International Journal of Hybrid Information Technology
Vol.9, No.4 (2016), pp. 427-436
http://dx.doi.org/10.14257/ijhit.2016.9.4.37

Innovative Technique to Produce Test Codes from Predefined Design Information

Shamil Al-Ameen[1], Roua Al-Taie[2,*], Zohair Al-Ameen[2]

[1]Department of Computer Science, Cihan University, Erbil, Kurdistan Region – Iraq

[2]Department of Information Technology, Lebanese French University, Erbil, Kurdistan Region – Iraq

** Corresponding Author: rouabasil@lfu.edu.krd*

Abstract

During any system's development life cycle, software testing is considered an important and key phase. Even so, the testing phase is commonly the first phase to be removed when time and resources constraints are faced due to its high consumption of recourses. As a result, developers waste their time and effort on developing software that is either filled with bugs or mismatches user specifications. The excessive work and time spent to program and execute a test case takes approximately 70% of a project's resources. The goal of this work is to produce a practical implementation of the extreme programming as a software development methodology that focuses on generating test cases from the design information before initiating the coding phase. Furthermore, a competent testing tool is built to employ the available information in the related design diagrams, Class and State diagram, wherein it automatically generates transition sequences to test any software system. The proposed tool can reduce the time, cost, and human resources used to test or redesign the system when it mismatches user specifications and requirements.

Keywords: *Code generation, Design information, GEP, State diagram, Test case*

1. Introduction

Unified Modeling Language (UML) is an Object Management Group (OMG) standard for object oriented modeling that is widely used in software system modeling [12]. UML has become the standard notation of object design methodologies [9]. Software testing is a key stage in any system development yet it is also the most expensive as it consumes more than half of the allocated resources for the entire software development process [12]. Nonetheless, the testing phase is the first phase that can be eliminated if the developers ran out of time or resources of any kind [2, 3, 4]. This causes developers to waste their time and effort on developing software that doesn't match the user specifications or building a faulty system. As an outline, the main stages of a software development life cycle are: requirement gathering and analysis, design, coding [10]. Conducting tests at each of the stated stages can guarantee to build a high quality system that matches the user specifications. However, such an option is feasible only when the system is allocated to unlimited resources and delivery time. According to the testing results at each stage, a decision must be made to redo the job or continue to the next stage. This will usually cost a lot of time, money and effort which may not be available for every project. As a solution to the aforementioned dilemma, an Extreme Programming (XP) methodology can be used to achieve better results. XP is a development methodology that can be used when a developer creates a set of testing units from the available design information before starting the process of code generation. Therefore, test cases can be ready when a system code is generated and thus, a high quality project can be delivered in time while saving lots of allocated resources [11]. Test case design and executions are time consuming and labor intensive. Likewise, a manual test design is very time consuming, error-prone, and may not be as good as the required quality [12]. Therefore, automating the generation of test case is an important yet challenging task. In this article, a completely automated testing tool is proposed, in which it operates in three stages. The first stage deals with converting the XML Metadata Interchange (Xmi) into Meta-Xmi document, while the second stage deals with using an improved method of Gene Expression Programming (GEP) to automatically generate a high quality transition sequences that can be used in the testing stage. The third stage deals with building testing programs in an automatic way for the generated transition sequences. As a closure, this article is arranged in the following fashion: In Section 2, the UML is explained sufficiently. In Section 3, the XP is clarified briefly. In Section 4, the process of test case generation is elucidated concisely. In Section 5, the required information about code generation is provided. In Section 6, the proposed framework is discussed in details. In Section 7, a case study about a coffee vending machine is given. In Section 8, a closure of key observations is delivered.

2. Unified Modeling Language

UML is an object oriented modeling language that implements many techniques for specification, visualization and documentation [1]. There exist two types of diagrams which are used in this work:

1. The state diagram: it describes the reaction of an object to the arrival of an event. Each reaction may be a sequence of actions, possibly accompanied by a transition from one named state to another. An event represents the reception of a signal or the effect of an operation call. An action represents the send of a signal or the call of an operation [1]. This is why state diagrams are always chosen to perform unit testing operations [10]. Such diagrams are used to define the state of a system after applying a series of actions to it.
2. The class diagram: the class consists of a set of objects that share the same attributes, operations, relationships and semantics. In a class diagram, the classes that describe the system are stated with their attributes and operations. Each class is drawn as a rectangle with three compartments: the top compartment holds the class name, the middle holds a list of attributes; the bottom holds a list of operations [1]. This type of diagrams is used to define the overall structure of the design presented as the class name, operations and attributes.

