Sudan University of Science and Technology

College of computer science and information technology

Automatic Generation Of System Test Cases For a Systems Under Development

November 2017

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Sudan University of Science and Technology

College of Computer Science and Information Technology

Automatic Generation of

System Test Cases For a Systems Under Development

This Is Summated As a Partial Requirement B.SC.(Honor) Degree In Software Engineering

Date: November 2017

By:

Abuhorira Faisal Abd Alraheem

Alsheikh Musa Mohammed Hamed

Makki Gamar Aldawla Makki

Zahier Bannaga Albashir Bannaga

Supervisor: Olla Faisal Noraldeen

Signature: ……

Date: ……

الآية

اعوذ بالله من الشيطان الرجيم

قال تعالي: (أَفَحَسِبْتُمْ أَنَّمَا خَلَقْنَاكُمْ عَبَثًا وَأَنَّكُمْ إِلَيْنَا لا تُرْجَعُونَ (115) فَتَعَالَى اللَّهُ الْمَلِكُ الْحَقُّ لا إِلَهَ إِلاَّ هُوَ رَبُّ الْعَرْشِ الْكَرِيمِ (116) وَمَن يَدْعُ مَعَ اللَّهِ إِلَهًا آخَرَ لا بُرْهَانَ لَهُ بِهِ فَإِنَّمَا حِسَابُهُ عِندَ رَبِّهِ إِنَّهُ لا يُفْلِحُ الْكَافِرُونَ (117) وَقُل رَّبِّ اغْفِرْ وَارْحَمْ وَأَنتَ خَيْرُ الرَّاحِمِينَ(118))

المؤمنون

إهداء

إلى من كلله الله بالهيبة والوقار .. إلى من علمني العطاء بدون انتظار .. إلى من أحمل أسمه بكل افتخار ..أرجو من الله أن يمد في عمرك لترى ثماراً قد حان قطافها بعد طول انتظار وستبقى كلماتك نجوم أهتدي بها اليوم وفي الغد و إلى الأبد

(والدي العزيز)

إلى ملاكي في الحياة .. إلى معنى الحب وإلى معنى الحنان والتفاني .. إلى بسمة الحياة وسر الوجود

إلى من كان دعائها سر نجاحي وحنانها بلسم جراحي إلى أغلى الحبايب

(أمي الغالية)

إلى من كانوا يضيئون لي الطريق ويساندوني ويتنازلون عن حقوقهم

لإرضائي والعيش في هناء

(إخوتي)

إلى توأم روحي ورفقاء دربي إلى أصحاب القلوب الطيبة والنوايا الصادقة إلى من رافقوني في دربي ونجاحي

(أصدقائي)

إلى كل من سقط من قلمي سهواً

أهدي هذا العمل

شكر وعرفان

إلى من علمتنا أن نقف و كانت سنداً لنا ولها الفضل في إرشادنا إلى الدرب القويم الأستاذة الجليلة

عُلا فيصل نور الدين وأيضا نتقدم بجزيل شكرنا إلى كل من مد لنا يد العون والمساعدة

في إخراج هذه الدراسة على أكمل وجه

Abstract

This study presents a tool that helps programmers to test their program using automatically generated test cases for return type methods without writing these test cases manually.

A tool does this by analyzing program under test and defines all methods that have testing values and then collecting these methods in a file and generate test cases for every returned value methods.

Each method has specific class saved in a file that contained the test cases and provides these test cases for executing without a need to write them manually.

The use of this method reduces the exploited effort in writing test cases manually so save time then accelerates the testing process.

Also, uses of this method accelerate the delivery time of the program of which accelerates program development and delivery is final.

المستخلص

هذه الدراسة تقدم أداة تساعد المبرمجين على إختبار برامجهم عن طريق كتابة حالات الاختبار لكل دالة تقوم بإرجاع قيمة تلقائيا بدون كتابة هذه الحالات يدويا.

وتقوم هذه الاداة بتحليل البرنامج قيد الإختبار وتحدد كل الدوال التي تقوم بإرجاع قيم و التي يوجد لها حالات إختبار وتجمع هذه الدوال في ملف وتولد حالات الإختبار لكل دالة على حدة وكل دالة لها صنف معين يحفظ في ملف هذا الملف يحتوي على حالات الإختبار الخاصة بالدالة وتكون حالات الإختبار جاهزة للتنفيذ من دون الحاجة ألى كتابتها يدويا.

هذه الطريقة تقلل من الجهد المبذول في كتابة حالات الإختبار يدويا وبالتالي توفر الوقت وتسرع من عملية إختبار البرنامج.

أيضا بإستخدام هذه الطريقة يتم تسريع عملية تسليم البرنامج ووضعه قيد الإستخدام مما يسرع عملية التطوير وتسليم البرنامج بشكل نهائي.

Table of Contents

Section Number	Section Name	Page Number
1.	**Introduction And Problem Statement**	
1.1	Introduction	1
1.2	Problem definition	2
1.2.1	Problems with manual Testing	2
1.2.2	Costs	2
1.2.3	Risks	3
1.2.4	Human resources	3
1.3	Automation testing	3
1.3.1	Problems with automation testing	3
1.4	Suggested solution	4
1.5	Research Questions	4
1.6	Importance of project	4
1.7	Scope of project	4
1.8	Objectives	4
1.9	Expected outcomes	5
1.10	Structure	5
2.	**Overview of the testing and previous studies**	
2.1	Introduction	6
2.2	General Concepts	6
2.2.1	Software testing	6
2.2.2	Testing Principles	7
2.2.3	Testing Objectives	7
2.2.4	Test-Case Design	8
2.2.5	Testing methods	8
2.2.5.1	White-Box Testing	8
2.2.5.2	Black-Box testing	9
2.2.6	Levels of Testing	10
2.2.6.1	Unit Testing	11
2.2.6.2	Integration Testing	11
2.2.6.3	System Testing	12
2.2.6.4	Acceptance Testing	12
2.2.7	Non-Functional Testing	13
2.2.7.1	Reliability Testing	13

2.2.7.2	Usability Testing	13
2.2.7.3	Security Testing	13
2.2.7.4	Efficiency Testing	13
2.2.7.5	Portability testing	13
2.3	Literature survey	14
2.3.1	Randoop	14
2.3.2	Jwalk	15
2.3.3	Tool for Automatic Test Case Generation in Spreadsheets	15
2.3.4	Jtest	16
2.3.5	Survey summary	17
3.	**Tools and techniques used in the Implementation**	
3.1	Introduction	18
3.2	UML	18
3.2.1	Use case Diagram	18
3.2.2	Activity diagram	19
3.2.3	Sequence diagram	19
3.3	Enterprise Architect	19
3.4	Eclipse IDE	19
3.5	Java language	20
3.5.1	Java Is	20
3.5.2	GUI	22
3.5.3	Why we use Java language	22
3.6	JUNIT	22
4.	**Analysis by UML diagram**	
4.1	Introduction	23
4.2	Functional Requirements	23
4.3	Nonfunctional Requirements	23
4.4	Use Case Diagram Figure	24
4.5	Activity Diagram Figure	25
4.6	Sequence Diagram Figure	26
4.6.1	Create Project	26
4.6.2	Code Writing	27
4.6.3	Code Saving	28
4.6.4	Project Opening	29
4.6.5	Test Case Generation	30
4.6.6	Test Case Displaying	31
5.	**Implementation Automatic Generation of System test cases for system under development**	
5.1	Introduction	32

5.2	Tools Process	32
5.2.1	Writing code	33
5.2.2	Import code	34
5.2.3	Generate Test Cases	35
5.2.3	Figure Test case for find maximum number method	36
5.2.3	Figure Test case for method that finds division result	37
5.2.3	Figure Test case for method that finds summation result	38
5.2.3	Figure Test case for method that finds subtract result	39
5.2.3	Figure Test case for method that finds multiple results	40
5.2.3	Figure Test case for method that finds minimum number result	41
5.3	Conclusion	41
6.	**Results and Recommendation**	
6.1	Introduction	42
6.2	Result	42
6.3	Conclusion	42
6.4	Recommendation	43
	References	
	References	44
7.	**Appendix**	
A	Business Process Modeling Notation(BPMN)	45
B	Analyze code	46

List of Figures

Figure Name	Page Number
Figure (2.1) White-Box Testing	9
Figure (2.2) Black-Box Testing	10
Figure (2.3) Levels Of Testing	11
Figure (2.4) Show Test Cases	16
Figure (4.1) Use Case Diagram	24
Figure (4.2) Activity Diagram	25
Figure (4.3) Create Project	26
Figure (4.4) Code Writing	27
Figure (4.5) Code Saving	28
Figure (4.6) Project Opening	29
Figure (4.7) Generate Test Case	30
Figure (4.8) Display Test Case	31
Figure (5.1) Main screen	32
Figure (5.2) Writing code	33
Figure (5.3) Import code	34
Figure (5.4) Example	35
Figure (5.5) Test case for find maximum number method	36
Figure (5.6) Test case for method that finds division result	37
Figure (5.7) Test case for method that finds summation result	38
Figure (5.8) Test case for method that finds subtract result	39
Figure (5.9) Test case for method that finds multiple results	40
Figure (5.10) Test case for method that finds minimum number result	41
Figure (7.1) BPMN	45

Chapter One
Introduction and Problem Statement

1.1 Introduction

Software engineering is aimed to development and design of high-quality software taking into account user requirements and customizations at all levels of software development lifecycle.

Concerned with the composition of software engineering program since its early stages during the analysis of the problem and then to find an adequate solution of that problem design and implement, test that program, and use that program, finally maintain it if there is a need to change or modify.

Software testing is one of the important parts of software development life cycle because that it is the stage at which we can ascertain the validity of the program have been developed, and determine the correctness of software under the assumption of some specific hypotheses by detecting the bugs, defects, and problems that lead failure of software.

The software historically evolved, in the beginning, it was the debugging oriented period, where testing was often associated to debugging: there was no clear difference between testing and debugging.In the period that followed there was the demonstration oriented period where debugging and testing was distinguished now - in this period it was shown that software satisfies the requirements.

Also in the period that followed is announced as the destruction oriented period, where the goal was to find errors. The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979 And the concept of software testing has been used in the 80s.[1]

The process of testing generally consumes between 30 and 60 percent of the overall development effort.

At the beginning the testing was performed manually and it performed without using any tool or test script the tester playing the role of end user but this type of testing has many problems such as it requires human resources, it consumes time ,also it has less reliable and cost for this it was developed to been execute automatically.In automation, the tester writes scripts and uses software to test the product. This process involves automation of a manual process.

But still, in this type, we write test cases manually to ensure the actual values of a program were agree with expected values.

We are going to develop a tool that generates test cases automatically instead of write manually.

1.2 Problem definition

Software testing includes testing software manually without using an automated tool. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. Testers must ensure the completeness of testing by use test plans, test cases, or test scenarios to test software, also involve exploratory testing, as testers explore the software to identify errors in it. [2]

1.2.1 Problems with manual Testing

There are many problems with manual testing it consumes human resources, increase risk and cost. Also, less reliability. Certain tasks are difficult to do manually.

1.2.1.1 Costs

- The manual creation of test cases needs high project efforts.
- Generation of test cases is consuming time.
- The traditional design of test cases is costly.

