short description for each of them.

<table>
<thead>
<tr>
<th>Fault</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1</td>
<td>Injected</td>
<td>Error in reading file</td>
</tr>
<tr>
<td>F2</td>
<td>Injected</td>
<td>Error in writing file</td>
</tr>
<tr>
<td>F3</td>
<td>Real</td>
<td>Incorrect path</td>
</tr>
<tr>
<td>F4</td>
<td>Injected</td>
<td>Missing Functionality</td>
</tr>
<tr>
<td>F5</td>
<td>Real</td>
<td>Missing error handling</td>
</tr>
<tr>
<td>F6</td>
<td>Injected</td>
<td>Wrong expression</td>
</tr>
<tr>
<td>F7</td>
<td>Injected</td>
<td>Wrong condition</td>
</tr>
<tr>
<td>F8</td>
<td>Injected</td>
<td>Interface specification</td>
</tr>
<tr>
<td>F9</td>
<td>Injected</td>
<td>Wrong algorithm</td>
</tr>
<tr>
<td>F10</td>
<td>Real</td>
<td>Incorrect Message</td>
</tr>
</tbody>
</table>

**6.4.1 Results of the Case Study**

This section describes the experimental results obtained when the prototype tool, TestGen, was used to test two case studies, namely, CoffeeMaker and the Supermarket Automation System. First, scenarios were generated. Then customer inputs on priority of use cases was obtained. Further, priority values based on structural primitives were calculated. The combined priority of the use case and scenario is the priority of the scenario. The ordered scenarios were used to obtain results of testing. Test selection was done using the three techniques and the selected scenarios were executed. Testing results and evaluation of the same is presented in this section.
Test Generation

Objective: Each use case has at least one activity diagram elaborating it. Scenarios need to be generated for each use case. A collection of such scenarios form the entire set of scenarios related to the system.

Input and Output: The input to the test generation process is activity diagrams related to each use case diagram stored as .xmi files. The output is scenarios related to each use case.

Strategy followed: First, for each use case, related activity diagrams were designed. In case of concurrent activities, where possible, level/priority details were annotated. A modified DFS algorithm discussed in Chapter 4 is used to generate scenarios.

Checking Test Adequacy:
Test adequacy criterion specifies the requirements of a particular test. To measure the quality of test scenarios produced, the following test adequacy criterion [CMK08] is used:

- Activity Coverage requires that all the activities in the activity diagram be covered. Activity coverage is calculated as the ratio between the covered activities and all the activities in the activity diagram.

- Transition Coverage requires that all the transitions in the activity diagram be covered. Transition coverage is calculated as the ratio between the checked transitions and all the transitions in the activity diagram.

- Path Coverage requires that all the paths in the activity diagram be covered. Path coverage is the ratio between the traversed paths and all the paths in the activity diagram.
**Prioritization**

*Objective*: Given a collection of scenarios, there is need for prioritization to widen coverage and detect defects early.

*Input and Output*: The input to the prioritization process is scenarios generated from all activity diagrams.

*Strategy followed*: Customer priority of requirements (here use cases) was obtained. The customer ranked use cases on a scale of 1-10. Further, priority based on structural primitives of use case and activity diagram was calculated (Figure 6.7). A combined priority for each use case was calculated based on weights given to customer priority and structure based priority. The prioritized list of scenarios thus obtained was tested on the system with each scenario having a minimum of at least one test case.

*Evaluation Measure*:

Results of prioritization are compared against optimal, coverage and random ordering to study effectiveness using the APFD metric discussed in Chapter 2, Section 2.6.2.

**Selection**

*Objective*: Given a collection of scenarios, a subset of scenarios need to be selected for testing to meet constraints of cost and time.

*Input and Output*: The input to the selection process is scenarios generated from all activity diagrams.

*Strategy followed*: Scenario selection involves selecting a subset of scenarios for testing. Results are compared against random selection. In this work, picking one of two scenarios with minimum distance is done based on one of the following: priority (structure based and combined priority), type and random.
Evaluation Measure:

The Average Percentage of Faults Detected (APFD) metric discussed in Section 2.6.2, is used for evaluating performance of the technique.

