2018 5th International Conference on Computational Science/Intelligence and Applied Informatics

Stubs and drivers generator for object-oriented program testing using sequence and class diagrams

Peerawut Luengruengroj
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
Peerawut_lueng@hotmail.com

Taratip Suwannasart
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Bangkok, Thailand
Taratip.s@chula.ac.th

***Abstract*—This paper aims to present a tool named Stubs and Drivers Generation Tool which is a web application for generating stub and driver source code from an UML, sequence diagram and a class diagram. Testers can automate the unit testing with our tool. The tool will read the XML file of a sequence diagram and a class diagram. Next, the tool processes the XML file and create a call graph from the sequence diagram. After a tester selects a class under test and set values of the class under test source code attributes, the tool will create the stub and driver from set attributes. The tester can customize the source code and export the source code file for using in the testing process.**

***Keywords*—Software Testing, Object-Oriented, Sequence Diagram, Stub, Driver**

I. Introduction

Software testing is one of the process in software development life cycle which finds defects in software and assures the software quality [1] so software testing is one of important process in software development. By using V-Model [2] as a software development methodology, unit testing will be planned as of the detailed design is done [2]. In object-oriented software development, unit test cases are calling the method of each class and examine actual results whether they are consistent with expected results [3]. However, the software under test may contain a ton of classes and each method will be called within a class or between classes. It can consume a lot of time for creating a stub and a driver for testing each object. According to analysis and designing phase, the relationship among classes can be described by UML (Unified Modelling Language) diagram named Class Diagram [2] and the method calling can be described by UML diagram called Sequence Diagram [2]. In test case generation for object-oriented software, testers can refer the method calling from the class diagram and sequence diagram and can create a stub or a driver to test the class under test if the methods in that class call or are called by other classes. According to the related works, there are tools which able to create stubs and drivers from UML diagrams [4] or a program workflow [5] but they need an information from source code and there is a tool which can identify the stubs and drivers from UML diagrams only but it cannot generate stubs or drivers automatically [6]. The objective of this paper is to propose a tool for generating stubs and drivers to test a class of object-oriented software.

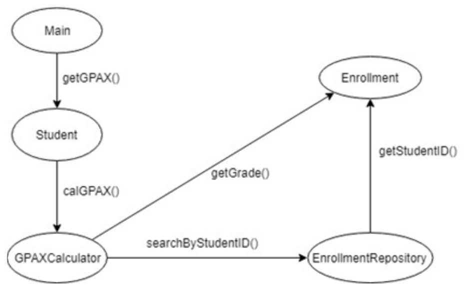


Fig. 1. A graphical example of a call graph.

We use a sequence diagram and a class diagram for creating stubs and drivers. The remainder of this paper is organized as follows. Section 2 provide backgrounds and works related to this paper while section 3 shows the structure of the tool. The next section shows the implemented tool and the last section is a conclusion.

II. Background and related works

A. Stub

A stub is a piece of throwaway code that emulates a called unit [1]. A Stub is usually a hard code or a simple code which does a basic operation of a class and is used in top-down integration testing to be called by the module under test.

B. Driver

A driver is a module that contains wired-in test inputs, calls the module being tested and displays the outputs or compares the actual output with the expected output [3]. A driver is usually used in bottom-up integration testing.

C. Call Graph

A call graph is a kind of control flow graph which shows a relationship between subroutine inside the program [7]. Fig. 1 shows the graphical example of call graph generated from sequence diagram. Each node can refer to the process and each edge can refer to the process calling. The call graph is directed graph. (Each edge must have direction)

978-1-5386-7875-6/18/$31.00 ©2018 IEEE
DOI 10.1109/CSII.2018.00013

D. An Automated Testing Tool for Java Application Using Symbolic Execution based Test Case Generation [5]

This research presents a concept of creating stubs, drivers, and test cases for testing the source code written in Java using symbolic execution technique for analyzing the program flow. This research presents stubs, drivers, and test cases creation method in three steps.

1) Stub-target Method Decision: First, find and analyze the signature of methods which are not a method from java library and other classes.

2) Driver/Stub Definition Generation: This step changes the signatures which are collected from the first step to XML structure.

3) Driver/Stub Code Transformation: This step generates Java source code from the XML from the previous step and tests them by Junit.

According to the research, it is possible to create stubs and drivers from the program flow but it still needs the source code to determine the program flow so stubs and drivers cannot be created until the program source code has finished.

E. A catalogue of stub and driver patterns to support integration testing of aspect-oriented programs [6]

This research presents the concept of stub and driver pattern to test an aspect-oriented program by classifying the testing scenario. The program under the test of this research is an object-oriented program but designed by using aspect-oriented approach.

