A Novel Framework For End-To-End Automation Testing

Praveen M Bidarakundi, Raghavendra Prasad S.G

Abstract: - Often implementation of the program will change .Implementations are changed to reduce running time and/or to reduce memory consumption (space complexity) of the program. Often there is need to test the two version of the software, one current and another newer version. Newer version will be having some extra methods/functions, but the remaining methods/functions will be same as that of current version. We need to make sure that these methods of current version have not been affected by the changes done in new version. (Regression Testing), and also often the methods will be refactored to different prototypes/signatures to offer abstraction. These new prototyped methods (signature changed methods) will in-tern invoke the previous method (before to new prototyping), i.e. newer prototyped methods are just wrappers around the previous methods. For instance APIs are wrapped around by corresponding methods .In this situation, it becomes important to test the newly prototyped methods that are wrappers around the old methods/APIs, as we need to verify the correct bindings/mappings of the older and newly prototyped methods. Usually developers write the unit tests to test their logic. But the testers cannot write them as tester lacks the knowledge of logic implemented, and tester may not have any knowledge of coding, but tester knows what each method does and what is it's expected behavior/return type. Thus we need to offer new way to test each method. We propose a novel framework, which addresses these important issues. Framework takes three input parameters namely, class to be tested, variable initialization values (test data), and expected results. From this information, framework automatically builds test driver class at runtime; on the fly on running the framework. Test driver class is used to test the class under test. This test driver class is compiled and executed to get the actual results for class under test. These generated actual results are compared with expected result to find methods different behaviors. Methods whose actual result is not matching with the expected result, then this implies that methods have different behavior, thus the test is failure, and else test is successful.

Index Terms: - Class Testing, Differential Unit testing, End-to-End Automation of Unit Test, Framework for unit test, Object Oriented programs testing, Unit Testing, Integration Testing.

————————————◆————————————

1 Introduction

Software developers often manipulate (slightly) different versions of the same software. The most common scenario is changing software systems by evolving them from one version to another. Another scenario is having multiple implementations of the same interface, feature, or functionality to reduce running time and memory space requirement. For example, we may have multiple C compilers that handle ANSI C code [3]. Yet another scenario is in the context of mutation testing [4]: intentionally making slight changes to a program to create mutant versions. In all these scenarios with multiple versions of programs, the versions can have different functional behaviors. A typical task then is to determine how the behaviors of one version differs from (or are the same as) the behaviors of a different version[2]. In such tasks, testers would like to generate test inputs that show the behavioral differences between the two versions (producing different outputs for the same inputs), if any differences exist. This type of testing is called differential testing [3]. Regression testing and mutation testing are examples of differential testing. Researchers have developed approaches for differential testing of software at the system level, including testing of C compilers [3], flash file system software [5], and grammar driven functionality [6]. We focus on differential unit testing, where differential testing is applied on a program unit. Experiment-driver Integration testing needs to be addressed for object oriented software [1]. This paper attempts to provide integration testing approach. Specifically, we focus on object-oriented programs, where a unit can be a class or a set of classes. Object-oriented unit tests for a class consist of sequences of method invocations. Behavior of an invocation depends on the method's arguments and the state of the receiver at the beginning of the invocation. Behavior of an invocation can often be observed through the method's return values and the state of the receiver at the end of the invocation. Differential unit testing of object oriented programs thus requires (1) execution of pairs of corresponding methods from the two versions by providing same inputs and (2) comparison of the outputs of the resulting method executions. We propose a framework that can be used in differential unit testing of object-oriented programs, for checking the implementation errors and java bindings between newer methods (newer prototype) and existing methods (before newer prototype). Framework takes three input parameters, namely class to be tested, variable initialization values, expected result values. From this information, framework automatically builds test driver class at runtime, on the fly. Methods can receive the same inputs, and framework compares their outputs. If the outputs are different, framework reports to testers the different behaviors of the two versions. For Differential unit testing, we will execute the framework against the newer version to get actual results and these actual results are compared with expected results of the current version, to find variation/differences. We have implemented framework in a tool that operates on Java classes. This paper introduces the framework, presents our

- *Praveen M Bidarakundi is currently pursuing masters degree program in information technology engineering in Visvesvaraya Technological University, Karnataka, India, PH-+918971470267 E-mail: pm.bidarakundi@mail.com*
- *Raghavendra Prasad S.G is currently an Assistant Professor, at Dept. of ISE R.V.C.E Bangalore, India, Country, and PH-+919845003187.*
- *E-mail: sgrp.vishnu@gmail.com*

implementation, discusses results obtained, and proposes differential test generation based on the code instrumented by framework.