- Bad test cases lead to low production quality and high maintenance costs.
- We can't reuse manual tests if there is any change to the software; you have to run the tests again by hand. This is valuable time lost.

1.2.1.2 Risk

- not all test cases captured.
- Incomplete tests may have serious impacts on a production environment.
- Untested scenarios lead to failures.
- Untested technical interfaces lead to failures.

1.2.1.3 Human resources

Tester white test plan and perform testing, perform testing this need effort, cost, and it consumes time. This problem can solve by automation testing.

1.3 Automation testing

This is also known as test automation in automation testing tester writes scripts and uses software to test the product. This process involves automation of a manual process. [2]

Automated tests are easily repeatable and quickly executed and it reduces cost, time, human effort but it never replaces human testers, still need to write test cases manually there is some tools generate test cases automatically but it is not captured all test cases some was missing or ignored.

1.3.1 Problems with automation testing

- Automation testing tools may or may not generate test case automatically
- Automation testing tools (that generate test cases automatically) are generating test cases after program execution.

1.4 Suggested solution

Despite the use of automated testing we see that there is some problemssuch as still need to write test cases manually and the tools that generate test cases automatically not captured all test cases, based on this we will go to develop tool that generates test cases automatically while programmer writing his code; when he completes his program then we save all test cases of program in library and running test cases without comeback to code.

1.5 Research Questions

- Is it possible to develop a tool that generates test cases automatically?
- Is this tool are covered all possible test cases?
- What is the methodology by which this tool can be developed?

1.6 Importance of project

Reduces cost and time of created test cases during the coding time by a tool that generates test cases automatically while programmer writing his code this reduces the time of delivery by reducing the time of testing.

1.7 Scope of project

This project focuses on a generation of the all possible test cases for returned value methods of a program under test automatically.

1.8 Objectives

- Automatic generation of test cases (reducing the time of test cases generation).
- Minimize the maintenance cost of the test.

- Produce high-quality test cases (that ensure reliable coverage).
- Developexecution tool to execute all generated test cases automatically.
- Improve performance of test case generation and execution.

1.9 Expected outcomes

- A tool that generates test cases automatically for a system under test.
- Make automation testing more powerful (generate test cases during coding).
- Make generation of test case more simple and quick.

1.10 Structure

This is research is divided into six chapters as follow after this introductory first chapter, second chapter discusses general concepts about testing and literature survey, third chapter discusses technologies, tool and techniques used in this research, chapter four discusses analysis of the system, chapter five discusses the implementation of the system, and the last chapter contains the result of this research and recommendation.

Chapter Two
Overview of the Testing and previous studies

2.1 Introduction

In this chapter, we will explain the most important concepts in the software testing of the types of testing and test of levels and testing methodologies and previous studies conducted in this field of generating test cases and software testing.

2.2 General concepts

The software testing became an independent and professional specialty in itself. It was common practice in the field of software that the developer tests his software manually, while now find tools for testing the software, the main goal of the research is to developa tool that generates test cases automatically during coding, in order to improve a testing process and develop it.

2.2.1 Software testing

Is a set of activities that can be planned in advance and conducted systematically.Forthis reason, a template for software testing a set of steps into which we can place specific test-case design techniques and testing methods should be defined for the software process. [3]

Also is a process of demonstrating that errors are not present. [4]

As well as the process of executing a program with the intent of finding errors. [4]

From all above definitions, testing means an activity performed for evaluating a product or system quality for improving it by identifying defect and problems in the product.

Testing has many levels of testing (unit testing, module testing, integration testing, system testing, acceptance testing). We are focus In this level of testing Unit Testing which can be performed via black box method or white box method.[5]

2.2.2 Testing Principles

There are many principles that guide software testing. Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing. The following are the main principles for testing:

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- The Pareto principle applies to software testing.
- Testing should begin "in the small" and progress toward testing "in the large."
- Exhaustive testing is not possible.
- To be most effective, testing should be conducted by an independent third party.[3]

2.2.3 Testing Objectives

The testing objective is to test the code, whereby there is a high probability of discovering all errors.

This objective also demonstrates that the software functions are working according to software requirements specification (SRS) with regard to functionality, features, facilities, and performance. It should be noted, however, that testing will detect errors in the written code, but it will not show an error if the

code does not address a specific requirement stipulated in the SRS but not coded in the program.

Testing objectives are:

- A good test case is one that has a high probability of finding an as-yet-undiscovered error.
- A successful test is one that uncovers an as-yet-undiscovered error.

2.2.4 Test-Case Design

A test case is a set of instructions designed to discover a particular type of error or defect in the software system by inducing a failure. [3]

The goal of selected test cases is to ensure that there is no error in the program and if there is it then should be immediately depicted. Ideal test casement should contain all inputs to the program. This is often called exhaustive testing. There are two criteria for the selection of test cases:

- Specifying a criterion for evaluating a set of test cases.
- Generating a set of test cases that satisfy a given criterion.[3]

2.2.5 Testing methods

There are two basic types of testing methods White-Box Testing and Black-Box Testing. These two basic are used to describe the point of view that test engineering tasks when designing test cases.

2.2.5.1 White-Box Testing

Tester used source code for testing, White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or open-box testing. In order to

perform white-box testing on an application, a tester needs to know the internal workings of the code.

White-box testing is also known by other names, such as glass-box testing, structural testing, clear-box testing, open-box testing, logic-driven testing, and path-oriented testing. In white-box testing, test cases are selected on the basis of examination of the code, rather than the specifications.[3]

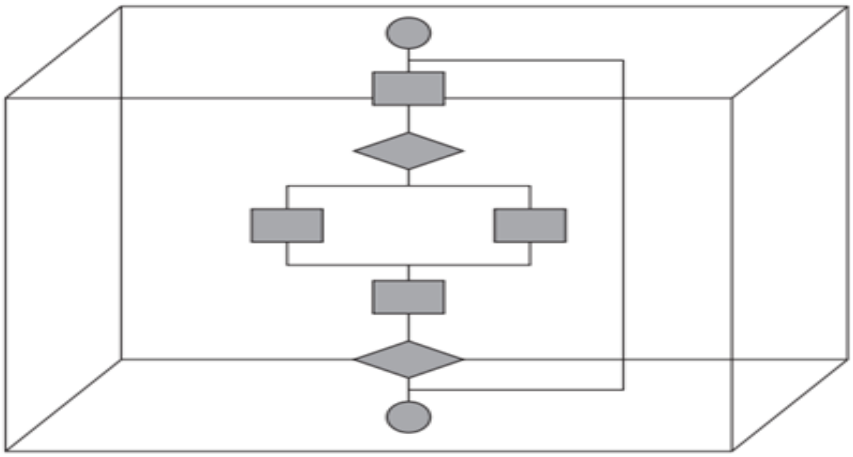

Figure (2.1) White-box testing [3]

2.2.5.2 Black-Box Testing

The tester not having any knowledge of the interior structure of product the tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

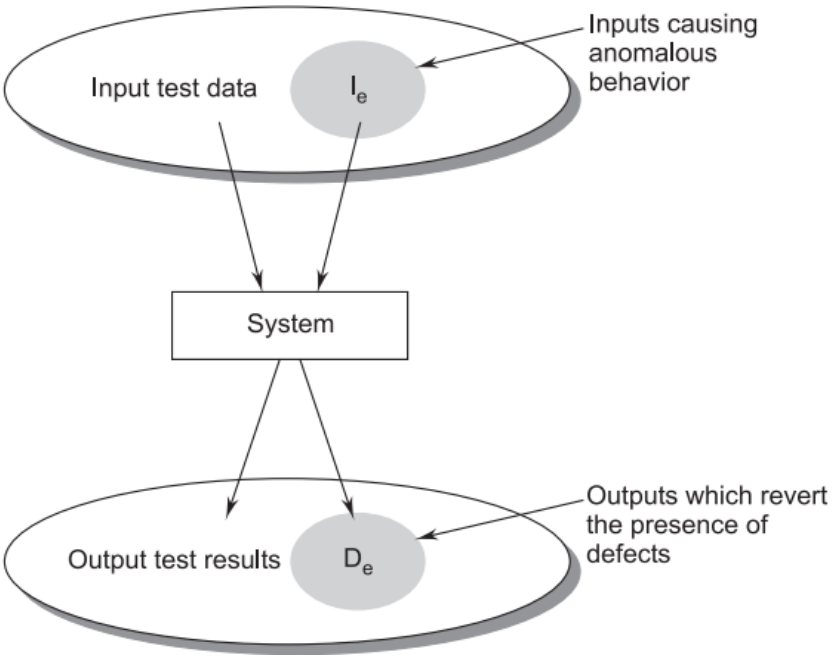

Figure (2.2) Black-box testing [3]

2.2.6 Levels of Testing

Testing Levels Based on Software Activity Tests can be derived from requirements and specifications, design artifacts, or the source code. A different level of testing accompanies each distinct software development activity.

There are many levels during the process of testing. In this part, a brief description is provided about these levels. [5]

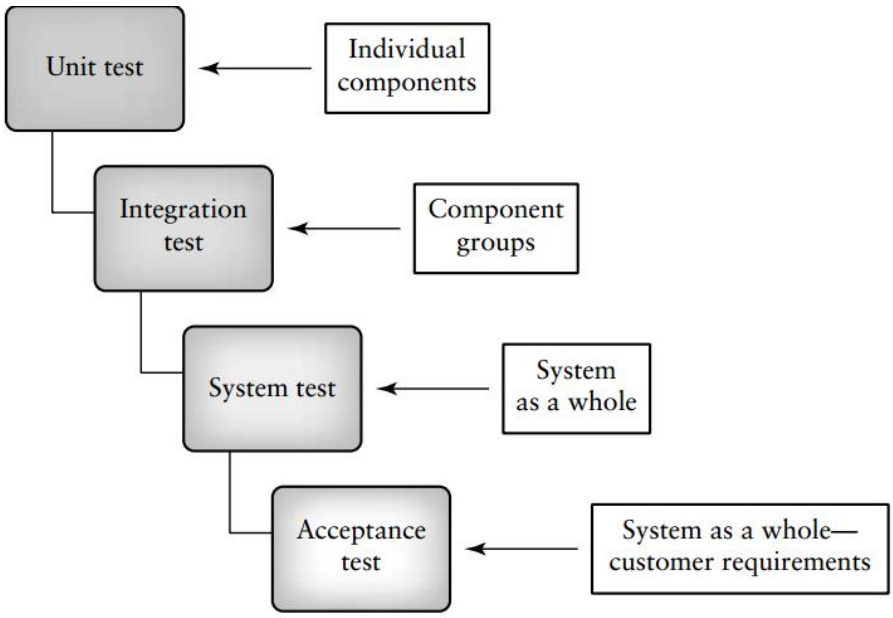

Figure (2.3) Levels of testing [5]

2.2.6.1 Unit Testing

Unit testing is the smallest possible testable software component. This type of testing has involved the Functions, Procedures, Classes, and Methods as Units performed by developers before the setup is handed over to the testing team to formally execute the test cases. [5]

2.2.6.2 Integration Testing

Integration test for procedural code has two major goals:

- to detect defects that occur on the interfaces of units.
- to assemble the individual units into working subsystems and finally

a complete system that is ready for system test. [5]

Approaches to Integration Testing there are various approaches used for integration testing are:

- Incremental Approach
- Top-down Integration
- Bottom-up Integration
- Regression Testing
- Smoke Testing
- Sandwich Integration

2.2.6.3 System Testing

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. [5]

There are several types of system tests are as follows:

- Functional testing
- Performance testing
- Stress testing
- Configuration testing
- Security testing
- Recovery testing

2.2.6.4 Acceptance Testing

Validation testing with respect to user needs, requirements, and business processes conducted to determine whether or not to accept the system. [5]

2.2.7 Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Nonfunctional testing involves testing software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc. [5]

2.2.7.1 Reliability Testing

Is defined further into the sub-characteristics maturity (robustness), fault-tolerance, recoverability and compliance [5]

2.2.7.2 Usability Testing

Is divided into the sub-characteristics understandability, learnability, operability, attractiveness, and compliance. [5]

2.2.7.3 Security Testing

Security testing involves testing software in order to identify any flaws and gaps from security and vulnerability point of view. [5]

2.2.7.4 Efficiency Testing

Is divided into time behavior (performance), resource utilization and compliance. [5]

2.2.7.5 Portability Testing

It consists of five sub-characteristics: adaptability, installability, co-existence, replaceability, and compliance. [5]

2.3 Literature survey

In the following paragraph we will present some studies and techniques that have been generated to generate automated test case we will explain the problems and defects in these techniques and the following studies and we will compare these techniques.