3. Extreme Programming

XP is a relatively new software development process that enables developers to rapidly create high-quality code [11]. The purpose of XP development methodology is to create high quality products in short time frames. XP is much better than classical software processes because the classical approaches are usually time consuming and this may cause delays in the scheduled deadline [10]. The XP model relies heavily on units and test acceptance of modules, because the primary difference of an XP methodology is that it focuses on testing. In XP, unit tests must be created first. Then, the code is created to pass the tests [11].

4. Test Case Generation

A product specification and its source code are usually needed when building test cases for a software product. A specification defines the product’s input, output parameters and functions. To put the principle of XP into practice, developers use extreme testing by generating unit tests [11]. Unit testing is the primary testing approach used in extreme testing and has two simple rules: All software product modules must have unit tests before the coding begins and pass unit tests before being released into production [10]. The big difference between unit testing and XP is that unit tests must be defined and created before coding the module [11].

5. Code Generation

Code generation means creating a code for design purposes. Using the information extracted from the design models of the product, the developer creates a programming code that can perform certain tasks as specified by the design model. To ensure a high quality product, the code generation stage must perform exactly what the user required while ensuring the production of an error-free program. The aforementioned can be achieved by teaching the computer to generate the whole program skeleton. This ensures an error-free program with a fast implementation time. The code generation process is important in keeping consistency between a model (the design diagram) and its implementation (the code of the design). It also reduces the time of writing source code [7]. Generating the testing data alone to improve the systems quality by reducing its generation time is not enough. Such data must be molded by a testing program in a way that it becomes easy for use directly by developers after completing the code of the system. Developers should not waste their time on writing the testing program for this testing data. Therefore, an automatic generation of the testing program is required to prevent time wastage and produce high quality projects which are built and tested in the specified time line.

6. Proposed Framework

In this article, a new framework is proposed to provide a competent testing program from the information of the design specification presented by the used UML. As illustrated Figure. 1, the work of the proposed tool can be divided into three distinct stages:

1. The first stage involves inputting the UML state diagram in the form of Xmi files. Then, the unstructured Xmi information are transformed to a structured form by eliminating the unnecessary modeling information and creating the Xmi meta-model using a special parser which is designed specifically as a part of the testing tool. This is done by creating an Xmi data structure that contains only the required information about the design's transitions.

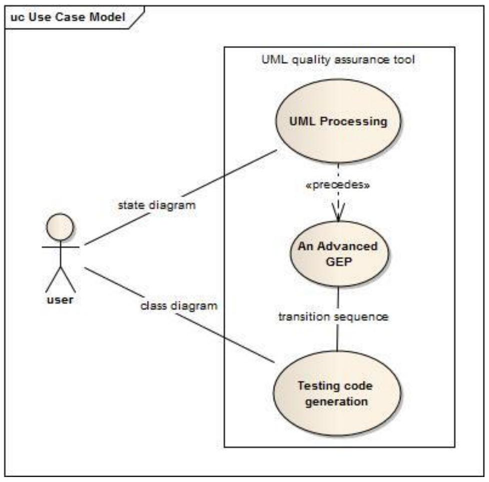

Figure 1. The general use case diagram of the proposed framework

2. The second stage receipts the structured Xmi representation to automate the generation of transition sequences. This is done by utilizing an advanced approach of GEP that is developed earlier. Such approach is employed to produce high quality sequences of transitions and obtain the improved GEP.
3. The third stage receives the produced transition sequences automatically to create testing programs written with Visual C++. The aforesaid programs are built to generate the main class, define its attributes and operations. Each of these operations is used to do a specific function inside the entire software system. Based on the testing sequence that is generated in an earlier stage, a set of operation calls is achieved to test the system.

After generating the required system source code, the results of these tests are compared with specific results which are defined by user specifications requirements and modeled in the state diagram. Identical results can indicate a successful system modeling. If not, the design must be modified to accommodate the user requirements and the same process is repeated. The advantages of using such framework are:

1. Discovering and correcting errors of the development process at early stages.
2. Preparing high quality tests to avoid time wastage when developing code testing cases.
3. Generating transition sequences automatically to use them for testing purposes while ensuring high quality test cases.

6.1. Processing the UML State Diagram

As shown in Figure. 2, the process starts with the creation of a UML state diagram which can be described as a graph-like diagram that defines a system in a matter of its different states. A state diagram should contain a number of transitions which fire some triggers that causes the system to change its state. That’s why the state diagram is the best diagram to be used when system testings’ are considered [3, 6].