This research presents three types of stub and driver patterns.

1) Concrete Stub: This pattern is for creating stubs for abstract methods of abstract classes.

2) Operational Driver: This pattern is for the driver which calls a method indirectly.

3) Stub Aggregator: This pattern is for creating a stub for testing an inner class.

According to the research, this research only provides the stub and driver pattern but does not provide the stub and driver generating method.

F. An Automated Test Suite Regeneration Technique Ensuring State Model Coverage Using UML Diagram and Source Syntax [4]

This research presents the concept of generating the new test suite from original test suite using the information from UML diagram and original source code for improving the test suite coverage. The software of this research can compare the original source code to UML diagram in XML format for analyzing the coverage of original test suite and use the syntax from the test suite as the source code template for generating new test suite from original test suite to improve the coverage.

This research aims to generate new test suite from existing test suite by using information from UML diagram to improve original test suite. However, it is necessary to create the original test suite manually before using the tool from this research.

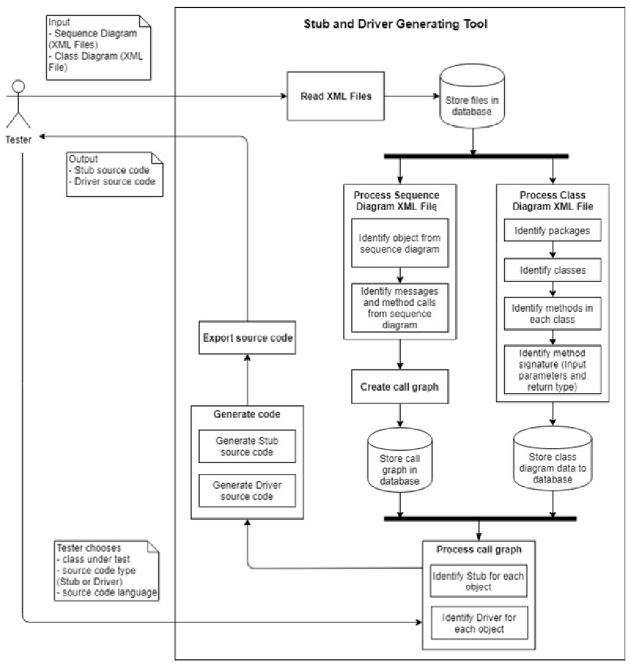

Stub and Driver Generating Tool
Input
- Sequence Diagram (XML Files)
- Class Diagram (XML File)
Tester
Output
- Stub source code
- Driver source code
Read XML Files
Store files in database
Process Sequence Diagram XML File
Identify object from sequence diagram
Identify messages and method calls from sequence diagram
Process Class Diagram XML File
Identify packages
Identify classes
Identify methods in each class
Identify method signature (Input parameters and return type)
Export source code
Create call graph
Generate code
Generate Stub source code
Generate Driver source code
Store call graph in database
Store class diagram data to database
Tester chooses
- class under test
- source code type (Stub or Driver)
- source code language
Process call graph
Identify Stub for each object
Identify Driver for each object

Fig. 2. The Structure of Stubs and Drivers Generation Tool.

III. Stubs and Drivers Generation Tool

This paper we proposed a tool for generating stubs and drivers for object-oriented programs. Our proposed tool has its structure as shown in Fig. 2. The tool can import UML sequence and class diagrams as XML files. Then, the tool creates a call graph to represent method calls between objects in an object-oriented program. Next a tester chooses a test component, which is a class under test, identify if the tester wants the tool to generate stubs or drivers, and specify a language for stubs or drivers to be generated. The tester chooses a source code type whether it is a stub or a driver and then, choose a source code language. The tool will process the call graph to identify which classes need to create stubs or which classes need to create drivers. The tester can edit stubs or drivers generated by our tool and then export the source code from the tool. Fig. 2 demonstrates details of the structure of our tool.

A. Read XML Files

Testers upload a sequence diagram and a class diagram as XML files into the tool. The tool will store the files in the database.

B. Process the sequence diagram XML file

The tool will process the sequence diagram XML file to identify objects, messages, and method calls in the sequence diagram for creating a call graph.

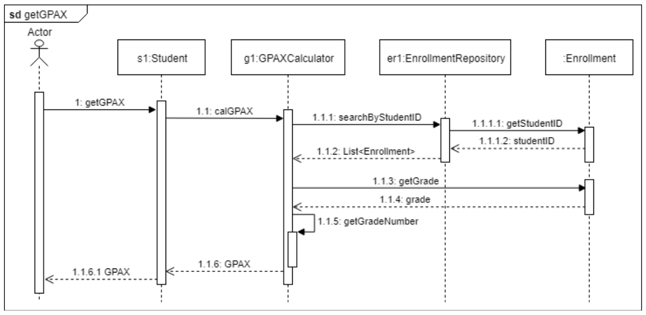


Fig. 3. The example of a sequence diagram.

