AUTOMATIC TEST PATH GENERATION BASED ON UML ACTIVITY DIAGRAM

Rosziati Ibrahim1, Tan Shi Pie2
1,2 Department of Software Engineering, Faculty of Information Technology and Multimedia, Universiti Tun Hussein Onn Malaysia (UTHM), Batu Pahat, Johor, Malaysia
Email: rosziati@uthm.edu.my1, sptan86@yahoo.com2

ABSTRACT

Unified Modeling Language (UML) is a standard language for modeling of a system. UML is used to visually specify the structure and behavior of a system. The system requirements are captured and then converted into UML specification. UML specification uses a set of rules and notations, and diagrams to specify the system requirements. One of the diagrams that is used in specifying the system requirements is an activity diagram. In this paper, we present a tool for generating test path from an activity diagram. Deriving manually all test scenarios from an activity diagram is a very time consuming. Therefore, an automation of test path generation based on UML activity diagram is developed to generate possible test paths automatically from an activity diagram. From the generated test paths, the test plan can be produced. With the test plan, the software developers would be able to compare the activity flows in actual system developed with the generated paths to ensure the system behaved properly. The tool can be used to check the consistency between the program execution traces and behaviour of activity diagram as defined in analysis phase.

Keywords: Software Engineering, Unified Modeling Language (UML), Activity Diagram, Test Path, Test Plan

1 INTRODUCTION

Testing software based on its defined design specifications is necessary before releasing a system. This is to ensure that the software meets its requirements and user’s needs, and also to discover bugs and error that exists in the software that developer may unaware. Software testing is one of the “verification and validation” (V&V) software practices [12]. Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Verification activities include testing and review. On the other hand, validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. At the end of the development, validation activities are used to evaluate whether the features that have been built into the software satisfy the customers requirements and exactly meet their needs.

This system will use UML activity diagram as design specification, and present an automatic test path generation approach. This approach first provides a platform for user to input an activity diagram. After scanning the complete activity diagram, a set of test paths will be generated. Then a test plan with possible test paths will be produced based on the user input. By comparing the generated results with the corresponding program execution traces, user can check the consistency between the actual program execution and the behavior of UML activity diagram. Thus the coverage of the system will be increased and the correctness of activity diagram will be determined. This system will generate test paths and test plan automatically, thus it can reduce and cut down the high cost of manual testing. On the other hand, time consumed is shortened and the results generated will be more reliable.

The rest of the paper is organized as follows. Section 2 describes the UML specification in details and Section 3 reviews the related work. Section 4 discusses the implementation of the tool in details. Finally, we conclude our paper in Section 5.

2 UML SPECIFICATION

Unified Modeling Language (UML) is a standard set of diagramming techniques that provides a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation. The current version of UML 2.0 has introduced 13 diagrams for UML specification [9]. UML diagrams are broken
into two major groupings: one for modeling structure of a system and one for modeling behavior. Activity diagram is an example of behavioral diagram that illustrates business workflows independent of classes, the flow of activities in a use case, or detailed design of a method. Activity diagram is capable of successfully specifying an entire set of use-case scenarios in a single diagram. In addition, it is potentially a rich source of test related information in both business and software-based models. Deriving all test scenarios from an activity diagram is a very time consuming activity when performed manually. Therefore, an automation of test path generation based on activity diagram is developed to generate possible test paths automatically from an activity diagram. Finally, a test plan with all those test paths will be produced.

A test path is defined as system’s activity flow or process flow from the start to the end. While a test plan is a document describing the scope, approach, resources, and schedule of intended test activities.

Functional models describe business processes and the interaction of an information system with its environment. In object-oriented systems development, two types of models are used to describe the functionalities of an information system: activity diagrams and use-case diagrams [3]. Activity diagram is a diagram that are commonly used to model business processes, basic control and data flow in software system and they require little technical expertise to develop and understand [1]. UML activity diagrams are developed using elements that are divided into two groups: nodes and edges. Nodes are defined as action nodes, object nodes and control nodes; while edges are defined as the transitions that represent control flow between nodes.

3 RELATED WORK

Software testing is an important process in software development life cycle (SDLC). The purposes of software testing are to reveal errors in a program and to make sure that the programs meets its users requirements and needs. Software organizations spend considerable portion of their budget in testing process [10]. According to Boyapati et al. [2], manual software testing, in general, and test data generation, in particular, are labor-intensive and time-consuming processes. To cut down cost of manual testing and to increase reliability of it, researchers and practitioners have tried to automate it [10]. Automated testing can significantly reduce the cost of software development and maintenance.

In system testing, test cases are manually generated from use cases. Heumann [5] discusses on how to manually generate test cases from use cases using the basic flow of events and alternate flows of events. The scenarios are first created from combination of the basic flow and alternate flows. These scenarios are then used as the basis for creating test cases. However, deriving test cases manually consumes more time and requires more tedious job. Therefore, many researchers study ways to automate the process of generating test cases, for example, Wee et al. [13], Hui et al. [6], Gutierrez et al. [4] and Boyapati et al. [2]. Wee et al. [13] study the automation of test cases using the behaviour of object-oriented classes, where the test cases are automatically generated from the closed specifications of classes. They propose a scheme that combines the setup process, test execution and test validation into a single test program for testing the behaviour of object-oriented classes. The test program is generated automatically using the test cases and closed specifications of the classes.

Hui et al. [6] study the validation of input and propose an approach for automated verification and test case generation of input validation from source code. Their work is concentrated on program source codes to get the input validation for generating the test cases. Gutierrez et al. [4], on the other hand, propose an approach for automatic generation of test cases from use cases for web applications using activity diagrams representing the behavioural model of the system’s requirements. Boyapati et al. [2] study the specification-based testing and generate optimal test cases based on the specifications of Java predicates.