6.4.1.1 Case Study - I: CoffeeMaker

The first case study is a CoffeeMaker example. The requirements of the system is as follows: The coffee maker remains in wait state when not in use. There are a total of six options:

1. Add a Recipe
2. Delete a Recipe
3. Edit a Recipe
4. Add Inventory
5. Check Inventory
6. Purchase Beverage

A maximum of three recipes, each being unique, can be added to the CoffeeMaker. A recipe consists of a name, price, units of coffee, units of milk, units of sugar and units of chocolate. The price of an item is an integer. Recipes may be added, edited and deleted. Each recipe being unique, it is not possible to add a duplicate recipe. Also, it is possible to delete only an existing recipe from the list of recipes. The recipe to be deleted is chosen by name. A recipe can be edited by changing the price, units of milk, sugar, coffee or chocolate. Again, the recipe to be edited is chosen by name. After completion of the action, the CoffeeMaker returns to waiting state.
Inventory can be added to the system. Inventory includes units of coffee, milk, sugar and chocolate, units of measurement being integers. On addition of inventory, message is printed and the CoffeeMaker returns to waiting state.

To buy a beverage, a user needs to deposit enough change and select the item required. In case the change is insufficient, the user is prompted for more change. The user may enter change, or select the cancel button, in which case, the change is returned. In case of enough change and the inventory being insufficient, the CoffeeMaker prints a message and prompts for selection of another item. Cancellation of the order returns the change. In case of sufficient change and sufficient inventory, coffee is prepared and dispersed. Also, change is returned to the user, if any. The use case diagram of the CoffeeMaker is shown in Figure 6.8. The coffeemaker case study consists of 900 lines of code.

Figure 6.8: Use case diagram for the ‘CoffeeMaker’
Results

Scenario Generation

The UML specification consists of 6 use cases. Each use case has one activity diagram related to it. The system consists of a total of 50 activities and 73 transitions. A total of 23 scenarios were generated for the CoffeeMaker system. All scenarios preferred by the customer was obtained.

Prioritization

The use cases were prioritized taking a weighted sum of customer as well as structure based priority. Equal weight has been given for both customer and structure based priority in this work. The results of prioritization applied to the CoffeeMaker case study were calculated using the metric, APFD, discussed previously in Chapter 2, Section 2.6.2.

Figure 6.9: Comparison of APFD values obtained for CoffeeMaker System

Figure 6.9 shows the APFD values obtained for the technique, compared to optimal, statement coverage and random prioritization. The same results are summarized in Table 6.2. The results of the technique proposed in this work using customer inputs as well as structural inputs of both use case and activity diagrams are equal to coverage based prioritization. Random prioritization performs substantially worse with APFD value being 10% lower. The automated technique is thus an effective technique in prioritizing scenarios. Also, coverage information
Table 6.2: Results in terms of APFD values for CoffeeMaker System

<table>
<thead>
<tr>
<th>Prioritization Technique</th>
<th>APFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>.98</td>
</tr>
<tr>
<td>Combined Priority</td>
<td>.93</td>
</tr>
<tr>
<td>Coverage</td>
<td>.93</td>
</tr>
<tr>
<td>Random</td>
<td>.79</td>
</tr>
</tbody>
</table>

is not available in the case of a new set of scenarios to be tested. Thus, besides requiring minimal effort, the automated prioritization technique can be used along with other prioritization techniques to aid prioritization of scenarios obtained from UML use case and activity diagrams.

Scenario Selection

![Graph](image)

Figure 6.10: Defect detection using Levenshtein distance as distance measure for selection

In the first case, levenshtein distance was used as a distance measure to determine similarity among scenarios. The curve shows the relations between percentage of scenarios tested to percentage of defects detected. The upper curve shows the result for Levenshtein based technique as described in Section 4.4.4. The lower curve shows the result for random selection of scenarios. The aim is to detect as many defects early and by testing less number of scenarios. Figure 6.10 shows the results of scenario selection using levenshtein distance. In case of Levenshtein based selection, 60% of scenarios selected give 100% defect detection when com-
Figure 6.11: Defect detection using Levenshtein distance as distance metric for selection-applying different methods for picking one of two scenarios.
pared to random selection which requires 100% testing to detect all defects. The percentage of Levenshtein distance spaced defect discovery is persistently more compared to random. Further, it is noted that 60% of scenario testing captures 100% defects indicating Levenshtein distance is powerful in discovering defects. Hence, Levenshtein based selection is recommended for selection of scenarios.

Different techniques are used to pick one of the two similar scenarios (computed with Levenshtein distance), like, structure based priority, combined priority, type of scenario and random selection as shown in Figure 6.11. The results of the techniques are compared with random selection. Results show that structure based priority, combined priority and type data may be used effectively to better results of selection. Random picking of one between two scenarios may not always give best results.