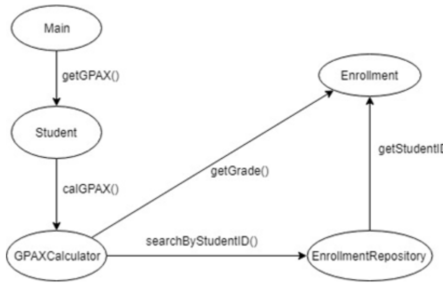


Fig. 4. The example of a call graph.

C. Process the class diagram XML file

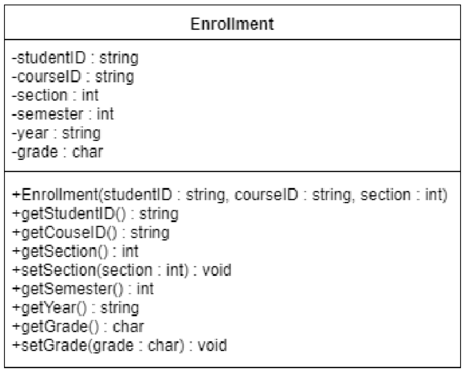
The tool will process the class diagram XML file to identify packages, classes and method signatures. The method signatures are methods input parameters and a return type. After processing the class diagram, the tool will store the packages, the classes, and the method signatures into the database.

D. Creating a call graph

The tool will use objects and messages data from the sequence diagram to create the call graph and save into the database. Fig. 3 shows an example of a sequence diagram while the Fig. 4 is a call graph created from the sequence diagram.

E. Processing the call graph

The tester chooses a class under test from the call graph and specifies stubs or drivers to be generated. In case the tester specifies the tool to generate stubs, the tool will analyze the call graph whether the class under test calls methods of other classes. Then the tool will generate stubs but if the class under test do not call any methods of other classes, the tool will not generate anything except a warning message. In case the tester specifies the tool to generate drivers, the tool will analyze the call graph whether the class under test is called by methods of other classes. Then the tool will generate drivers but if the class under test is not called by any methods of other classes, the tool will not generate anything except a warning message. From Fig. 4, if the tester chooses GPAXCalculator class as a class under test, and specifies to create stubs, the tool will analyze that it needs to create a stub for Enrollment class and


Enrollment

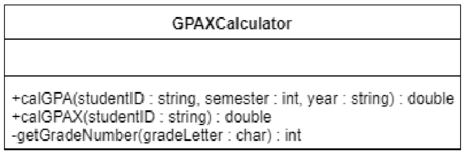
-studentID : string
-courseID : string
-section : int
-semester : int
-year : string
-grade : char

+Enrollment(studentID : string, courseID : string, section : int)
+getStudentID() : string
+getCouseID() : string
+getSection() : int
+setSection(section : int) : void
+getSemester() : int
+getYear() : string
+getGrade() : char
+setGrade(grade : char) : void

Fig. 5. The Enrollment class.

```
public class EnrollmentStub {
    public char getGrade(){
        return ' ';
    }
}
```

Fig. 6. A stub to be called by GPAXCalculator class.


GPAXCalculator

+calGPA(studentID : string, semester : int, year : string) : double
+calGPAX(studentID : string) : double
-getGradeNumber(gradeLetter : char) : int

Fig. 7. The GPAXCalculator class.

a stub for EnrollmentRepository class. If the tester chooses to create drivers, the tool will analyze that it needs to create a driver for Student class.

F. Generating Source Code

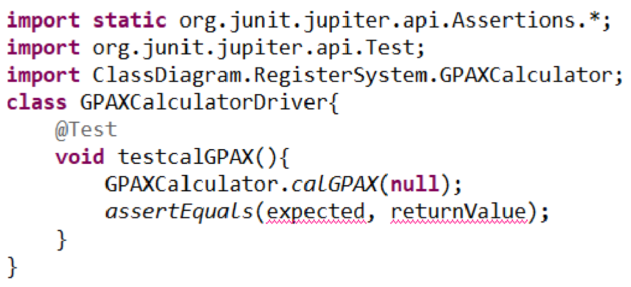
The tool will create source code depending on source code type (Stub or Driver) and a source code language. There are two cases of generating source code.