2 EXAMPLE

To illustrate the framework, we use a binary search tree class BST that implements a set of comparable elements, shown in Figure 3. The class My_Input in the figure is the comparable type of elements (e.g., integers in this example) stored in the stack. Each tree has a pointer to the root node and a field size that denotes the number of elements in the tree. Each node has an element and pointers that maps to right child and left child. The class provides implements for the standard set operations: insert, remove, and contains. The class also has a constructor that creates an empty tree. In Fig.4, we have binary search tree class WrBST, this implement a set of comparable elements, as shown in Figure 4. The class also implements the standard set operations: insert, remove, and contains. These set operations intern invokes the operations/methods of the class BST(fig 3),so there needs to be proper one to one mapping of the method of class WrBST(Fig.4) to method of class BST(Fig.3).These are called java bindings(binding of a method to another method). These mappings/bindings have to be correct, and then only the correct operation/result is obtained. If there is any wrong/misconfigured bindings ,the framework will detect the same and display them, framework is be able to do this as the method's return values of current(class BST) and newer version(class WrBST) will differ in case of wrong bindings. In this example, we consider WrBST (newer version) to be the class under test (fig 4), and its current version to be the reference class BST(fig 3).

```
public class My_Input implements Comparable {

private int o;

public My_Input(int i) { o = i; }

public boolean equals(Object that) {

if (!(that instanceof My_Input)) return false;

return (o == ((My_Input)that).o);

}}

class BST implements Set {

Node root;

int size;

static class Node {

My_Input value;

Node left;

Node right;

}

public BST() { ... }

public BST insert(My_Input m) { ... }

public BST remove(My_Input m) { ... }

public boolean contains(My_Input m) { ... }

}
```

Figure 3 A set operations implemented as a binary search tree (BST) – Current Version

Here we can see that newer version class has different implementation than that of Current Version. Often implementations may be changed, to increase performance (running time and space) of the programs, to have one more layer of abstraction to provide interface to user. Here Newer Version (Fig 4) is actually wrapper around the Current Version (Fig 3).We need to test whether bindings between Newer Class methods and the Current class methods are proper i.e. to check Newer class methods are appropriately calling the Current Version methods and also the methods should provide same return value as that of current version, irrespective of the implementation. Framework will test newer version to see all the methods of this class provide the same output as that of Current Version. User provides class to be tested (Newer Class), method input parameter values i.e. test data (Variable initialization file), and Expected result (results of the Current Version's methods execution or new results) to the framework.

```
public class WrBST extends ReferenceBST {

public WrBST() { super(); }

public boolean equals(Object t) {

if (!(t instanceof BST))

return false;

BST b = (BST)t;

if (size != b.size) return false;

if (!root.equals(b.root)) return false;

return true;

}

public BST insert(BST c, My_Input m) {

return c.insert(m);

}

public BST remove(BST c, My_Input m) {

return c.remove(m);

}
```

```
   public boolean contains(BST c, My_Input m, boolean r) {

   return c.contains(m);

   }}
```

Figure 4. A set operations implemented as a binary search tree (BST)-Newer Version.

3 Framework

Framework will test all the method of a class, user/Tester need not to have any coding knowledge. User/Tester needs to provide three parameters for the framework. These are input parameters as class under Test, Variable Initialization file and expected result.

3.1 High Level Design

As shown in above Fig.5, java source file is converted to xml file with required class information in it and then framework will parse the xml to get relevant information for creating Test driver class at runtime. This Test driver class is generated at runtime of the framework automatically on the fly. Once the test driver class is automatically generated, then it is compiled, executed and the actual results are populated in result xml and this actual result is compared with expected result to produce success/ failure appropriately.