Figure 6.12: Defect detection using similarity measure based on common subsce-
narios as distance measure for selection

In the second case, metric based on common subscenarios was used to de-
termine similarity among scenarios. Figure 6.12 shows the results of scenario selection. Upper curve shows the rate of defect detection in case of using the similarity measure whereas the lower curve shows the result of random selection. With increasing percentage of scenarios, the number of defects detected increases. 60% of scenarios selected using the similarity measure detects 100% defects. Here also, the similarity measure spaced defect discovery is persistently more compared
Figure 6.13: Selecting one of two similar scenario based on similarity measure
to random selection of scenarios.

Again, different techniques have been used to pick one of the two similar scenarios. Results show the difference in defect detection rate of the technique for picking one of two scenarios between priority based, type and random. It is observed that random selection performs worse than other techniques (Figure ??).

The third technique used for selection of scenarios is clustering. Here, percentage of scenarios are plotted in the x-axis against the percentage of defects detected in the y-axis. One scenario is selected as representative of the cluster randomly. Figure 6.14 shows the results of selecting scenarios for testing. Clustering based selection is better than random selection as shown.

![Figure 6.14: Clustering based selection of scenarios](image)

### 6.4.1.2 Case Study - II: Supermarket Automation System

The second case study considered is a Supermarket Automation System (SAS). SAS deals with the automation of several activities involved in the management of a supermarket, ranging from business to market activities. Areas of automation include: billing, product purchase, personnel management, inventory management, feedback collection and report generation. SAS consists of 19 use cases, with a minimum of one activity diagram for each use case.

There are two types of users, namely, the administrator and the sales people. The administrator manages staff details, creates user access for sales people, keeps
Figure 6.15: Use case diagram for 'Supermarket Automation System(SAS)’
track of inventory as well as generate various reports. The sales persons can only perform the action 'Billing'. The functionality of the system include: adding, editing and deleting products and suppliers, setting reorder limits, intimation in case a product reaches the reorder point, generating purchase orders, managing user accounts, maintaining personnel details, perform billing of products and generating various reports. The use case diagram(Figure 6.15) shows the requirements of the system.

6.4.1.3 Results

Scenario generation

Using the test generation algorithm, a total of 102 scenarios were obtained. The case study consists of 10 classes and 114 activities. Each test scenario has a minimum of one test case. All scenarios preferred by the customer was obtained.

Prioritization

Use cases were prioritized taking a weighted sum of customer as well as structure based priority. Equal weight has been given for both customer and structure based priority in this work. The results of prioritization applied to the SAS case study were calculated using the metric, APFD.

Figure 6.16: Comparison of APFD values obtained for SAS

Figure 6.16 shows the APFD values obtained for the technique, compared to optimal, statement coverage and random prioritization. The same results are
Table 6.3: Results in terms of APFD values for SAS

<table>
<thead>
<tr>
<th>Prioritization Technique</th>
<th>APFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal</td>
<td>.80</td>
</tr>
<tr>
<td>Combined Priority</td>
<td>.78</td>
</tr>
<tr>
<td>Coverage</td>
<td>.77</td>
</tr>
<tr>
<td>Random</td>
<td>.68</td>
</tr>
</tbody>
</table>

summarized in Table 6.3. The results of the technique proposed in this work using customer inputs as well as structural inputs of both use case and activity diagrams are better than coverage based prioritization. Random prioritization performs substantially worse with APFD value being 10% lower. The automated technique is thus an effective technique in prioritizing scenarios. Besides requiring minimal effort, the automated prioritization technique can be used along with other prioritization techniques to aid prioritization of scenarios obtained from UML use case and activity diagrams.

Scenario Selection

![Defect detection using Levenshtein distance as distance measure for selection](image)

Figure 6.17: Defect detection using Levenshtein distance as distance measure for selection

In the first technique, levenshtein distance was used as a distance measure to determine similarity among scenarios. Figure 6.17 shows the results of scenario selection using levenshtein distance. Upper curve shows the rate of defect detection in case of using the similarity metric whereas the lower curve shows the result of
Figure 6.18: Selecting one of two similar scenario based on Levenshtein distance
random selection. With increasing percentage of scenarios, the number of defects detected increases, with levenshtein based selection curve showing higher rate of defect detection.

Again, different techniques are used to pick one of the two similar scenarios, like, structure based priority, combined priority, type of scenario and levenshtein based random selection as shown in Figure 6.18. The results of the techniques are compared with random selection. Results show that similarity selection based on levenshtein distance as a similarity measure is better than random selection of scenarios. The defect discovery rate is persistently more when compared to random selection.

In the second case, similarity measure based on common substrings was used to determine similarity among scenarios. Figure 6.19 shows the results of scenario selection. The upper curve shows the result for similarity measure based selection whereas the lower curve shows the result for random selection of scenarios. The aim here is to detect as many defects as early as possible by testing least number of scenarios. The percentage of similarity measure based defect discovery is persistently more compared to random.