2.3.1 Randoop

Randoop is a unit test generator for Java programming language. Randoop is used to automatically generate a test for classes in Java language programs, by using JUnit format and feedback-directed random test generation. This technique randomly, but smartly, generates sequences of method and constructor invocations for the classes under test. The Randoop has two type of test output. The first is to test code errors in java language The second output is tested that program still performance correctly after the change.[9]

A disadvantage of Randoop was the high cost of test code maintenance. The regression tests generated in 30 seconds ran to some 96K lines of test code. Another limitation was the inability to control values supplied as arguments to tests (which were randomized), making it impossible to guarantee equivalence partition coverage on inputs.

We will try to solve this problem by developing a fully automatic tool that generates all test cases for all methods have returned value in the code.

2.3.2 JWalk

JWalk is a unit testing feature for the Java programming language. Developed by Anthony Simons, it supports a testing paradigm called Lazy Systematic Unit Testing. This is based on the two notions of a lazy specification, the ability to infer the evolving specification of a class on the fly by dynamic analysis, and systematic testing, the ability to explore and test the class's state space exhaustively to bounded depths. It used to test single and compiled classes in the Java programming language. [9]

2.3.3 Auto Test: A Tool for Automatic Test Case Generation in Spreadsheets

This paper presents a system that helps user's tests their spreadsheets using automatically generated test cases.

The system generates the test cases by backward propagation and solution of constraints on cell values, these constraints are obtained from the formula of the cell that is being tested when we try to execute all feasible definition of use (DU) associations within the formula.

The system generates a set of candidate test cases for the formula in the cell and presents it to a user as shown in Figure.

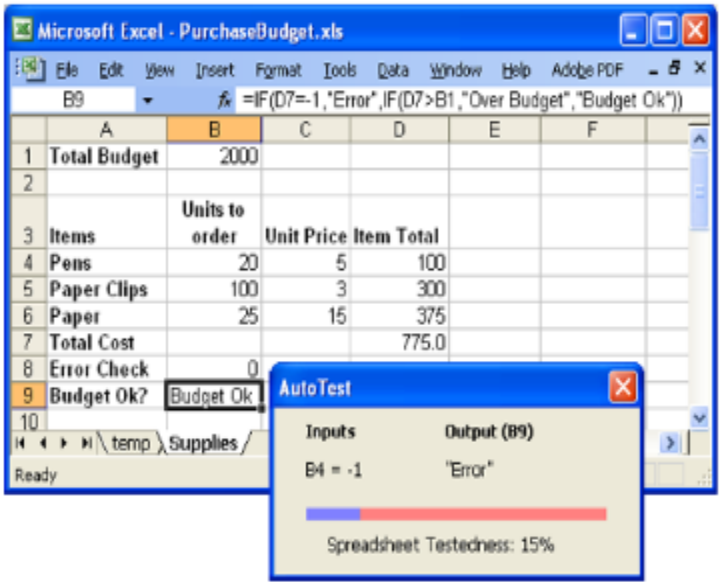

Figure (2.4) Show test cases[10]

Users can validate generated test cases also can ignore generated test cases if they are unable to decide if the computed output is right or wrong.

This is a good tool to generate test cases for spreadsheets but still generate test cases specific for spreadsheets only.

That we try to solve it by a tool that generates test cases not prompt and for all. [10]

2.3.4 Jtest

"Jtest Its purpose is to help you increase your Java software reliability while dramatically reducing the amount of time you spend testing. Jtest is designed to reduce the burden on Java developers as they write and test their programs from the class level at the earliest stages of development. Jtest automates four essential types of Java testing including--for the first time in software development history automating black-box testing by reading Design by Contract information built into your class with the Design by Contract language. Jtest then automatically creates and executes test cases based on these specifications. Jtest gives you the detailed

error information you need to make your JavaBeans, APIs and libraries, applications, and applets more robust.
Jtest automatically performs white-box testing, black-box testing, regression testing, and static analysis (coding standard enforcement) of Java code

During black-box testing Jtest automatically reads Design by Contract specification information built into a class with the Design by Contract language, then automatically creates and executes test cases that verify the functionality of the specification. Jtest can also automatically provide a set of inputs based on sophisticated analysis and then executes the class with the inputs. You may also provide your own sets of inputs to be used by Jtest.

Design by Contract is a formal way of using comments to incorporate specification information into the code itself. Basically, the code specification is expressed unambiguously using a formal language that describes the code's implicit contracts. Disadvantages of just are a Non Intuitive interface, Rules Builder can be frustrating at first, Pricey ". [11]

2.3.5 Survey summary

From all these studies and tools we recognized that all of these are generating test cases but there is some of the limitation such as not all test cases will be captured some are missing and some are ignored.

We find that JWalk tool is not open source, Randoop not fully automatic and take, auto test generate test cases for spreadsheets the only Jtest is not free, unfamiliar interfaces.

Chapter Three

Tools and techniques used in the Implementation

3.1 Introduction

In this chapter we will present the techniques, tools, and languages we have used and will try to clarify the advantages of these techniques and tools to implement this project.

3.2 UML

The Unified Modeling Language (UML) is general purpose visual modeling language that is used to specify, visualize, construct and document the artifacts of a software system. [6]

We have used the UML language because it provides a simplified graphical way to express the different software models made easy by it to the relevant stakeholders of analysts, designers, programmers and even the beneficiaries to communicate among themselves and pass information in a standardized format and concise, enriches them from the usual linguistic description

3.2.1 Use case Diagram

Use case Diagram is a graphical method that describes the interaction of the user with the system, which describes the basic processes performed by the user in the proposed system or system under development where the user represents the actor and the basic processes represent the Use case.

We used this diagram to describe the basic processes performed by the user in the proposed system.

3.2.2 Activity diagram

An activity diagram is a graphical schema that describes the sequence of activities from the beginning of the first activity to the end of the last activity in the system and there are conditional activities in order to move from one activity to another and parallel activities in the system to be developed.

3.2.3 Sequence diagram

A sequence diagram is used to represent the flow of messages, events, and actions between the objects or components of a system. The horizontal dimension shows the objects participating in the interaction, and the vertical arrangement of messages indicates their order.

3.3 Enterprise Architect

Sparx Systems' Enterprise Architect (EA) is a Computer Aided Software Engineering (CASE) tool for designing and constructing software systems, for business process modeling, and for more generalized modeling purposes. EA is based on the UML 2.1 specification, which defines a visual language that you use to model a particular domain or system (either existing or proposed).

EA is a progressive tool that supports all aspects of the development cycle, providing full traceability from the initial design phase through to deployment and maintenance. It also supports testing and change control.

3.4 Eclipse IDE

Eclipse is an IDE for "anything, and nothing at all," meaning that it can be used to develop software in any language, not just Java. It started as a proprietary replacement for Visual Age for Java from IBM but was open sourced in November

2001. Eclipse is now controlled by an independent nonprofit organization called the Eclipse Foundation. [7]

3.5 Java language

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling and released in 1995 as a core component of Sun Microsystems' Java platform (Java 1.0 [J2SE]).

The latest release of the Java Standard Edition is Java SE 8. With the advancement of Java and its widespread popularity, multiple configurations were built to suit various types of platforms. For example J2EE for Enterprise Applications, J2ME for Mobile Applications. On 13 November 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

On 8 May 2007, Sun finished the process, making all of Java's core code free and open-source, aside from a small portion of code to which Sun did not hold the copyright.

The new J2 versions were renamed as Java SE, Java EE, and Java ME respectively. Java is guaranteed to be Write Once, Run Anywhere. [8]

3.5.1 Java

Object-oriented in Java, everything is an Object. Java can be easily extended since it is based on the Object model.

Platform Independent unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent bytecode. This bytecode is distributed over the

web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

- **Simple** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- **Secure** with Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- **Architecture-neutral** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.
- **Portable** Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- **Robust** Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.
- **Multithreaded** with Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- **Interpreted** Java bytecode is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- **High Performance** with the use of Just-In-Time compilers, Java enables high performance.
- **Distributed** Java is designed for the distributed environment of the internet.

- **Dynamic** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.[8]

3.5.2 GUI

Stands for Graphical User Interface this term not used only in Java but in all programming languages that support the development of GUIs. A program's graphical user interface presents an easy-to-use visual display to the user. It is made up of graphical components (e.g., buttons, labels, windows) through which the user can interact with the page or application. To make graphical user interfaces in Java, use either Swing (older applications) or JavaFX.[8]

3.5.3 Why we use Java language

One of reason it's rich API and most importantly it's highly visible because come with Java installation ‘also powerful development tools e.g. Eclipse, NetBeans, that id's helps in code completion also provides powerful debugging capability, which is essential for real-world development' Great collection of Open Source libraries like Apache, Google, and other organization' Java is free ' Excellent documentation support Java docs and Java is Everywhere.

3.6 JUNIT

JUnit is a unit testing framework for the Java programming language. Units are the smallest module of functionality in a computer program. These are usually in the form of a method. Therefore, JUnit is most commonly used to test the functionality of individual methods. Experience with JUnit has been important in the development of Test-Driven Development.

Chapter Four

Analysis by UML diagram

4.1 Introduction

In this chapter will discuss the general shape of the project by analysing and designing the basic processes of the proposed system through the use of several techniques, including the Unified Modeling Language and the programming languages used and other necessary techniques, and how to design and implement the project generating test cases during the software development process.

4.2 Functional Requirements

- Develop a tool that generates test cases for a system under development process.
- A Tester can create a project to generate test cases for it.
- A Tester can generate test cases for the existing system.
- A Tester can write his own code to generate test cases for it.
- A Tester can extract generated test cases from a log file.
- The system must generate test cases for all methods in project code.