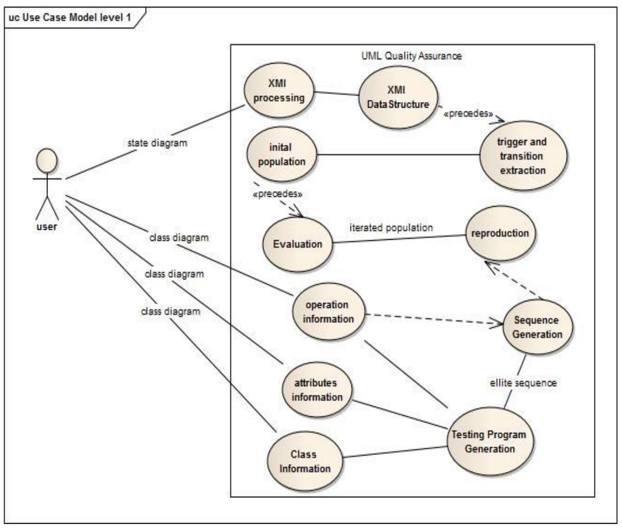

Figure 2. The detailed use case diagram of this study

A UML tool starts when a customer selects the state diagram. Then, the tool transforms the diagram into its intermediate representation as an Xmi document. Afterwards, the latter document is used to convert the design information from one diagram to another and from one tool to another. Dealing with the Xmi document is somewhat difficult considering that it contains a lot of unnecessary information about the layout of the design. The Xmi document is also known for its variety of versions, in that it has a lot of versions, starting from 1.0 to 2.1. It is important to mention that Xmi formats do not have a common reliable structure, which leaves no alternative but to analyze them separately [8]. This is achieved by building a special parser just to discover the correct information about the attained transitions by the system and the triggers. The parser eliminates all the unnecessary information in Xmi documents and transforms the Xmi document into a data structure which contains the transition information.

6.2. Generating the Testing Program

As mentioned earlier, the output of the improved GEP can provide a number of transition sequences. These sequences are transformed into a complete program which is written in C++. The program skeleton is divided into two important parts:

1. The definitions of system libraries used in the code and the design class coupled with the implementation of the latter. The aforesaid can create both classes of constructor and de-constructor automatically. Then, the execution of the class can define system attributes, their visibility, type and define the operations (transitions) that are stated in the diagram along with their visibility, return type and parameters.
2. The definition of the main program, this part takes each transition in the transition sequence and finds its equivalent operation which is responsible for this transition in the class implementation. Then, it defines a parameter to call the operation if it has a return type other than void. Therefore, the operation call is achieved by an automatic run time method invocation.

Using this way, the quality of the testing sequence is assured by using the improved GEP principle, which will improve the quality of the testing program that is generated automatically with the testing tool. As a result, an error-free code is generated.

6.3. Case Study: the coffee vending machine

In this section, a case study regarding the proposed framework is provided, in which it describes a coffee vending machine from a software engineering perspective. The tool takes place in the order illustrated by the sequence diagram shown in Figure. 3. It starts by receiving a UML state diagram and then transforming the essential diagram information into an Xmi representation. The aforesaid is achieved to extract the information from the design diagram. Figure 4 shows the employed case study. After the extraction process, all the unnecessary Xmi information is removed. The removed material includes irrelevant information about the layout of the design, font type, size, color, and many more. As a result, only the vital information of the project which is called meta-model is utilized. The remaining information is transformed into a more structured representation by transforming them into a data structure that describes each transaction's information in the project. The resulting data structure must include all the needed information to evaluate each pair of transactions. An example of the data structure of one for the transactions is displayed in Table 1.

Table 1. A sample data structure for a transaction

xmi.id	UMLTransition.18
Name	ti1
stateMachine	UMLStateMachine.3
Source	UMLPseudostate.17
Target	UMLCompositeState.5
State	Off