![Figure 6.19: Defect detection using similarity measure based on common subsccenarios as distance measure for selection](image)

Different techniques used to pick one of two similar scenarios shows better results when based on structure based priority, combined priority and type of sce-
Figure 6.20: Defect detection using similarity measure based on common subscenarios as distance measure for selection-applying different methods for picking one of two scenarios
nario as compared to random selection (Figure 6.20). This indicates the advantage of using one of the measure to pick scenarios instead of random.

The third technique used for selection of scenarios is clustering. Figure 6.21 shows the results of selecting scenarios for testing. Here, again a representative scenario is selected from each cluster. In this case, results are found to be close to random selection. This is due to the fact that selecting one scenario from a cluster is equivalent to picking a random scenario. Hence, this technique is best suited for regression testing where once an error is detected in a scenario, then similar scenarios can be determined and selected for further testing.

![Figure 6.21: Clustering based selection of scenarios](image)

### 6.4.1.4 Threats to validity

As with case studies, a number of threats affect the validity of the results. The main threats can be classified as:

- **Internal validity threats**, concerning factors that may have affected the experimental measures. The limited number of subjects to get input in case of customer priority, the software system chosen for the case study and the learning effect may all have influenced the results.

- **Construct validity threats**, concerning the relationship between theory and observation. The APFD metric used to estimate the effectiveness of different
prioritization techniques, is known to have limitations. In very special cases it may rank equally test case orderings that are not equivalent from the human point of view. However, the APFD metric has been widely used in related research and it represents a reasonable approximation of the desired measure.

- Conclusion validity threats, concerning the relationship between the treatment and the outcome. Since a case study was used, no statistical test could be applied to the results. Hence, the conclusions are supported by qualitative analysis, not by statistical tests, as always happening with case studies.

- External validity threats, concerning the generalization of the findings. Since only one data point is available, generalization of the results obtained in a case study is always hard. In addition to that, generalization to different kinds of subjects (e.g., professional programmers working in an industrial environment) and to different kinds of software (e.g., industrial systems) is difficult. The subjects had a profile which makes them not so different from professional programmers, so it is expected that some generality of the results hold.

### 6.5 Discussion

The first experiment, Section 6.4, indicates that the scenario generation algorithm generates all the user preferred scenarios for testing. Prioritization of scenarios using customer priority as well as priority obtained from the primitives of use case and activity diagrams aid in early fault detection. In case of scenario selection, where a subset of scenarios are selected for testing, the results show that using distance measures is an effective way of test selection. Besides, the technique
used for picking one of the two selected scenarios is effective when compared to random. Clustering scenarios is another approach to selecting scenarios. The results of the clustering approach indicate that grouping scenarios aids in selecting a representative set for testing.

The results of the second case study, showed that prioritization of scenarios based on primitives is effective for early detection of defects. Also, the technique can be used with customer inputs on priority to test based on needs of the customer. In case of selection, the techniques aid in selecting a representative set of scenarios. The parameter used to pick one of two similar scenarios aids the defect detection rate and hence, one of the techniques can be used based on information available.

In clustering based selection, the results were compared to random selection. Through the experiment, it was understood that where defects are distributed evenly in the application, clustering and selecting scenarios from each cluster for testing is similar to random testing. Hence, the results validate the performance. Clustering scenarios and testing from each cluster is expected to be productive when the defects will be from the same cluster. Such a case is expected during regression testing where the occurrence of a defect in a scenario indicates the possible occurrence of defects in similar scenarios. Here, related scenarios have to be tested. Another possibility is when the defects are localized to a requirement e.g. defects due to wrong design. In such a case, defects will be centered around that requirement, and hence a particular cluster. The results of testing based on cluster validate the same. Hence, there is need to study measures for clustering further to improve on the results obtained.

The advantage of the prioritization and selection techniques is that other than obtaining customer priority for prioritization, the techniques require no human intervention and is completely automated. Also, the technique can be used in
tandem with other techniques available in literature for prioritization and selection of scenarios to aid in effective testing of software.

6.6 Summary

This chapter presented the results of two case studies that have been conducted to study the proposed scenario generation, prioritization and selection techniques. The results obtained from the experiments indicate that the techniques aid in automated testing. The following observations were made:

- Prioritization of scenarios using customer priority as well as priority obtained from the primitives of use case and activity diagrams aid in early fault detection.

- In case of scenario selection, distance measure is effective to select a subset of scenarios for testing.

- The advantage of prioritization and selection techniques presented in this work require minimal human intervention and is completely automated.

- Besides, the techniques can be used along with other techniques for prioritization and selection.