4.3 Nonfunctional Requirements

- The system must be easy to use for a user (Usability).
- The system must consist of all test cases for all methods of code (consistency).
- The system must perform and give result in less than 10 seconds
- The system must generate all possible test cases for methods and not ignore any method.
- The system must decrease consuming of resources.

Use Case Diagram Figure

This figure shows the main processes performed by the user in the system

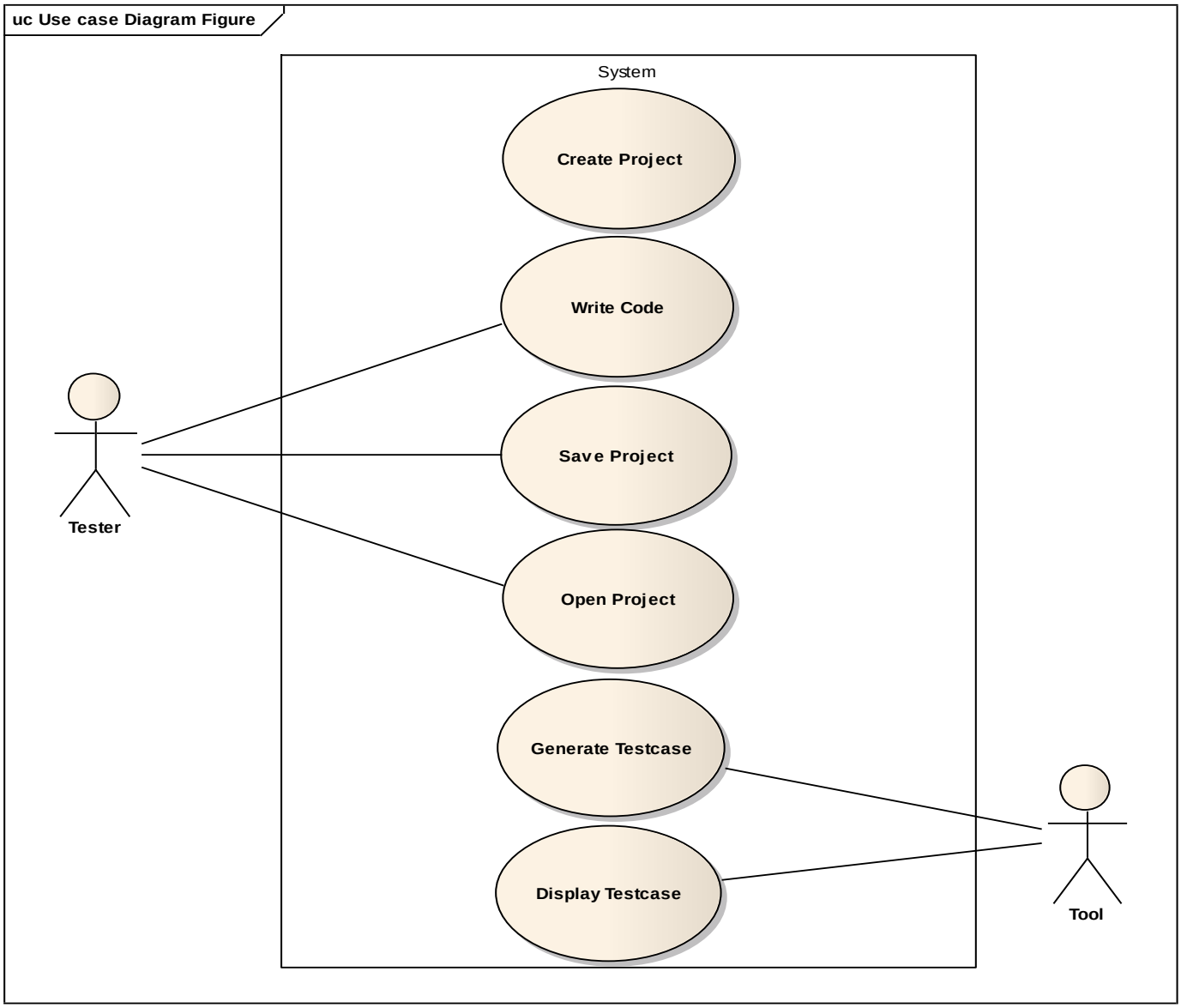

Figure (4.1) Use case Diagram

4.4 Activity Diagram Figure

This figure shows the activity performed by the user in the system