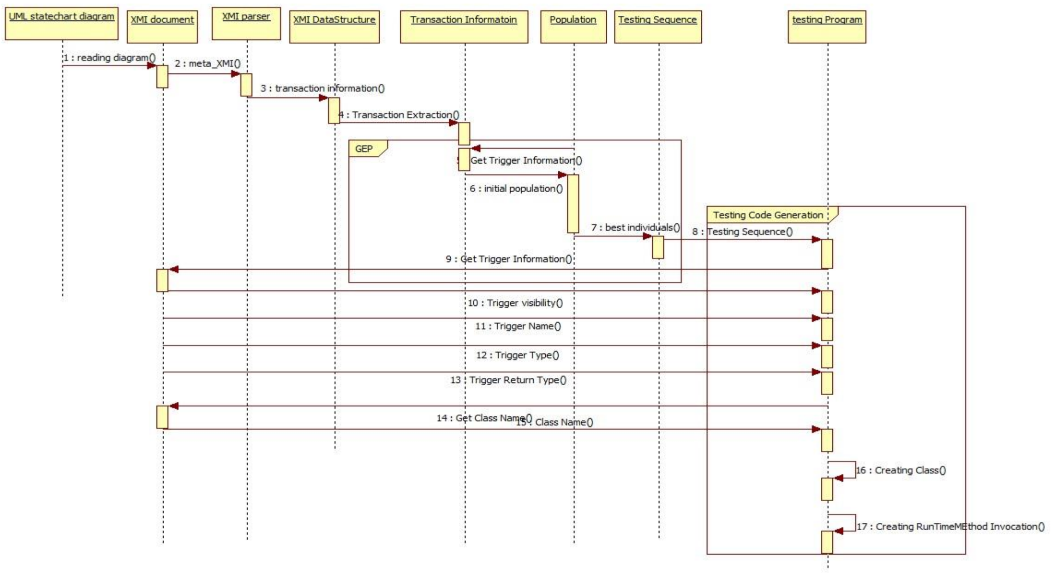

Figure 3. The sequence diagram of the proposed framework

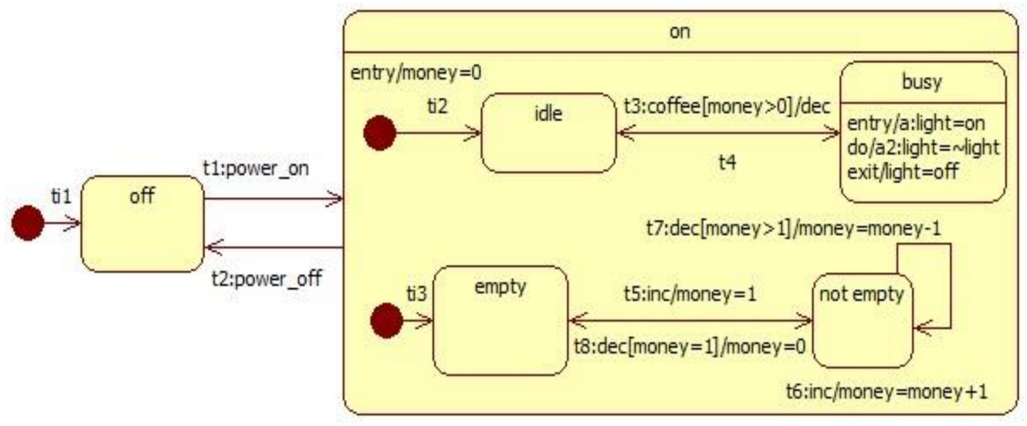

Figure 4. The state diagram of the coffee vending machine system

After a structuring process ends, the improved GEP starts to find a set of transactions which cover the entire diagram. The generated test sequences are then transformed into a complete testing program that runs immediately after completing the code generation phase. As shown in Figure. 4, the coffee vending machine consists of many states, namely: on, off, empty, not empty, idle, and busy. Each transition from a state to another is caused by a transaction, the transactions in this example are: power_on, power_off, inc, dec, and coffee. Initial triggers are also shown in the figure and they are: ti1, ti2, and ti3. After extracting the necessary information from this design, an example of the transition sequences is:

```
ti1, ti2, power_on , inc , coffee, inc, dec, power_off
```

These sequences are then used to generate a testing code that can achieve each one of them individually. An example of such testing code is:

```
//============================
//          File Name = CoffeeMachine .c++
//          Author Name = Roua Basil Al-Taie
//          TestCase Program That Provide A High Quality Test
//          Cases from the Design of the System to Be Build
//          This Program Is Automatically Generated From the UML
//          Version 2.0 Using the Xmi Version 1.3 and With The
//          Help Of Microsoft Visual Studio 2010

using namespace System;
using namespace System::Data;
using namespace System::Text;
using namespace System::IO;
class CoffeeMachine
  {
    public:
      CoffeeMachine();
};
CoffeeMachine::CoffeeMachine()
  {
 public:
int coffee();
public:
char power_on();
public:
void power_off();
public:
bool inc();
public:
void dec();
};
void main()  {
CoffeeMachine CoffeeMachineInstance;
char operation_variable_0;
operation_variable_0= CoffeeMachineInstance::power_on();
bool operation_variable_1;
operation_variable_1= CoffeeMachineInstance::inc();
int operation_variable_2;
operation_variable_2= CoffeeMachineInstance::coffee();
bool operation_variable_3;
operation_variable_3= CoffeeMachineInstance::inc();
bool operation_variable_4;
operation_variable_4= CoffeeMachineInstance::dec();
CoffeeMachineInstance::power_off();
}
//============ END OF CODE =========//
```