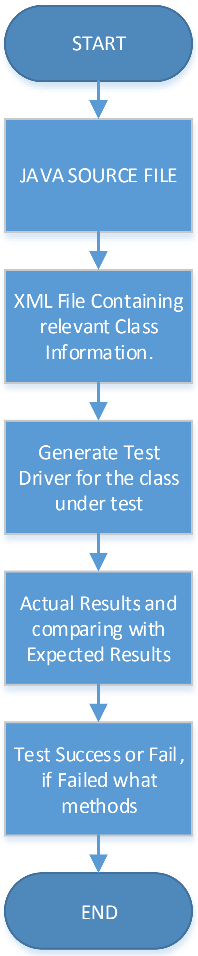

START
JAVA SOURCE FILE
XML File Containing relevant Class Information.
Generate Test Driver for the class under test
Actual Results and comparing with Expected Results
Test Success or Fail, if Failed what methods
END

Fig.5

If there is no source file available, then we can convert class file to source file, with the help of java decompiles. Once, if we get source file, we can give it to framework as input. Framework can be executed step by step also, as we can first give only source file as input, the framework converts source file to xml file and halts and asks for the variable initialization file. To increase ease of use of the framework, framework auto generates variable initialization file in appropriate xml format, in which user has to just change the values. Once we give this newly edited variable initialization file (test data) as input, the framework proceeds further for execution and builds the test driver for testing the class under test. Then it compiles and executes the test driver generated and produces actual results. Then framework once again halts and asks the user to provide expected result file. Framework auto generates the expected result file in the correct format; this is done to increase the ease of use of the framework. Tester just needs to edit the values in the file. Then framework proceeds for the further execution and compares the actual results with the expected results and produces test as failure, if expected and actual values does not match. If test is failed, then it also displays what methods have failed and also what was expected value and what is actually produced value. If the test has passed, then it is displayed as test successful.

3.2 Capabilities of the framework

Framework has faced various challenges like Passing Complex/user built-in data type as input parameter to a method, comparing two complex objects for equality, output of one method as input to another method, proper sequence of methods invocation, and handling of inner classes/interface. These challenges and how these are handled by the framework is elaborated in this section.

1) Passing Complex/user built-in data type as input parameter to method :

If a method takes a complex/user built-in data type variable as input, then frameworks handles this by creating the object of the complex data type/user built-in data type, it parses the xml(variable initialization xml file) and finds that it needs to create an object and also it automatically imports the required packages without manual intervention. This automatic import is done by creating index of all classes internally by the framework.

2) Comparing two complex objects for equality:

If the output of the method is complex/user built-in object, then we need to compare this object with the expected objects. This is handled by the framework by creating the “equals” method and passing the two objects as input parameter and the method returns true if both the objects are equal, else it returns false.

3) Output of one method as input to another method:

If a method’s output parameter is required to provide as an input parameter of another method, then the framework will recognize this by examining the xml variable initialization file and appropriately invokes the first method and assigns its output value to a variable and this same variable value is supplied as input variable to the second method. This solution is implemented in the framework, so tester/user need not to worry about handling it and also care has been taken as to invoke methods in correct order.

4) *Proper sequence of methods invocation:*
In a class, there will be getters and setter methods, and the setters should be invoked first and then only getters should be invoked, because unless we set the values, we will not be able to get the values. Frameworks parses the xml file for identifying setters and getters methods, and first arranges setters methods and invokes them, later getters methods are invoked.

5) *Handling of inner classes/interface:*
Inner class's methods have to be properly invoked by creating the object of the outer class, but the inner class methods can call the outer class methods as per java specifications. Thus we need to invoke the inner class methods using the object of outer class. This is handled by the framework by creating the object of the outer class and invoking the methods of inner class appropriately, as framework gets inner classes and inner class methods information from the xml.

6) *Automatic imports code generation*
There will be user built in/external library classes in the auto generated test driver, these classes will be present in the different packages and these are bundled in jars. Jar will be available in java build path, but framework need to import the corresponding packages in test driver class. This is handled in the framework by creating the index to the all classes and packages. Thus frameworks will automatically import the corresponding packages required for classes. These capabilities of the framework make it truly end-to-end automation i.e. framework is executed without any manual intervention.