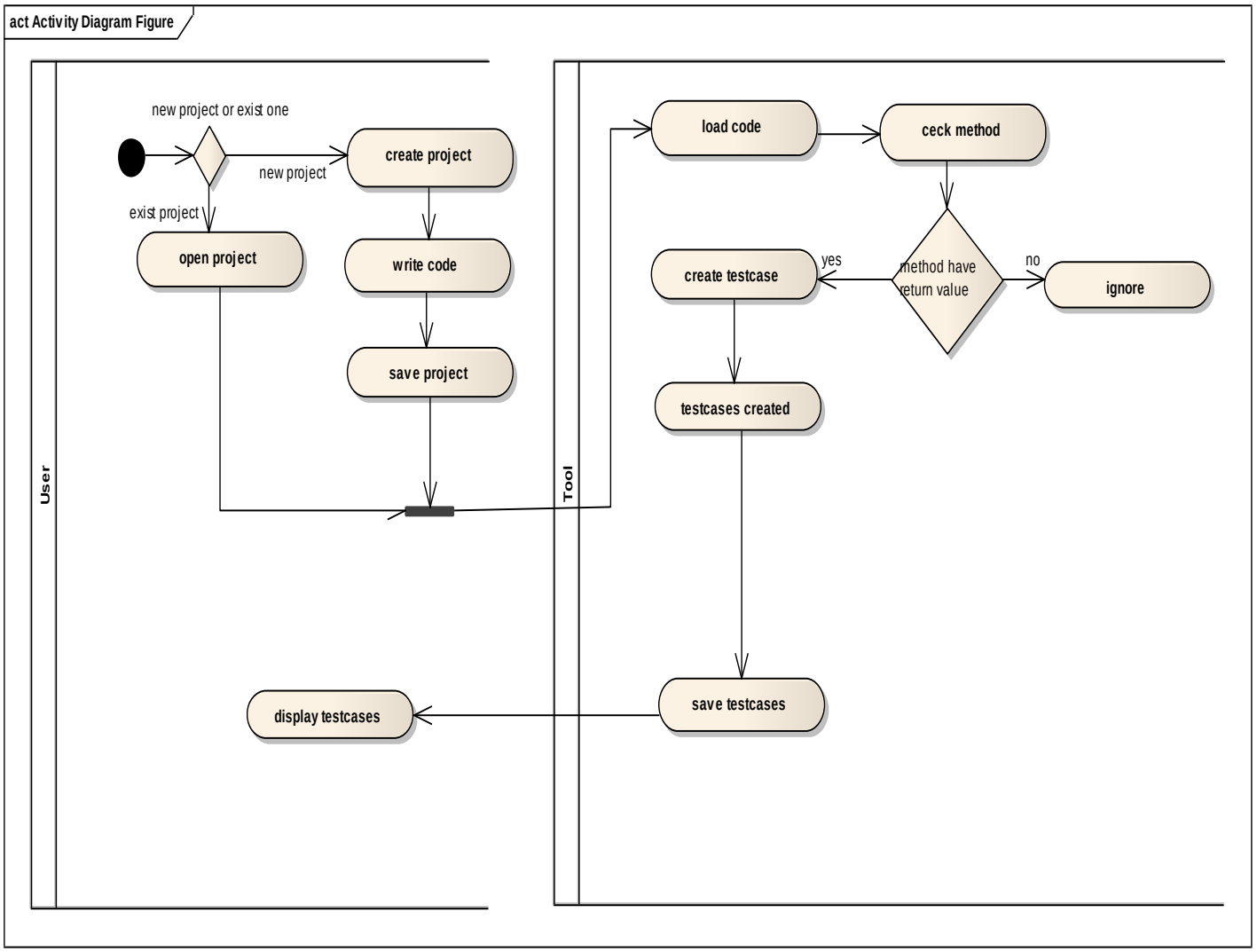

Figure (4.2) Activity Diagram

4.5 Sequence Diagram Figure

This figure shows project creation, by the user in the system

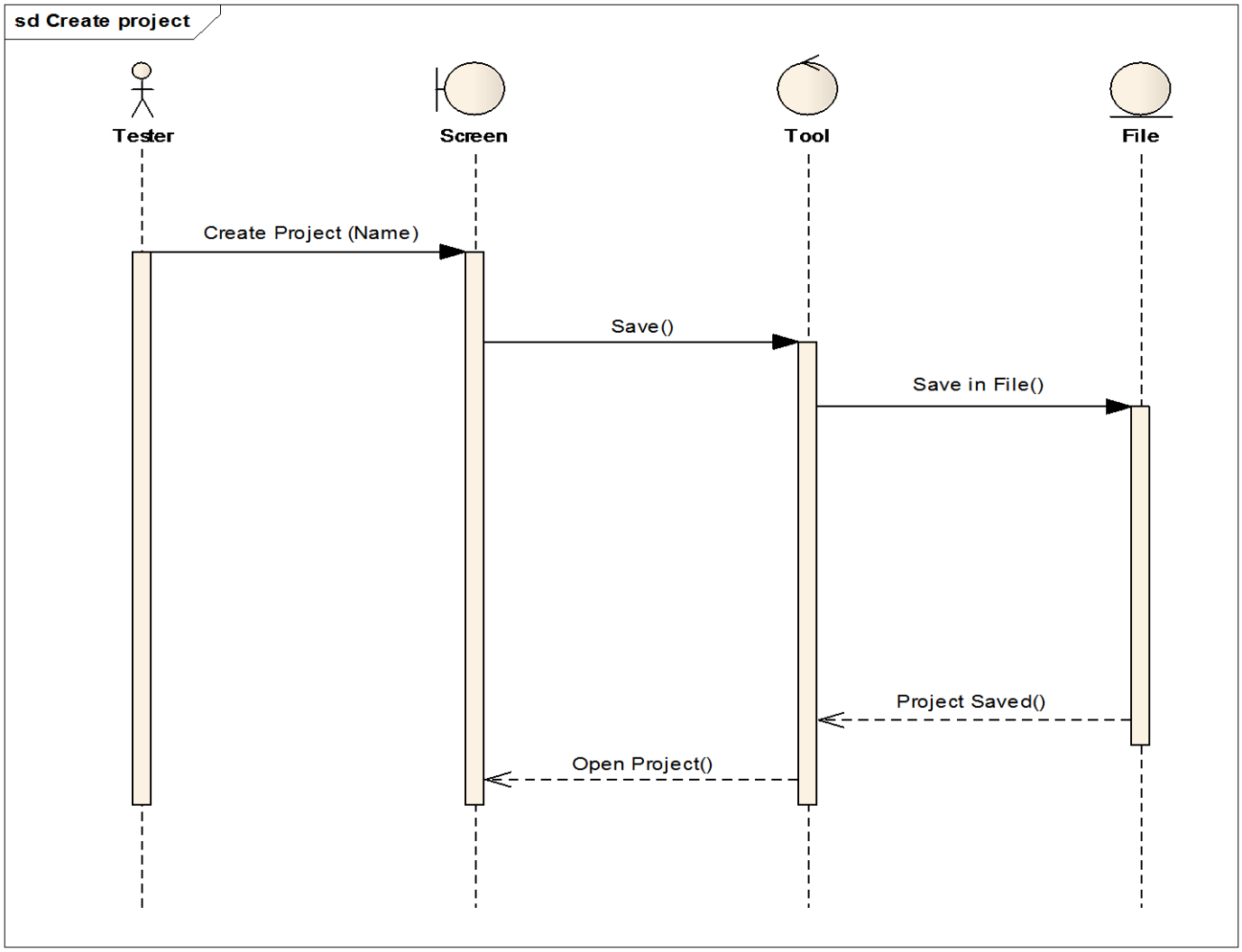

Figure (4.3) Create Project

This figure shows code writing by the user in the system

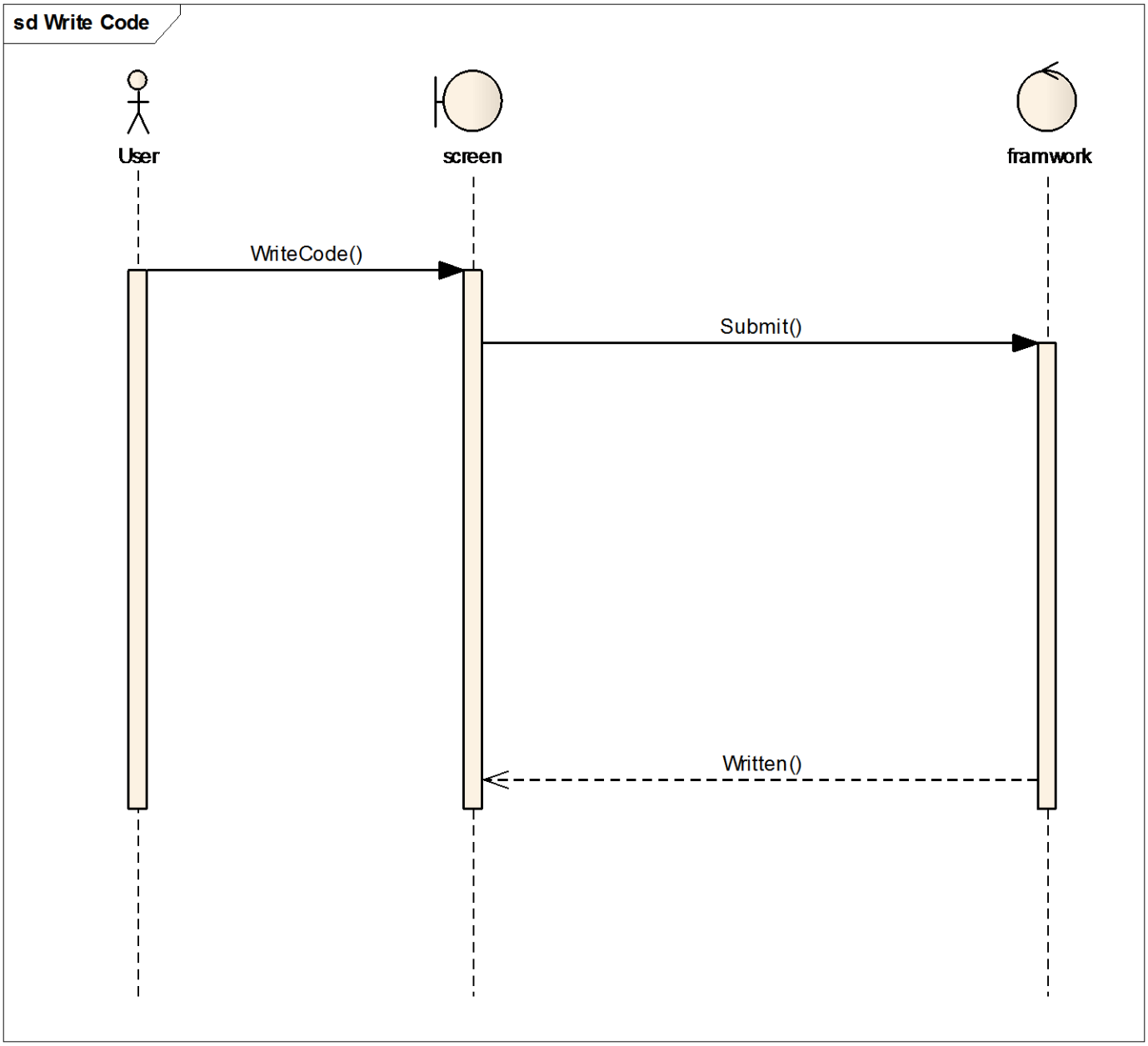

Figure (4.4) Code Writing

This figure shows code saving by the user in the system

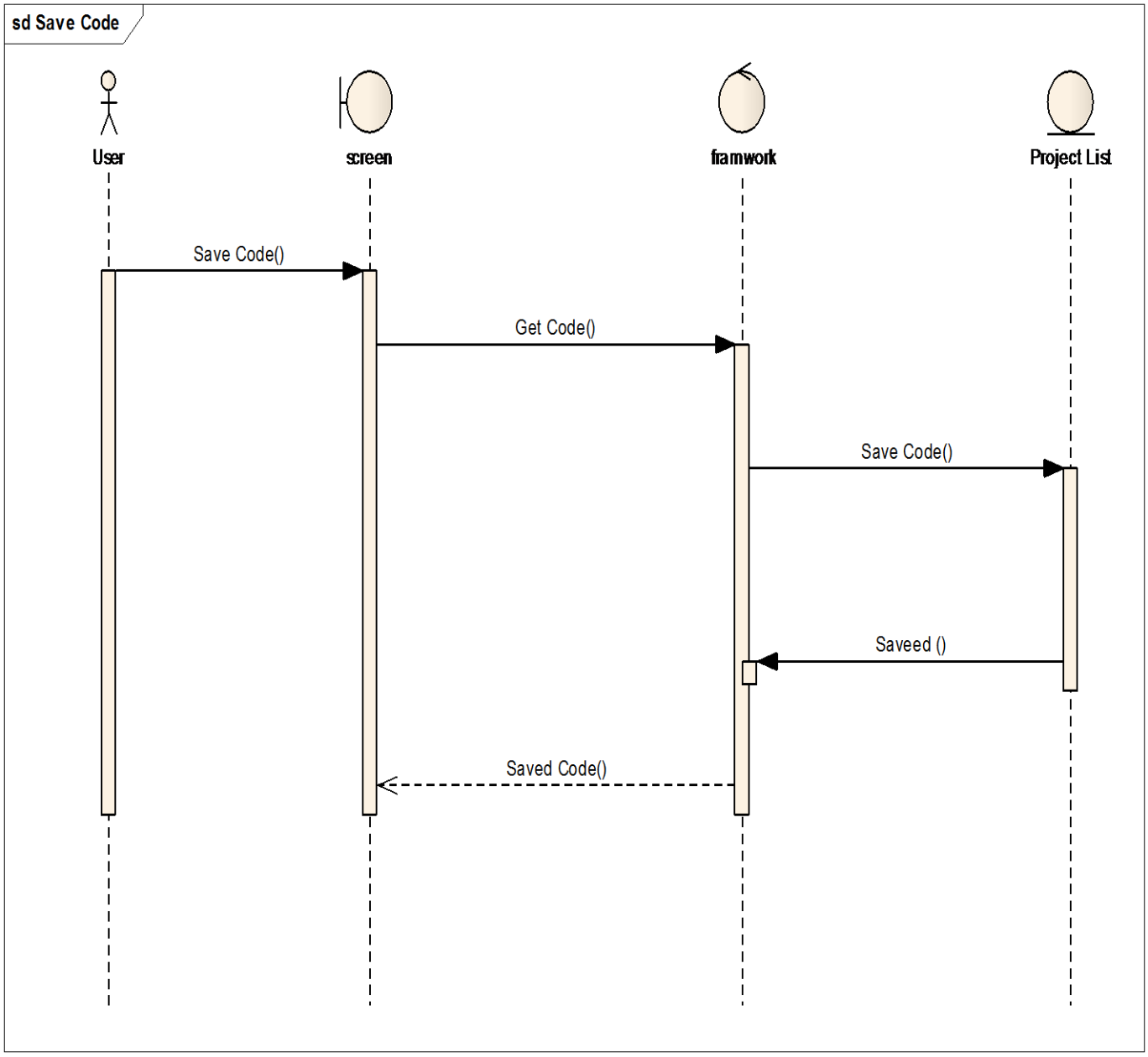

Figure (4.5) Code saving

This figure shows project openingby the user in the system

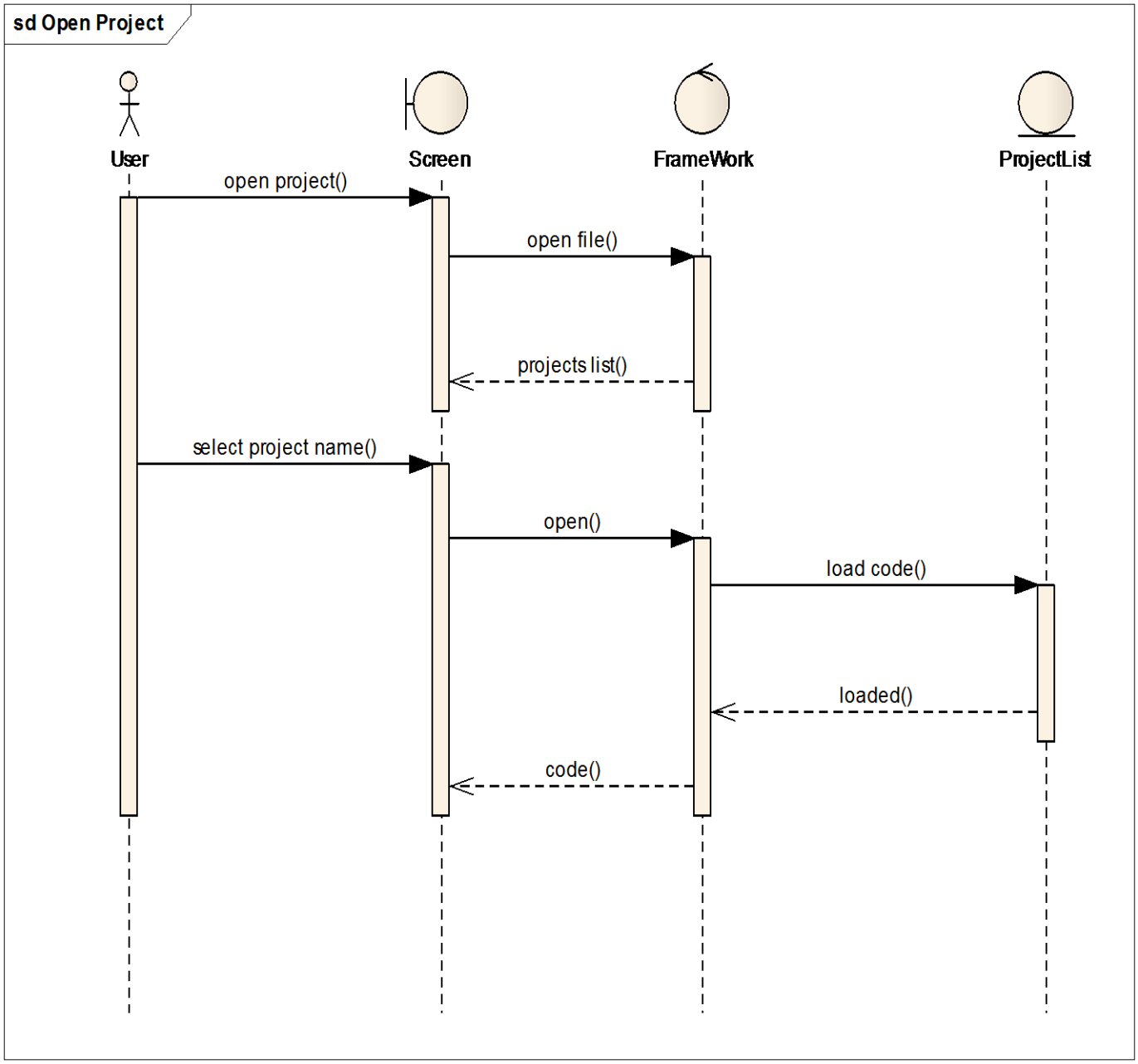

Figure (4.6) Project Opening

This figure shows generation of test cases

