TAPE
Test Code Adaptation Plug-in for Eclipse

Lehmia Kiran
Department of Computer Sciences
National University of Computer and Emerging Sciences,
Lahore, Pakistan
lehmia.kiran@nu.edu.pk

Fakhar Lodhi
Department of Computer Sciences
National University of Computer and Emerging Sciences,
Lahore, Pakistan
fakhar.lodhi@nu.edu.pk

Wafa Basit
Department of Computer Sciences
National University of Computer and Emerging Sciences,
Lahore, Pakistan
wafa.basit@nu.edu.pk

ABSTRACT

Refactoring improves the design of software and makes it easier to maintain by eliminating code smells. As manual refactoring is often error-prone and time-consuming, automated refactoring is considered a better choice. Refactoring tools are available in almost all major object oriented programming languages like java, C++, C# etc and provide varying degree of automation. The problem with these tools is that they do not support adaptation of unit tests of the refactored code resulting in invalid unit tests. Unit tests are the only safety net available to developers to verify the system behavior after refactoring. Once these tests get broken there is no way to describe whether the system preserved its behavior or not. In this paper we provide technical details of TAPE (Test code Adaptation Plug-in for Eclipse). This tool not only automates a few commonly used refactorings but also supports adaptation of unit tests affected by the refactoring process. Using this tool the developers can concentrate more on code development activities instead of resolving consistency issues between code and unit tests.

Keywords- Refactoring; Unit testing; Adaptation, maintenance

1. INTRODUCTION

Refactoring is the process to change the code, making it easier to understand and modify [6]. But modifications in code can induce new bugs in it and can affect various locations throughout a program. Therefore, it is important to verify syntactic and semantic correctness of the refactored code. Unit testing is the key method used to validate the code after refactoring [16]. Unit test cases in Object Oriented Paradigm are the programs that test classes [12]. Each program is composed of units and every unit is set of functions and these units are tested separately and test the code at lowest level of granularity [13].

Different frameworks for unit testing are available [1, 23, and 24]. JUnit [1] is a java plug-in which is used to execute the test cases in the background and informs developers about the failure or success of the test cases. Eclipse [17] is a development environment which is used to edit, compile and run the java code. When JUnit is embedded in eclipse, test cases can also run on java code. But JUnit does not automate the test case development process and developers have to write the test cases by themselves, this problem can be solved by using MoreUnit[5]. MoreUnit [5] is an eclipse plug-in that helps in generating automated JUnit tests. The refactoring plugin for Eclipse does not currently use MoreUnit [5]. Instead of reinventing the wheel, we have reused MoreUnit plugin along the refactoring API to achieve test code adaptation.

Whenever refactoring is applied on code, many test cases fail due to inconsistency with source code. The significant amount of effort is required to manually align the testing suite on every single refactoring [15]. The process of refactoring is automated by many tools [17, 20, 21 and 22]. Eclipse supports automated refactoring and provides separate API to perform refactoring. But this API only focuses on source and does not adapt test cases, making both inconsistent.

Cost and effort of adapting test cases is very high. Complication of the adaptation increases with complexity of refactoring applied. In some cases adaptation becomes very complex and test cases have to be thrown away and new test cases have to be developed and results in a very expensive process [8,19]. The solution to this problem is presented in this paper.

TAPE (Test Code Adaptation Plug-in for Eclipse) has been especially designed to improve the process of refactoring. It is an Eclipse plug-in which adapts test cases making it consistent and semantically aligned with the code after applying refactoring. TAPE reduces the cost of maintenance of test cases because it makes source code and test cases synchronized. Both source code and the test code come under the ownership of a developer. If there is change in the source code; the associated test code should also be adapted accordingly [18]. TAPE can assist developers by drastically reducing time and effort of manually resolving the consistency issues between code and unit tests.

This paper is organized as: Section 2 discusses currently available refactoring tools and the types of refactorings they support. Section 3 contains architecture and implementation details of TAPE, including the technical details on refactoring API for eclipse, “JUnit" and "MoreUnit" are briefly illustrated. In Section 4 we draw conclusions and future work.

2. LITERATURE REVIEW

Refactoring is a means to improve the quality of existing code. The main idea behind refactoring is to change the code in small steps, while maintaining its external behavior [6]. Preserving code behavior means that functionality of the program must be
unchanged after refactoring. Refactoring can be done manually but it’s very tedious and error-prone. Significant amount of work can be reduced and efficiency can be increased by automating the process [10]. Fowler’s refactoring guidelines provide a road map to perform 72 refactoring [6] but these guidelines are only applicable to production code. To incorporate the unit test adaptation in refactoring Fowler’s guidelines for refactoring are extended by Basit et. al [8] for Move Field [6] refactoring. The guidelines for the refactorings supported in TAPE have been extended in the similar manner.

Basit et al. [8] have categorized refactoring guidelines into three types based on their affect on test code and production code.

**Type I:** Refactoring guidelines are complete to adapt both production and test code. For Example: Extract Method, Inline temp, Replace temp with query etc

**Type II:** Refactoring guidelines are complete to refactor the production code correctly but break or need extension to the test code. For Example: Move Method, Inline Method, Introduce Foreign Method etc.