4 Test Results

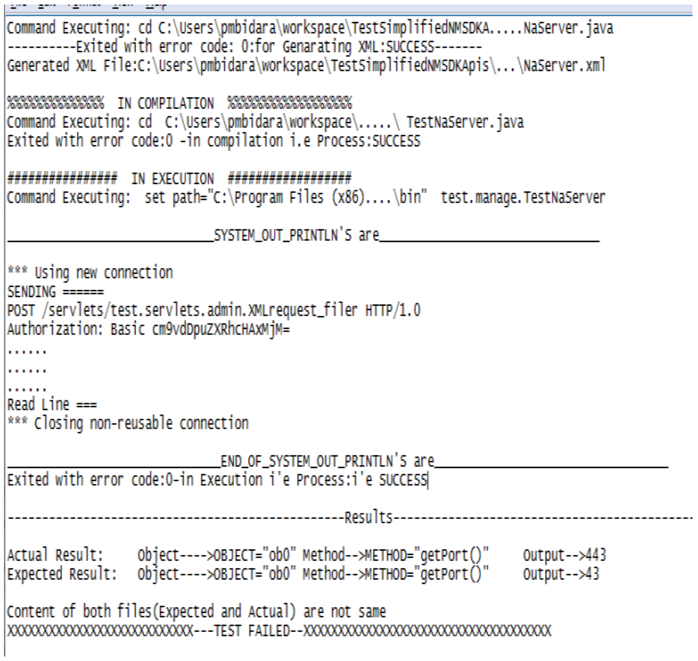
The framework is tested with classes of all types, i.e. classes having all possible conditions and constraints. This section provides the results of the framework execution against these classes and evaluates the framework.

1. *Sample console output from the execution of the framework.*

Framework is tested against the classes and the results are recorded. Below is the console output screen shoot of the framework execution. In this screen shot Fig 6, we can find that framework has followed step by step execution and as expected and actual values are different, the test has failed. We can also see that framework produces sensible system outs that will help in understanding the flow of the framework. Here NaServer, is a class, that contains the methods to connect to the server and execute the methods/APIs on the server. These methods/APIs will return actual values after execution; the same are recorded and compared with expected results.