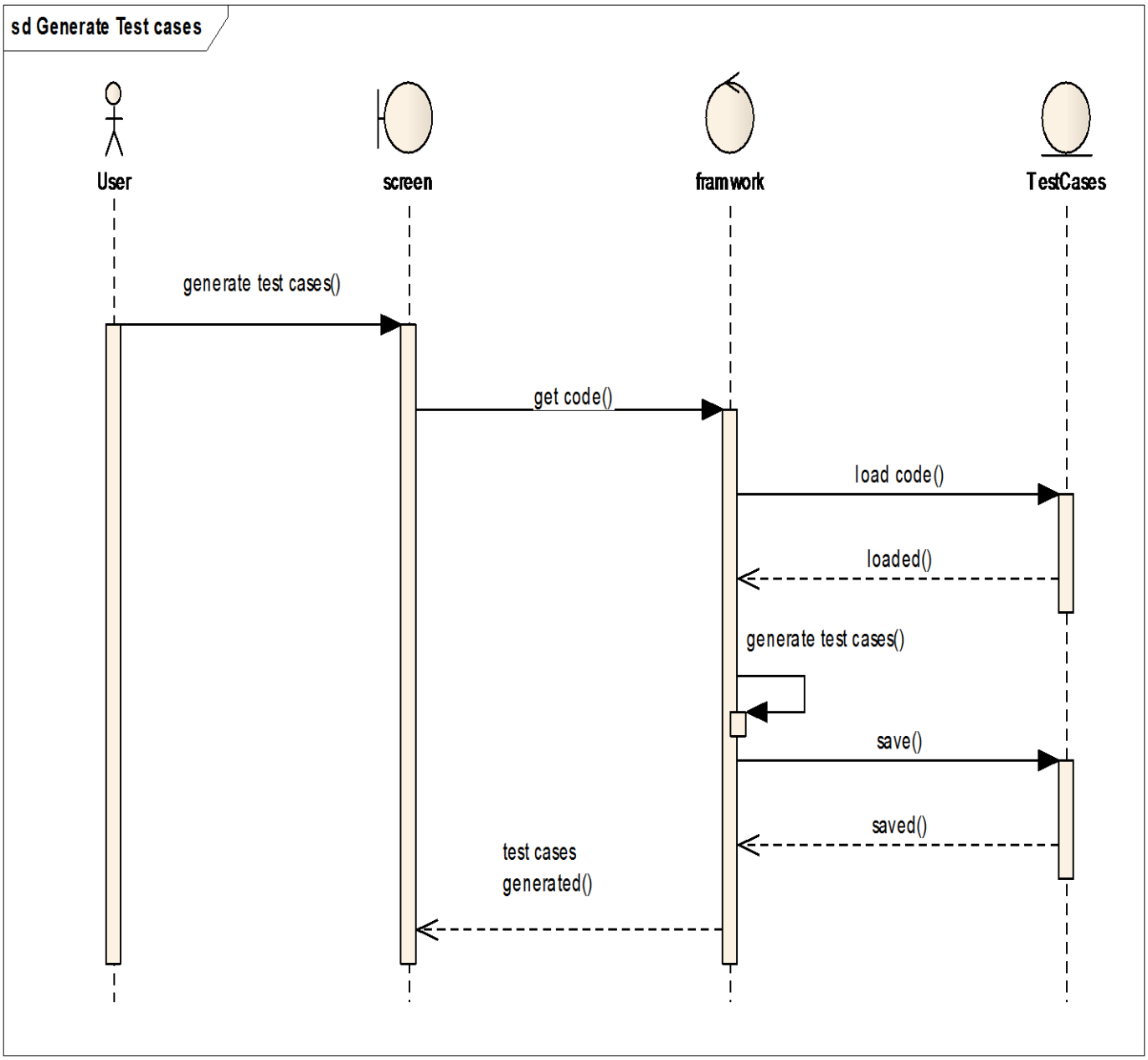

Figure (4.7) Test Case Generation

This figure shows test case displaying

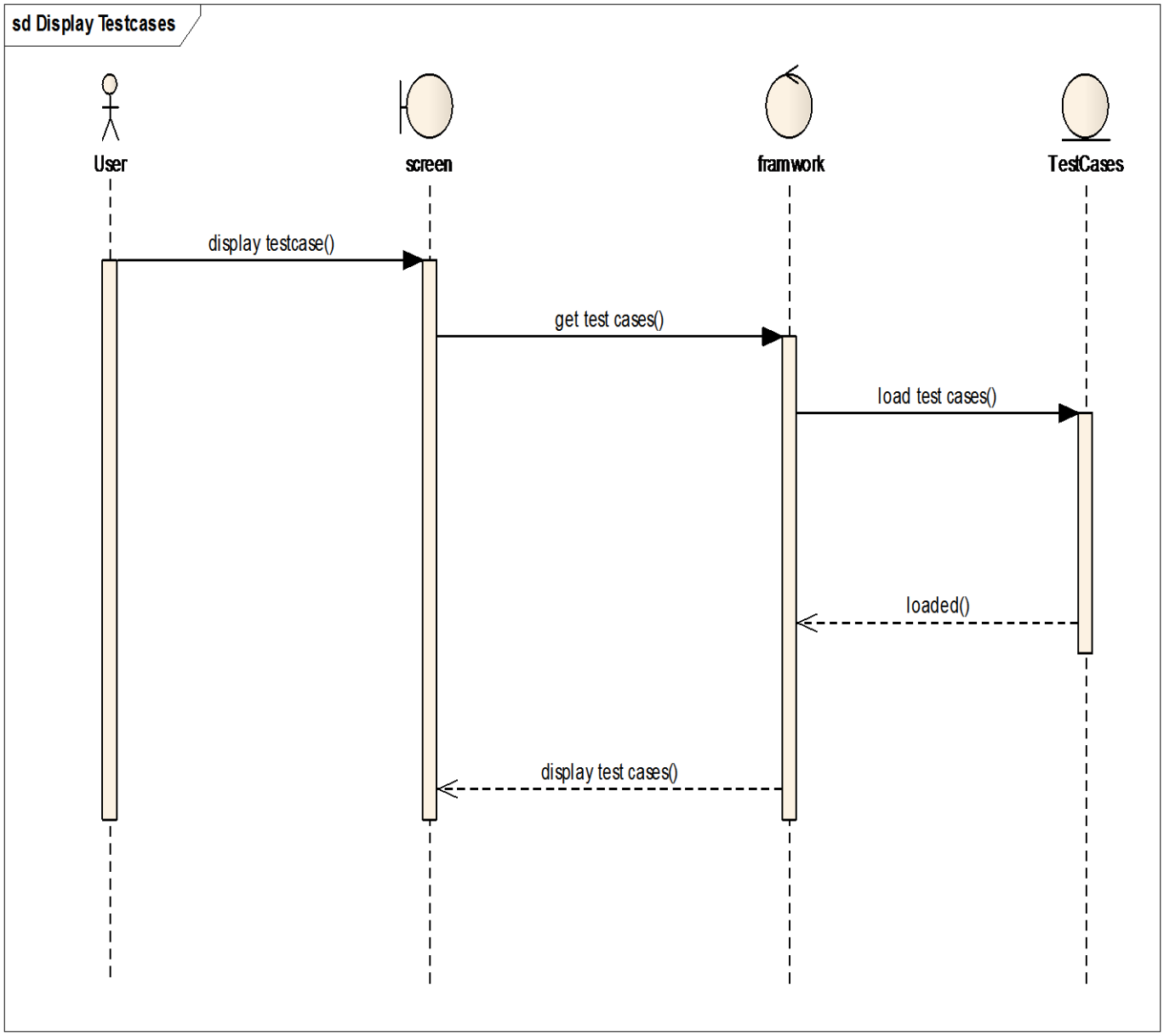

Figure (4.8) Test Case Displaying

Chapter Five

ImplementationAutomatic Generation of System test cases for system under development

5.1 Introduction

This chapter will explain the process of generating test cases for methods that have returned values.

5.2 Tool process

The programmer writes his program under test either that write his code in an area with a tool or import it from an external file the tool take the source code that analyzes it to find out all methods that have returned values.