**Type III:** Refactoring guidelines that are not complete for both restructuring production and test code. For Example: pull up method; pull up constructor body, Move Field etc.

In this paper the impact of refactoring on unit tests is considered only therefore the types II and III have been merged. Hence, two broad classes of refactoring guidelines are:

A: Refactorings that break test code

B: Refactorings that do not break test code

Automated tools are available for refactoring in almost all major object oriented programming languages like Java, C++, C# etc. IntelliJJIDEA, Eclipse and Net Beans are refactoring tool for java, CppRefactory and Xrefactory supports refactoring in C++ and to perform refactoring in C#, C# Refactory Tool and C# Refactory are available. Table 1 shows the total number of refactorings in different tools and the number of refactorings belonging to classes A and B.

<table>
<thead>
<tr>
<th>Tool name</th>
<th>Total</th>
<th>Fowlers’</th>
<th>Class A</th>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netbeans</td>
<td>14</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Eclipse</td>
<td>21</td>
<td>13</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>32</td>
<td>18</td>
<td>14</td>
<td>4</td>
</tr>
<tr>
<td>Cpp Refactory</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Visual Studio .NET(C# refactoring Tool)</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

Netbeans offers total 14 refactorings. And out of 14, 8 refactorings are defined by Fowler. 4 refactorings belong to class A and 4 refactorings belong to class B. These figures tell us that only 4 refactorings are those which do not break test code and it is very less in number. (In this paper we do not discuss the refactorings implemented by tools other than that of Fowler’s.)

Eclipse supports total 21 refactorings, out of which 13 are fowlers’ refactorings. 9 refactorings belong to class A and 4 refactorings belongs to class B. So, only 4 refactorings are those which do not require unit test adaptation. Similarly, in IntelliJJIDEA 18 refactorings are those whose guidelines are provided by Fowler and only 4 refactorings belongs to Class B and remaining 14 belongs to class A, which are very few in number. In CppRefactory there is no refactoring which belongs to class B and in visual Studio.Net only one such refactoring exists which does not break unit test after applying refactoring.

We can clearly see that all tools mentioned provide a very limited number of refactorings. This is because behavior preservation and consistency issues of refactoring are many fold. There has been very little work done on addressing and exploring the effects of refactoring on the artifacts and other components of the software system. In [8], the current literature on impact of refactoring on production and unit tests has been reported in detail. The major focus of this paper is on providing technical details of TAPE.

3. TAPE

Eclipse is one of the most commonly used IDE’s for software development. Considering this fact Eclipse refactoring plug-in has been used as a baseline for TAPE. Following is a short overview of how refactoring is done in Eclipse.

3.1 Refactoring in Eclipse

Eclipse supports automated refactoring and provides an API for implementing refactoring that can be applied to any of its workspace elements (e.g. class or method). Eclipse refactoring is implemented for several languages e.g. Java and C++; but only refactoring of Java code [7] is highlighted in this paper. Figure1 shows available refactorings in Eclipse IDE.

![Figure 1: Refactoring Menu in Eclipse](image)
TAPE is initiated using the menu provided by Eclipse, the refactoring process is done in the same manner as the refactoring API but at the back end unit test adaptation also takes place. This is how; no additional learning time is required at the developer’s end.

3.2 **JUnit**

Many testing frameworks are available that automate many aspects of test creation and maintenance [1, 23, 24]. An example of such a framework is JUnit [1].

To run the test cases in background JUnit can be used, but problem of writing test cases remains the same as JUnit only runs the test cases and gives output about failure or success. To resolve the issue of writing test cases MoreUnit an existing plug-in for Eclipse is brought into use by TAPE [5].

3.3 **MoreUnit**

MoreUnit is an Eclipse plug-in that helps in generating automated JUnit tests. MoreUnit supports the generation of test cases, associates code and test cases. Classes which have associated unit tests are decorated with green marker and it also provides the facility to navigate between code and corresponding test using short cut [5].

TAPE uses MoreUnit to make an association between source code and test code. Classes which have associated test cases, have green marker on with icon of the class as shown in figure 2. MoreUnit also helps in searching corresponding test cases or test methods of source code when refactoring is applied.

3.4 **TAPE**

Adaptation of unit tests with refactored code means the process of eliminating syntactic and semantic errors from the unit tests after applying refactoring; also the quality of test code and capability to test should remain the same as before refactoring of source code [8].

Unit tests can also be considered as client but these are specialized clients because these are closely associated with source code. Unit tests are directly affected due to any refactoring or restructuring in source code [8] and in certain cases it may require different transformational procedure from that of ordinary clients. Like in ‘move method’ refactoring, when a method is moved, its test methods should also be physically moved to the target’s test class [8]. On the contrary only references are updated for simple client code. The examples of unit test adaptation for move method, inline method and extract class are illustrated in [8].

TAPE has been especially designed to improve the process of refactoring. It is an Eclipse java plug-in which is used to assist in refactoring the code and adapting test cases in it. Currently four refactorings belonging to class A are supported in TAPE.