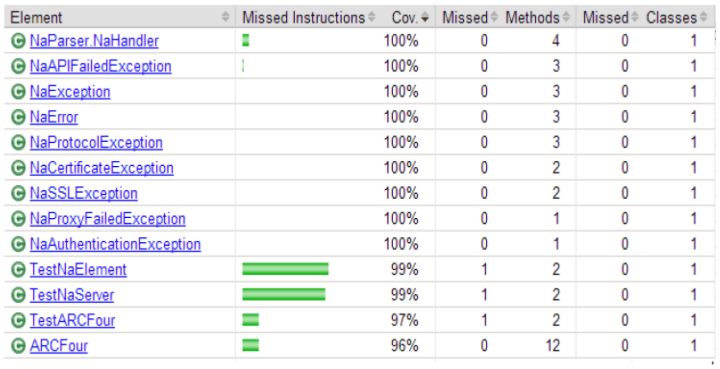
2. *Code Coverage from the framework*

We have used EclEmma code coverage tool for finding the code coverage of the classes executed using the framework. To keep this paper concise and to the point, here we have just shown code coverage of the some classes, but these classes will cover all the scenarios and constraints. As framework is able to test each method of the class, we will get 100% of code coverage, provided we supply correct variable initialization values (test data) that will test/cover all the instruction of a method. We will be also able to find what code has not executed and we can change the variable initialization values to execute this part of the code that has been missed in first time execution. In Fig 7, we can see that elements are the classes and the coverage of the instructions is almost 100%, and all the methods in each class have been executed, except private methods, as private methods are not to be executed, as these are not exposed to user/tester or external world. From analysing this result we can conclude that that framework is powerful enough to test all the methods of class, and provide almost 100% code covearage, without tester ever having to know any programming language and coding experience. The evaluation of this framework shows that framework has greater code coverage and method coverage than that of other frameworks.



```
Command Executing: cd C:\Users\pmbidara\workspace\TestSimplifiedNMSDKA.....NaServer.java
----------Exited with error code: 0:for Genarating XML:SUCCESS-------
Generated XML File:C:\Users\pmbidara\workspace\TestSimplifiedNMSDKApis\...\NaServer.xml

%%%%%%%%%%%%%  IN COMPILATION  %%%%%%%%%%%%%%%%%
Command Executing: cd  C:\Users\pmbidara\workspace\.....\ TestNaServer.java
Exited with error code:0 -in compilation i.e Process:SUCCESS

################  IN EXECUTION  ##################
Command Executing:  set path="C:\Program Files (x86)....\bin"  test.manage.TestNaServer

______________________________SYSTEM_OUT_PRINTLN'S are________________________________

*** Using new connection
SENDING ======
POST /servlets/test.servlets.admin.XMLrequest_filer HTTP/1.0
Authorization: Basic cm9vdDpuZXRhcHAxMjM=
......
......
......
Read Line ===
*** Closing non-reusable connection

_______________________________END_OF_SYSTEM_OUT_PRINTLN'S are____________________________
Exited with error code:0-in Execution i'e Process:i'e SUCCESS

-------------------------------------------------Results------------------------------------------

Actual Result:    Object---->OBJECT="ob0" Method-->METHOD="getPort()"    Output-->443
Expected Result:  Object---->OBJECT="ob0" Method-->METHOD="getPort()"    Output-->43

Content of both files(Expected and Actual) are not same
XXXXXXXXXXXXXXXXXXXXXXXXXXX---TEST FAILED--XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Fig 6 Console output from the execution of the framework



Element	Missed Instructions	Cov.	Missed	Methods	Missed	Classes
NaParser.NaHandler	🟩	100%	0	4	0	1
NaAPIFailedException	🟩	100%	0	3	0	1
NaException		100%	0	3	0	1
NaError		100%	0	3	0	1
NaProtocolException		100%	0	3	0	1
NaCertificateException		100%	0	2	0	1
NaSSLException		100%	0	2	0	1
NaProxyFailedException		100%	0	1	0	1
NaAuthenticationException		100%	0	1	0	1
TestNaElement	🟩	99%	1	2	0	1
TestNaServer	🟩	99%	1	2	0	1
TestARCFour	🟩	97%	1	2	0	1
ARCFour	🟩	96%	0	12	0	1

Fig.7 code coverage from the framework.

5 Conclusions

Framework provides end-to-end testing of any java class, tester need not know the logic implemented in the methods and also the need not have any coding experience to test methods using this framework. Framework can be easily used by everyone either novice or experienced. This framework works for all object oriented languages. By this result we can conclude that that framework is powerful enough to test all the methods of class. The evaluation of this framework shows that framework has greater code coverage and method coverage than that of any other framework. Thus it provides the novel approach towards unit testing, class testing and also integration testing (tests for proper binding/mappings between methods of two classes). Further enhancement would be to automate the workflow scenarios i.e. executing only certain methods in certain sequence, to achieve a workflow test case.

References

[1] Yvan Labiche, Integration Testing Object-Oriented Software System: An Experimental-Driven Research Approach, Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference,IEEE CCECE 2011 – 000652. 8-11 May 2011, Page(s): 000652 - 000655.

[2] Tao Xie,Kunal Taneja,Shreyas Kale and Darko Marinov,Towards a Framework for Differential Unit Testing of Object-Oriented Programs,Second International Workshop on Automation of Software Test (AST'07)-0-7695-2971-2/07,IEEE Computer Society, 2007.

[3] W. M. McKeeman. Differential testing for software. Digital Technical
Journal of Digital Equipment Corporation,10(1):100–107, 1998.

[4] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data
selection: Help for the practicing programmer. IEEE Computer,
11(4):34–41, April 1978.

[5] A. Groce, G. Holzmann, and R. Joshi. Randomized differential testing as a prelude to formal verification. In Proc. 29th International Conference on Software Engineering, 2007

[6] R. L¨ammel andW. Schulte. Controllable combinatorial coverage in grammar-based testing. In Proc. 18th IFIP International Conference on Testing Communicating Systems,pages 19–38, 2006.