After defining methods tool take every method separately and generate a test case for that method and save it in a file with the same name of the method

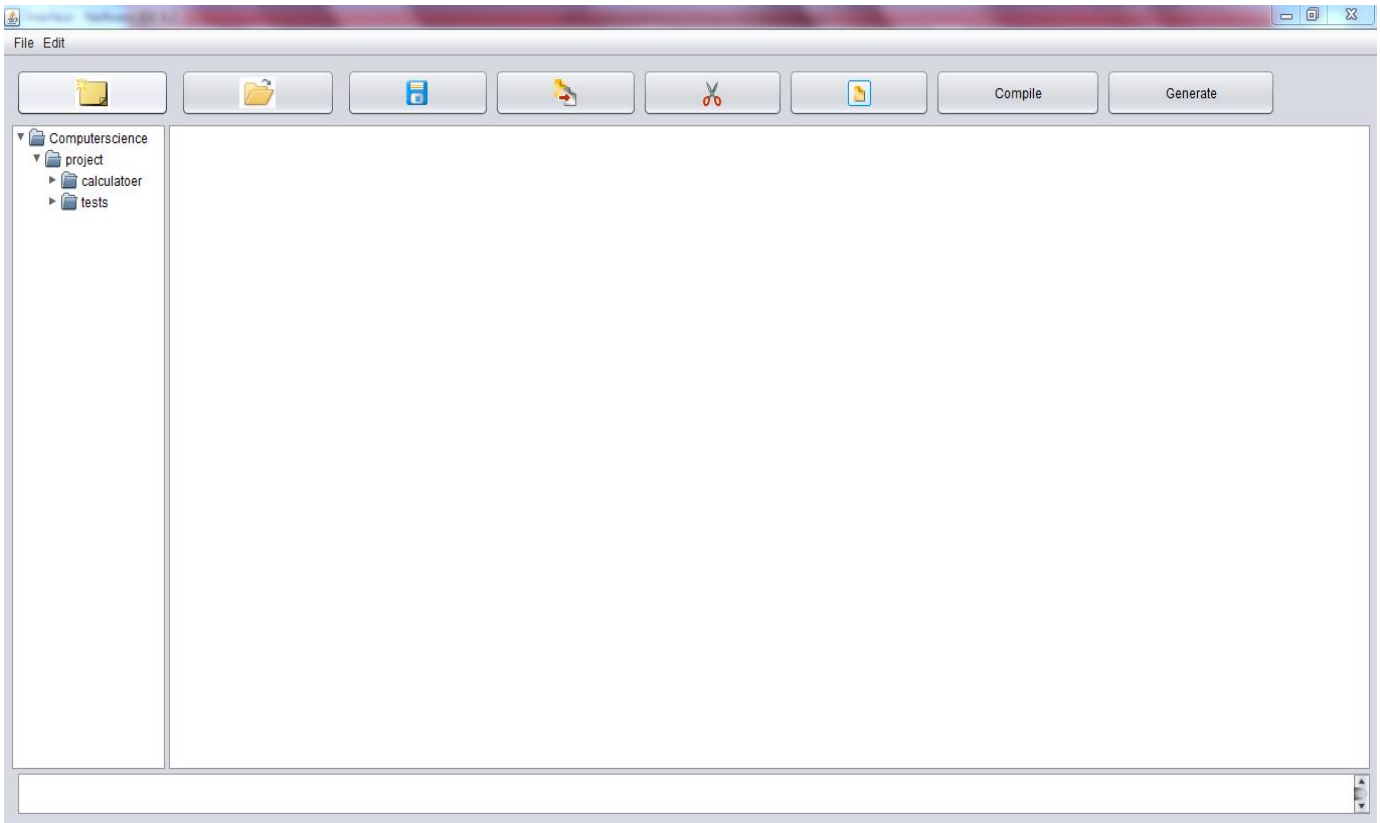

Figure (5.1) Main screen

5.2.1 Writing code

The tester writes his code under test in an area with the tool.

The figure below shows the main screen for writing code

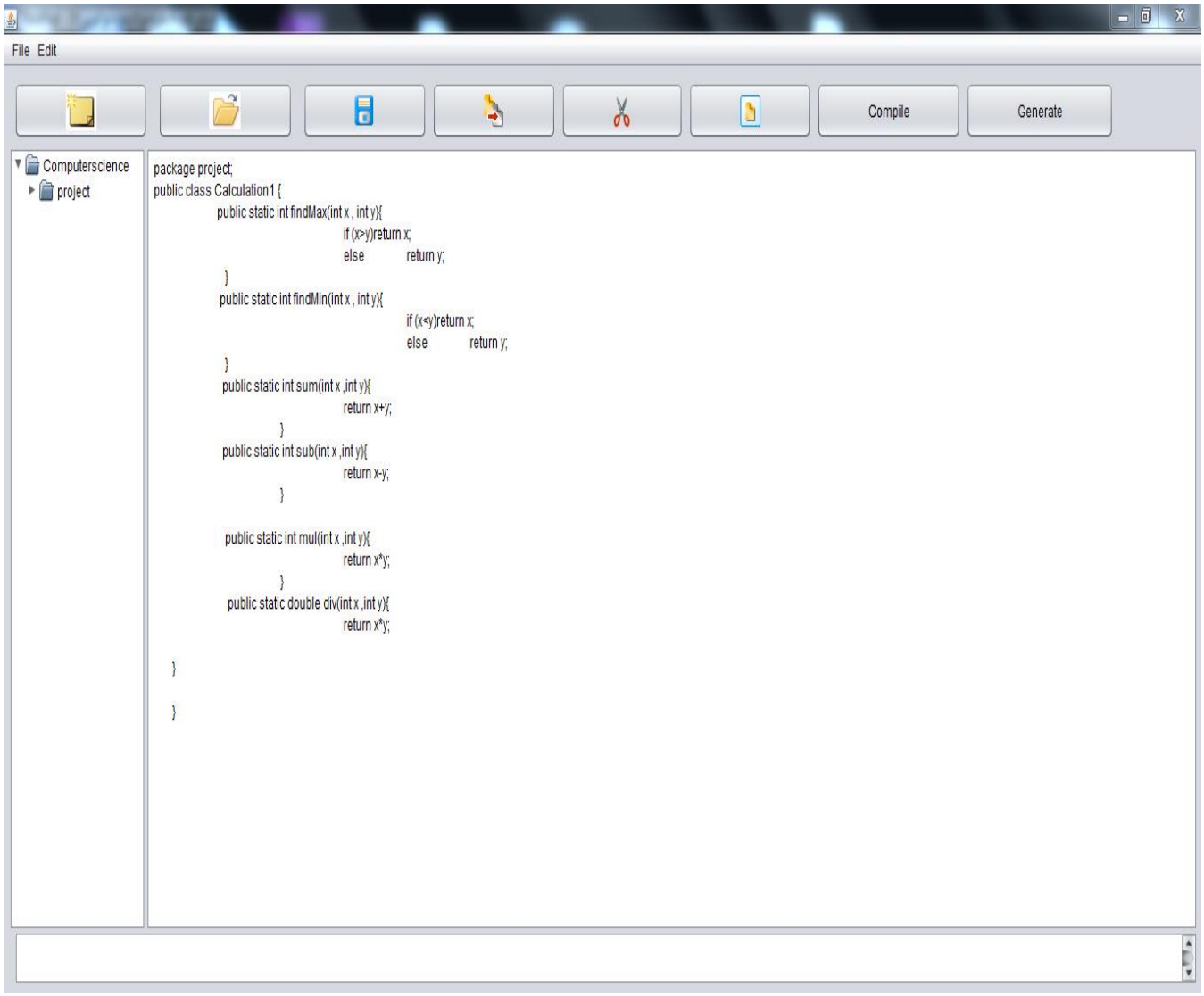

Figure (5.2) Writing code

5.2.2 Import code

The programmer import code from external file

The figure below shows the main screen for writing code

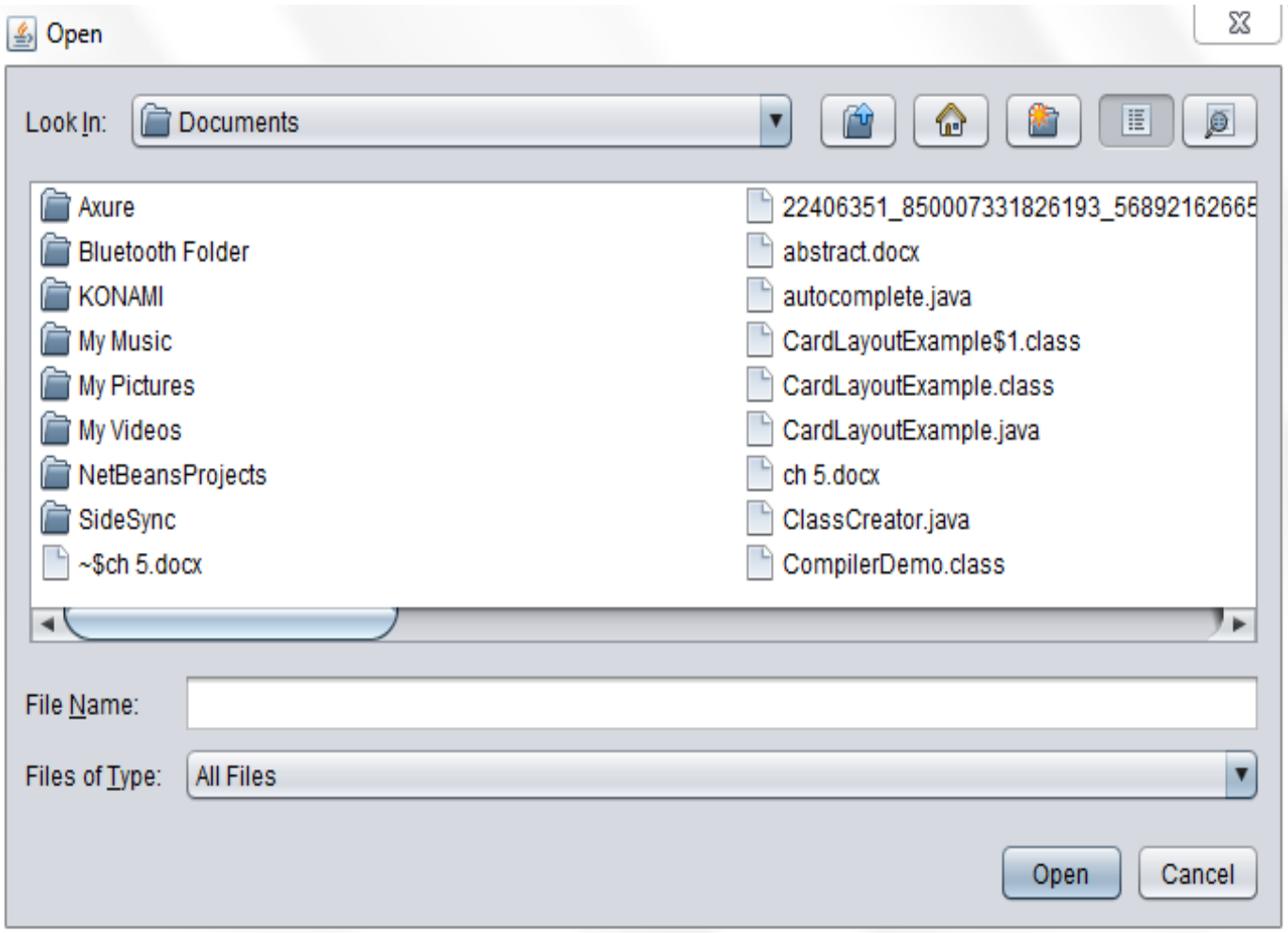

Figure (5.3) Import code

5.2.3 Generate test cases

After tool identifies methods with test values it takes every method and creates a class contained test case of the method that returns types.