7. Conclusion

Although software testing is an important phase in any system's development life cycle it is normally overlooked by developers to save resources. The work presented in this article provided a solution to this problem by generating test cases from the design phase before the coding begins which will save both time and resources. The proposed tool can accept a state UML diagram as an input and produce complete test programs as an output. The improved GEP achieved high quality testing procedures by automatically creating a number of transition sequences. Those sequences can generate complete testing programs which can execute definitions and implementations. In addition, a runtime invocation method is used to call the operations in classes. Finally, it is believed that this work can benefit many software engineering fields when it comes to fast testing phase and minimum resource usage.

Acknowledgments

The authors would like to thank the respectable reviewers for their helpful comments.

References

[1] P. Samuel, R. Mall and A. Bothra, 'Automatic test case generation using unified modeling language (UML) state diagrams', *IET Software*, vol. 2, no. 2,**(2008)**, pp. 79-93.

[2] V. Garousi, 'A Genetic Algorithm-Based Stress Test Requirements Generator Tool and Its Empirical Evaluation', *IEEE Transactions on Software Engineering*, vol. 36, no. 6, (**2010)**, pp. 778-797.

[3] J. Minj and L. Belchanden, 'Path Oriented Test Case Generation for UML State Diagram using Genetic Algorithm', *International Journal of Computer Applications*, vol. 82, no. 7, (**2013)**, pp. 1-4.

[4] R. Swain, V. Panthi, P. Kumar Behera and D. Prasad Mohapatra, 'Automatic Test case Generation From UML State Chart Diagram', International Journal of Computer Applications, vol. 42, no. 7, (**2012)**, pp. 26-36.

[5] N. Ryan and D. Hibler, 'Robust Gene Expression Programming', *Procedia Computer Science*, vol. 6, (**2011)**, pp. 165-170.

[6] N. Pahwa and K. Solanki, 'UML based Test Case Generation Methods: A Review', *International Journal of Computer Applications*, vol. 95, no. 20, (**2014)**, pp. 1-6.

[7] K. Seo and E. Choi, 'Traceability of UML Based Test Artifacts Using XML', *The KIPS Transactions: Part D*, vol. 16, no. 2, (**2009)**, pp. 213-222.

[8] M. Misbhauddin and M. Alshayeb, 'Extending the UML use case metamodel with behavioral information to facilitate model analysis and interchange', *Software & Systems Modeling*, vol. 14, no. 2, **(2013)**, pp. 813-838.

[9] S. Pickin, C. Jard, T. Jeron, J. Jezequel and Y. Le Traon, 'Test Synthesis from UML Models of Distributed Software', *IEEE Transactions on Software Engineering*, vol. 33, no. 4, **(2007)**, pp. 252-269.

[10] R. Pressman and B. Maxim, *Software engineering*. Boston, Mass.: McGraw-Hill, (**2014)**.

[11] G. Myers, C. Sandler and T. Badgett, *The art of software testing*. Hoboken, N.J.: John Wiley & Sons, **(2012)**.

[12] R. Swain, V. Panthi, P. Kumar Behera and D. Prasad Mohapatra, 'Automatic Test case Generation From UML State Chart Diagram', *International Journal of Computer Applications*, vol. 42, no. 7, (**2012**), pp. 26-36.

Authors

Shamil Al-Ameen, was born in 1987. He received his BSc and MSc degrees in Software Engineering from the University of Mosul in 2009 and 2011, respectively. His research interests include software engineering, computer networks and computer security. Currently, he is a full-time lecturer at the Department of Computer Science, Cihan University.

Roua Al-Taie, was born in 1987. She received her BSc and MSc degrees in Software Engineering from the University of Mosul in 2009 and 2011, respectively. Her research interests include software engineering, data structure and quality assurance. Currently, she is a full-time lecturer at the Department of Information Technology, Lebanese French University.

Zohair Al-Ameen, was born in 1985. He received his BSc degree in Computer Science from the University of Mosul in 2008. Then, he received his MSc and PhD degrees in Computer Science from the Technological University of Malaysia in 2011 and 2015, respectively. His research interests include algorithms design, computer forensics and image processing. Currently, he is a full-time lecturer at the Department of Information Technology, Lebanese French University.