Modeling languages are used to get the specification and generate test cases. UML is the most widely used language such as state-chart diagrams, use-case diagrams, sequence diagrams. Prasanna et al. [10] describe UML can be visualized as four meta-model architecture with three logical sub packages: Foundation, Behavioral elements and Model management. UML provides capability to explore static and dynamic behavior and physical deployment of a system. The possibility of using UML for software testing was addressed by Williams [14]. According to Lieberman [7], software testing team can use UML diagrams to directly aid in the creation of the test plan and test cases. Since UML has been accepted as the standard notation by the Object Management group (OMG), almost all object-oriented development projects today utilized activity diagrams and use

ISSN 2085-1944
cases to document and organize the requirements that are obtained during the analysis phase [8].

For our approach, instead of testing the software at the later stage, this research will use activity diagram to automatically generate the test paths which then later produce the test plan which consists of test cases. The test cases will then be analyzed in order to validate the requirements of the system. This will give more strength to our approach of using activity diagram for system specification, where ambiguity of the system’s requirements will be reduced.

4 THE TOOL - TPG

The tool, which we call TPG (Test Path Generator), can be used to develop the activity diagram and then generate the test path based on the given activity diagram. Figure 1 shows the components of the tool.

<table>
<thead>
<tr>
<th>STAGE 1 (EDITOR)</th>
<th>STAGE 2 (ENGINE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workspace</td>
<td>Activity Diagram</td>
</tr>
<tr>
<td></td>
<td>Test Path</td>
</tr>
</tbody>
</table>

Figure 1: The Components of TPG

From Figure 1, the tool allows a user to develop the activity diagram of any system in the workspace provided. The workspace is used as a place for a user to provide the activity flows of system requirements by means of the activity diagrams. In the workspace, a ToolBox is used to create, edit, display and print the diagrams. The ToolBox consists of standard symbols and notations for activity diagram as described in Table 1.

Table 1: Symbols in ToolBox

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>●</td>
<td>Initial/Start Node</td>
<td>Is used to start the actions or activities</td>
</tr>
<tr>
<td>●</td>
<td>Final Node</td>
<td>Is used to stop all control flows</td>
</tr>
<tr>
<td></td>
<td>Action Node</td>
<td>Is used for behavior in actions</td>
</tr>
<tr>
<td>□</td>
<td>Control/Decision Node</td>
<td>Is used to represent a test condition to ensure the control flow goes down one path</td>
</tr>
</tbody>
</table>

Figure 2 shows the snapshot of the interface of the tool. The tool has two main interfaces which are the workspace for a user to input the activity diagram and the output which shows the test path generated based on the input.

Figure 2: The Interface Design of TPGEN

From Figure 2, the Workspace will allow a user of the tool to develop the activity diagram according to any system requirements. The activity diagrams can also be saved for later use and printed for hardcopy. Figure 3 shows the system requirements of the tool using a use-case diagram. A use-case diagram is used graphically to describe the functionalities of the tool as the dialog between a user and the tool itself.

Figure 3: The Use-Case Diagram of TPG

From Figure 3, the user can draw and open the activity diagram, open new file for a new
activity diagram, request to generate the test paths and test plan, save the test plan and print the activity diagram. Figure 4 shows an example when an activity diagram is drawn in the workspace.

Once the activity diagram is correct, a user can generate the test path by clicking the generate test path button. The engine of the tool will parse the activity diagram to get the start node ID and then assign the text “Start” to the variable “test_path” which stores the test path of the overall activity diagram. After that a recursive function will be called to get the test path continuously following the connections between nodes to get all the possible test paths. The function will loop the next test path once it meets the end node of that particular path. The function will come to an end once all the paths are parsed through. Figure 6 shows the generated test path from the activity diagram in Figure 4.

From Figure 6, in order to come up with complete paths from an activity diagram, each node in the activity diagram has to be connected properly by edges so that each element can be located and connected to produce a complete path. The process of generating the test paths will begin by tracing the activity diagram’s associated activity graph where the initial node is located within the activity diagram. Then the edge is detected with the following node. The process is finished when the final node is detected. When the final node is detected, the test path is then generated.

The next process is that the test plan can be produced based on the generated test path. The test plan will be saved and opened in Microsoft Office Excel file format with extension file of “.xls”. Figure 7 shows the test plan produced from the generated test path.

The tool is able to check the correctness of the activity diagram against the set of rules and notations imposed by the UML specification which are stored inside the tool database for syntax checker of activity diagram. The checker will first check whether the start and final nodes are existed because the basic of an activity diagram is the start node, final node and connection between nodes. Besides that, an activity diagram should not have more than one start node. The checker will also check the text of the connections where the source of the connection is a decision node. If the connection texts of a decision node have the same text, then the activity diagram is considered as incorrect and error messages will be displayed. All error messages will be appended and displayed in the dialog box. Figure 5 shows the error messages when the activity diagram is incorrect.
5 CONCLUSION

TPG is a tool that is able to generate the test paths automatically according to the activity diagram. Then the test plan can be produced from the test paths. The test plan can be used as a checklist for software developers to validate that the system meets its requirements. The test plan can also help to check the consistency between the program execution traces and behavior of activity diagram as defined in analysis phase. The purpose of TPG is to reduce the cost of testing the system.

We have also described in this paper our method on how to check the syntax of an activity diagram according to the set of rules and notations described by UML specification. The syntax checker is embedded inside of the tool. It is acted just like a compiler to check the syntax of the programming language. Our syntax checker, on the other hand, check the syntax of the UML activity diagram.

REFERENCES