In the example below we have six methods and tool going to generate test cases for them:

```
package examples;
public class Calculation1 {
    public static int findMax(int x , int y){
            if (x>y)return x;
            else    return y;
      }
     public static int findMin(int x , int y){
               if (x<y)return x;
               else    return y;
      }
      public static int sum(int x ,int y){
            return x+y;
       }
      public static int sub(int x ,int y){
            return x-y;
       }

       public static int mul(int x ,int y){
            return x*y;
       }
        public static double div(int x ,int y){
            return x*y;

      }

      }
```

Figure (5.4) example

The figures next page show the test cases after generating.

```
package org;
import java.io.*;
import java.util.*;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class  findMaxTest {
@Test
public void  findMaxTesting (){
Calculation1 o = new Calculation1();
Scanner scan = new Scanner(System.in);
System.out.println("ENTER YOUR  VALUE");
int x =scan.nextInt();
System.out.println("ENTER YOUR  VALUE");
int y=scan.nextInt();
System.out.println("ENTER YOUR EXPECTED VALUE");
String expected = scan.next();
String actual =""+o.findMax(x , y) ;
assertEquals(actual ,expected);
}
}
```

Figure (5.5) Test case for find maximum number method

```
package org;
import java.io.*;
import java.util.*;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class  divTest {
@Test
public void  divTesting (){
Calculation1 o = new Calculation1();
Scanner scan = new Scanner(System.in);
System.out.println("ENTER YOUR  VALUE");
int x =scan.nextInt();
System.out.println("ENTER YOUR  VALUE");
int y=scan.nextInt();
System.out.println("ENTER YOUR EXPECTED VALUE");
String expected = scan.next();
String actual =""+o.div(x , y) ;
assertEquals(actual ,expected);
}
}
```

Figure (5.6) Test case for method that finds division result

```java
package org;
import java.io.*;
import java.util.*;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class  sumTest {
@Test
public void  sumTesting (){
Calculation1 o = new Calculation1();
Scanner scan = new Scanner(System.in);
System.out.println("ENTER YOUR  VALUE");
int x =scan.nextInt();
System.out.println("ENTER YOUR  VALUE");
int y=scan.nextInt();
System.out.println("ENTER YOUR EXPECTED VALUE");
String expected = scan.next();
String actual =""+o.sum(x , y) ;
assertEquals(actual ,expected);
}
}
```

Figure (5.7) Test case for method that finds summation result

```
package org;
import java.io.*;
import java.util.*;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class  subTest {
@Test
public void  subTesting (){
Calculation1 o = new Calculation1();
Scanner scan = new Scanner(System.in);
System.out.println("ENTER YOUR  VALUE");
int x =scan.nextInt();
System.out.println("ENTER YOUR  VALUE");
int y=scan.nextInt();
System.out.println("ENTER YOUR EXPECTED VALUE");
String expected = scan.next();
String actual =""+o.sub(x , y) ;
assertEquals(actual ,expected);
}
}
```

Figure (5.8) Test case for method that finds subtractresult

```
package org;
import java.io.*;
import java.util.*;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class  mulTest {
@Test
public void  mulTesting (){
Calculation1 o = new Calculation1();
Scanner scan = new Scanner(System.in);
System.out.println("ENTER YOUR  VALUE");
int x =scan.nextInt();
System.out.println("ENTER YOUR  VALUE");
int y=scan.nextInt();
System.out.println("ENTER YOUR EXPECTED VALUE");
String expected = scan.next();
String actual =""+o.mul(x , y) ;
assertEquals(actual ,expected);
}
}
```

Figure (5.9) Test case for method that finds multiple results

```
package org;
import java.io.*;
import java.util.*;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class  findMinTest {
@Test
public void  findMinTesting (){
Calculation1 o = new Calculation1();
Scanner scan = new Scanner(System.in);
System.out.println("ENTER YOUR  VALUE");
int x =scan.nextInt();
System.out.println("ENTER YOUR  VALUE");
int y=scan.nextInt();
System.out.println("ENTER YOUR EXPECTED VALUE");
String expected = scan.next();
String actual =""+o.findMin(x , y) ;
assertEquals(actual ,expected);
}
}
```

Figure (5.10) Test case for method that finds minimum number result

5.3 Conclusion

This chapter illustrated the process of generating test cases for methods that have returned value.

Getting source code in two ways either tester writes it or importing it from an external file, analyzes that code to capture all return type methods with testing values and save these methods, finally generating test cases for these methods.

Chapter Six

Results and Recommendation

6.1 Introduction

In this chapter, we show the results of our research and how our tool performed

6.2 Result

From the implementation, we extracted the flowing results

- The developed tool generates test cases for return type methods.
- That tool reduces the time of final delivery.
- Improve efficiency of the testing process (tests cases are generating in a few minutes).
- Increase reliability of testing process (test cases are generating correctly).
- Reduce consuming resource, time and cost of the testing process.
- Create file contained generated test cases.

6.3 Conclusion

The main idea of research is to help programmer test his program and reduce time, cost, risk, resources consuming and effort; there are many steps we did to develop this tool:

- Firstly we define the requirements of the system.
- Analyze the requirements and suggested the system.
- Designing the system.
- Finish the system development.

The system has been developed to generate test cases for methods that are returned values automatically.

6.4 Recommendation

- This research only generates test cases. These test cases can be used to execute the testing process.
- This research only generates test cases for programs developed in Java programming language; these test cases can be generated in multiprogramming language.

References

References

1. Software testing ,https://complextester.wordpress.com, 21/9/2017 01.00Pm
2. Manual testing, http://www.tutorialpoint.com/software_testing, 21/3/2017 01.00Pm
3. Jones and Bartlett Publishers, Software Engineering and Testing, 2010
4. GlenfordJ.Myers, The Art of software testing, 2nd Edition, 2004
5. Ilene Burnstien, Practical Software Testing, 2003
6. James Raumbough, Ivar Jacobson, Grady Booch, The Unified Modeling Language Reference Manual, 1st Printing, December 1998
7. Ed Burnette, Eclipse IDE Pocket Guide, 1st Edition, Aug 2005
8. Herbert Schildt, Java beginner,s guide, 6st Edition, 2014
9. Comparing the Effectiveness of Automatically Generated tests by Randoop , Jwalk, and µjava with JUnit Tests, NastassiaSmeets, July 19, 2009
10. Auto Test(A Tool for Automatic Test Case Generation in Spreadsheets , Robin Abraham, Martin Erwig , School of EECS
11. Jtest , http://char.tuiasi.ro/doace/www.parasoft.com/products/jtest/quick.htm, 21/9/2017 10.00Pm

Appendix

A. Business Process Modeling Notation(BPMN)

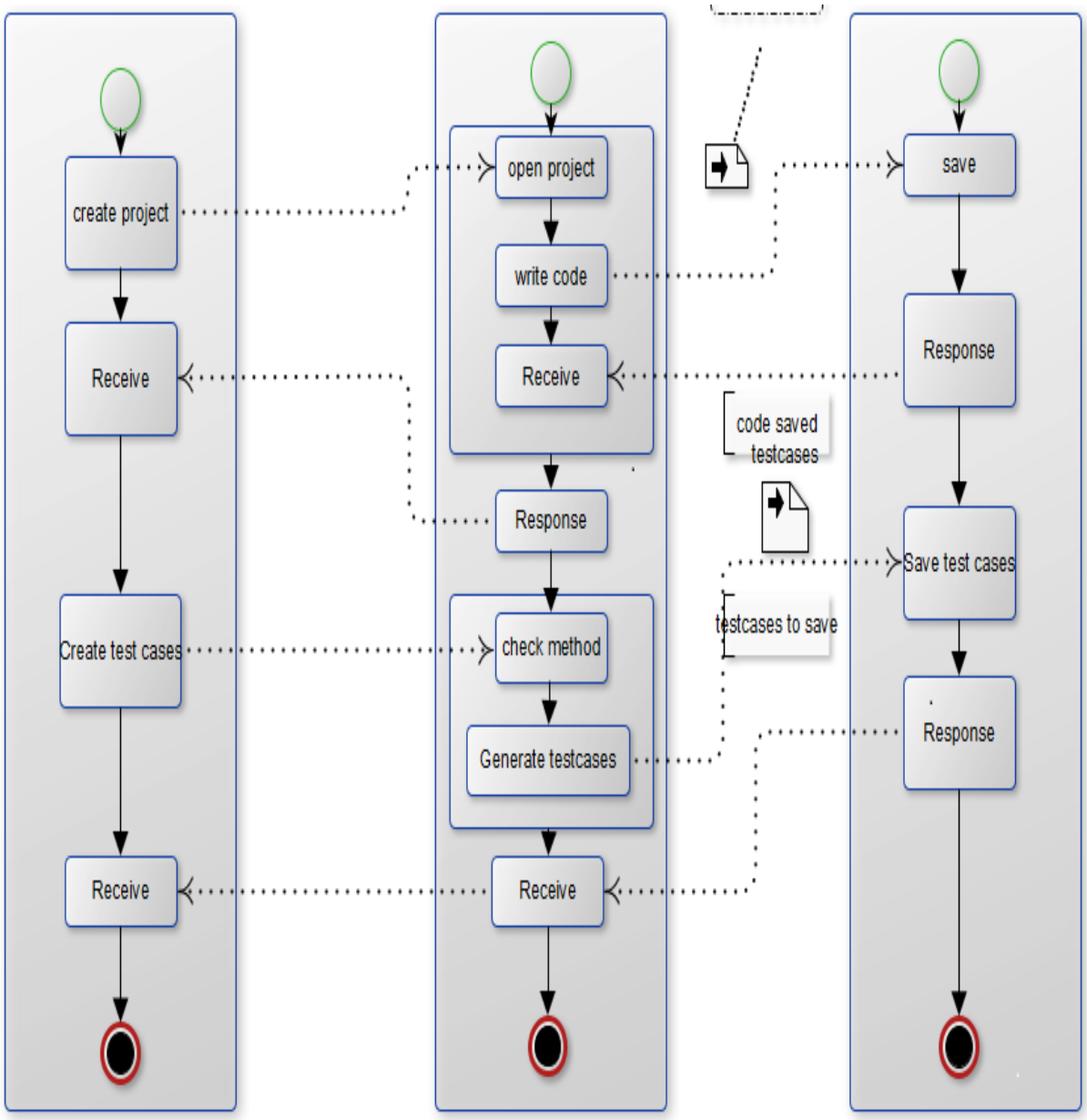

Figure (7.1) BPMN

B. Analyze code

The tool takes source code for testing and reads code line by line to find the methods that have returned values and save header of these methods in a file.

```
import java.io.*;
importjava.util.*;
public class Analyzing
{
    static Scanner x ;
    staticPrintWriter y ;
    static String a ,q , s ;
    public static void openFile()
    {
        try
        {
            x = new Scanner(new File("D://MGM.txt"));
        }
        catch(Exception e)
        {
```

```
                    System.out.print("err");
                }
        }
public  static void readFile()
 {
        try
        {
         y = new PrintWriter("D://MGMx.txt");
while(x.hasNext())
    {
         a = x.nextLine();
        StringTokenizerst=new StringTokenizer(a, " (),");
        while(st.hasMoreTokens())
         {
                String key = st.nextToken();
                    if(key.equals("public")  ||  key.equals("private")  ||
key.equals("protected"))
                    {
                        key = st.nextToken();
```

```
                    if(key.equals("static"))|| key.equals(""))
                {
                        key = st.nextToken();
                    }
                        if(key.equals("int") || key.equals("float") ||
key.equals("double") || key.equals("char") || key.equals("String") ||
key.equals("boolean") )
                        {
                            key = st.nextToken();
                            int last = a.indexOf(')');
            String methodHeader = a.substring(0, last+1);
                y.println("The function is : "+methodHeader);
                    System.out.println("The TestCase is  : "+methodHeader);
                        }
                }
        }
   }
y.close();
   }
            catch(Exception e)
```

```
                {
        System.out.println(e);
      }
  }
public static void closeFile()
  {
                x.close();
  }
public static void main (String[]args)
  {
                openFile();
                readFile();
                closeFile();
  }
}
```