1) Create stubs: The tool will create a stub source code based on a specified source code language. The source code contains a class with methods generated based on method signatures from the class diagram. Fig. 5 shows the Enrollment class with attributes and signatures. Fig. 6 illustrates the source code of a stub to be called by GPAXCalcultor.

2) Create driver: The tool will create a driver source code based on a specified source code language. In this example we choose Java as the source code language. The source code contains a class with methods generated based on method signatures from the class diagram. Fig. 7 shows the GPAXCalculator class with attributes and signatures. Fig. 8 illustrates the source code of driver calling the GPAXCalculator class.

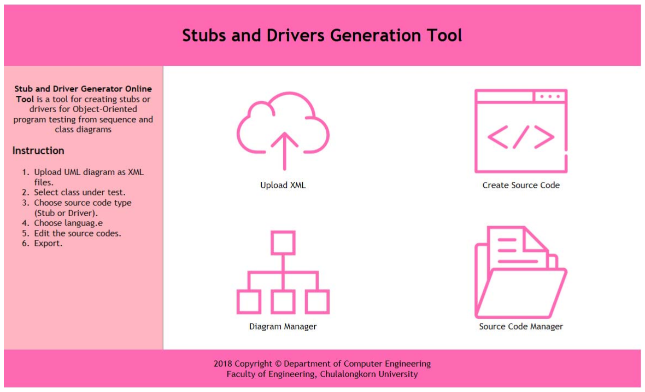
IV. Result

Our tool is implemented in Javascript and PHP as a web application. Fig. 9 shows the main page of the tool. A tester



```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import ClassDiagram.RegisterSystem.GPAXCalculator;
class GPAXCalculatorDriver{
    @Test
    void testcalGPAX(){
        GPAXCalculator.calGPAX(null);
        assertEquals(expected, returnValue);
    }
}
```

Fig. 8. The driver for calling the GPAXCalculator class.



Stub and Driver Generator Online Tool is a tool for creating stubs or drivers for Object-Oriented program testing from sequence and class diagrams

Instruction

1. Upload UML diagram as XML files.
2. Select class under test.
3. Choose source code type (Stub or Driver).
4. Choose languag.e
5. Edit the source codes.
6. Export.

2018 Copyright © Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University

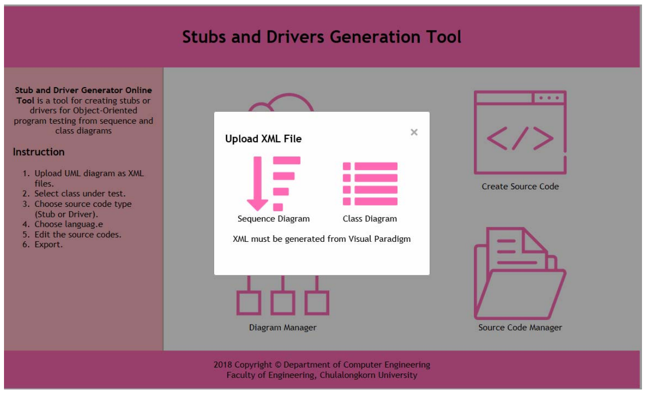
Fig. 9. The main page of the tool.

can upload XML files by pressing Upload XML button and create a source code file by pressing Create Source Code button. The two buttons below are Diagram Manager and Source Code Manager features which we have not developed yet.

The tester can choose a diagram type on uploading XML files. After uploading a file, the tool will process the XML files. In case of sequence diagram uploading, the tool will create call graph, and save the call graph into the database. We use MySQL as a database management system (DBMS). Fig. 10 shows the main page after pressing the Upload XML button. The tester can choose either sequence diagram or class diagram will be uploaded.

After pressing the Create Source Code button, the tester can choose which a sequence diagram and a class diagram that contains a class under test. The tester selects a class under test, sets the filename for a generated stub or driver, selects a source code type (Stub or Driver), and selects a source code language as shown in Fig. 11.

After pressing the Create Code button, the tool will create a source code and source code editor will pop-up as shown in Fig. 12. The tester can edit the source code by adding default values, minimum values, maximum values, or random values and a return type. The tester is also able to export the source code and save the source code to the database for editing in the future.



XML must be generated from Visual Paradigm

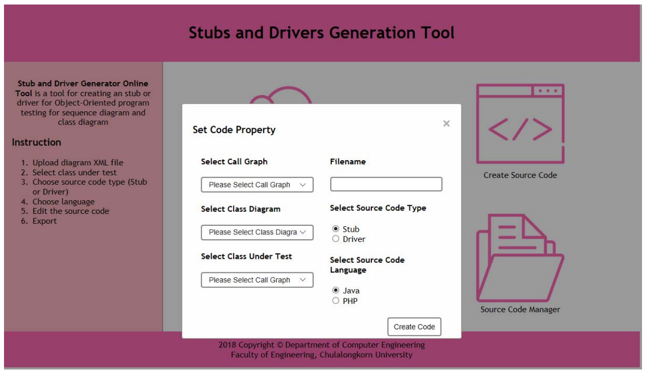
Stub and Driver Generator Online Tool is a tool for creating stubs or drivers for Object-Oriented program testing from sequence and class diagrams

Instruction

1. Upload UML diagram as XML files.
2. Select class under test.
3. Choose source code type (Stub or Driver).
4. Choose languag.e
5. Edit the source codes.
6. Export.

2018 Copyright © Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University

Fig. 10. The page after pressing the Upload XML button.



Stub and Driver Generator Online Tool is a tool for creating an stub or driver for Object-Oriented program testing for sequence diagram and class diagram

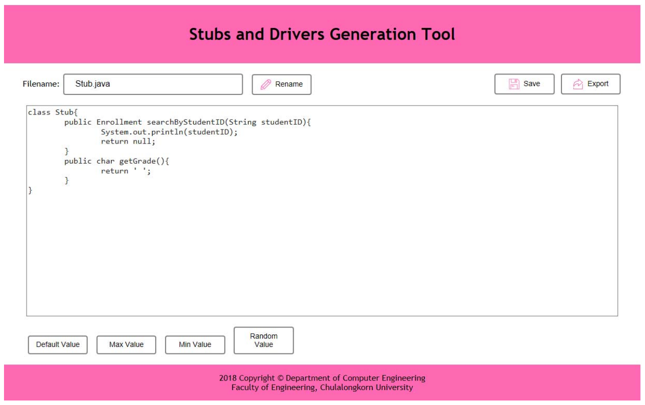
Instruction

1. Upload diagram XML file
2. Select class under test
3. Choose source code type (Stub or Driver)
4. Choose language
5. Edit the source code
6. Export

Select Call Graph: Please Select Call Graph
Filename:
Select Class Diagram: Please Select Class Diagra
Select Source Code Type: [✓ Stub] Driver
Select Class Under Test: Please Select Call Graph
Select Source Code Language: [✓ Java] PHP

2018 Copyright © Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University

Fig. 11. The page after pressing the Create Source Code button.



Filename: Stub.java

```
class Stub{
        public Enrollment searchByStudentID(String studentID){
                System.out.println(studentID);
                return null;
        }
        public char getGrade(){
                return ' ';
        }
}
```

2018 Copyright © Department of Computer Engineering
Faculty of Engineering, Chulalongkorn University

Fig. 12. Source code editor page.

V. Conclusion

This paper we proposed a tool for generating stubs and drivers for object-oriented programs. Our proposed tool is capable for uploading sequence and class diagrams in XML file format, creating a call graph, generating stubs and drivers, editing stubs and drivers source code, and automated specifying signature values. Our tool has not cover class diagrams that contain abstract, inner, and interface class, inheritance, as well as overloading. Our tool has not verified that the sequence diagram is consistent with the class diagram. For the future work we plan to enhance some features such as managing uploaded diagrams and generated source code. We also will handle the limitations mentioned above.

References

[1] P. Jorgensen. Software Testing A Craftsmans Approach. 4 ed. NY: CRC Press, 2014

[2] R. Pressman and B. Maxim. Software Engineering A Practitioners Approach. 8 ed. NY: McGraw-Hill Education, 2015

[3] G. Myers, C. Sandler, and T. Badgett. The Art of Software Testing. 3 ed. NJ: Wiley, 2012

[4] A. Khatun and K. Sakib, ”An automatic test suite regeneration technique ensuring state model coverage using UML diagrams and source syntax., 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), Dhaka, 2016, pp. 88-93.

[5] S. Monpratarnchai, S. Fujiwara, A. Katayama and T. Uehara, ”An Automated Testing Tool for Java Application Using Symbolic Execution Based Test Case Generation.” 2013 20th Asia-Pacific Software Engineering Conference (APSEC), Bangkok, 2013, pp. 93-98.

[6] B. Barbieri De Pontes Cafeo, R. R, R. Braga, and P. Masiero. A catalogue of stub and driver patterns to support integration testing of aspect-oriented programs. Proceedings of the 8th Latin American Conference on Pattern Languages of Programs, ACM, New York, USA, 2010.

[7] U. Khedker, A. Sanyal and B. Karkare. Data Flow Analysis Theory and Practice. FL: CRC Press, 2009