1. Move method
2. Inline method
3. Pull Up method
4. Rename method

In this paper, move method and pull up method are discussed in detail.

3.5 **TAPE Overall Architecture**

Architecture diagram of “TAPE” is shown in Figure 3, it describes how different components involved in the TAPE are associated with each other. As described above, Eclipse refactoring plug-ins are used to perform refactoring on the code (Eclipse plug-ins are associated with Eclipse workspace). MoreUnit generates unit test cases and builds an association between tests and corresponding classes. The inputs to TAPE are code and its corresponding test cases. TAPE adapts test cases thus making both code and its corresponding test cases synchronous.

![Figure 2: Package explorer of eclipse: source classes and corresponding testcases](image)

![Figure 3: Overall Architecture Diagram](image)
3.5.1 Basic Elements of TAPE

Figures 3 and 4 illustrate how TAPE fits into existing elements of Eclipse refactoring plug-in [3]. User starts refactoring by using TAPE GUI. The refactoring implementer gets the requirements for refactoring as specified by the user and then sends this information to the refactoring processor. Based on the requirements provided by refactoring implementer, the refactoring processor determines what changes are required in the code. Textual changes are performed in workspace resources like the java project or java files to reflect the changes made in underlying code. After that, selected element is sent to MoreUnit plug-in; it searches for the test cases which are associated with source code. Test cases are sent to TAPE where corrective actions are taken to make test code and source code consistent. Output of TAPE plug-in is the adapted test cases.

3.5.2 Life Cycle of TAPE

Figure 5 illustrates the lifecycle of TAPE which is initiated when eclipse refactoring UI forwards the client’s request to the eclipse refactoring plug-in to initialize the refactoring. In this initiation phase of refactoring, initial conditions for refactoring are being checked and if some information is missing, then it gathers required details and checks for final conditions. If all conditions are satisfied, corresponding testcases of the code are gathered. After that refactoring is applied to the source code and respective corrective actions are taken on unit tests to make them synchronous with source code.

3.5.3 Object Model

Figure 6 exhibits a high level object model of TAPE. “Type Façade” and “Class Type facade” are the classes of MoreUnit which help in parsing eclipse workspace to search for test cases and test methods. Interface class serves as the bridge between GUI and wrapper classes. ConditionChecker contains two abstract methods checkInitialCondition and checkFinalCondition. These methods are overridden in its subclasses: “MoveMethodConditionChecker”, “InlineMethodConditionChecker”, “PullUpMethodConditionChecker” and “RenameMethodConditionChecker”. If the conditions are satisfied the textual change is made. Change class which contains abstract method createChange and in each of its subclasses it is overridden to create according to the selected refactoring
3.5.4 Implementation details of TAPE

In this sub section the process of refactoring along with test code adaptation is elaborated. Few refactorings’ implementation are illustrated separately. As mentioned above, moreUnit plug-in helps to make an association between source and its test-cases. Using this association the corresponding test cases and test methods are searched and then the process of refactoring is applied on both source and test method.

3.5.4.1 Move Method Refactoring:

In Move Method refactoring, a method is physically moved from the source class to the target class and its references are updated accordingly. So, any method which was previously referenced to source class method will be referenced to destination class.

In figure 7, there are two classes student and course and there is a method registerCourse (course) in student class. Both classes have their test classes, named studentTest and courseTest. registerCourse (course) frequently uses the attributes of course class so it is moved to course class.

In original eclipse refactoring plug-in, when refactoring is applied only registerStudent(course) is moved to course class but testRegisterCourse() is not moved to courseTest, so unit tests and production code classes become inconsistent. On other hand, when refactoring is initiated through TAPE, registerCourse (course) is moved to course class, and its corresponding test method, testRegisterCourse(), is also moved to CourseTest. UML diagram after refactoring is shown in figure 8.

In ‘Move Method’ refactoring, the source class must have association with destination class and this association can be made by passing destination class object as parameter in the Eclipse refactoring plug-in.

If this pre condition is not satisfied, Eclipse refactoring will not allow method to move, but when one writes test method for a
given method, it cannot pass any parameter. So, for moving test method refactoring Eclipse refactoring plug-in is not used.

The flow of move refactoring is as follows (figure 9): firstly the method which is to be moved is selected from GUI and its corresponding test method is searched by ClassTypefacade. This information is sent to MoveMethodProcessor to check preconditions of refactoring. If these conditions satisfy test method is physically moved to destination class using “move” function which is defined in IMethod (IMethod represents a method in java Model of Eclipse) class [2]. Up till now client code updation is not incorporated in TAPE.

3.5.4.2 Pull Up Method Refactoring:
Identical methods in subclasses are moved to super class this type of refactoring is known as pull up method refactoring [25].
For example, there is a super class named “Animal” and there are two sub classes “landAnimal” and “MarineAnimal”. Both classes have method “getName()” with same method body. Each class has its own unit test and its corresponding test methods as shown in Figure 10.

As “getName” method has same method body in both classes so it should be moved up to Animal class. After applying refactoring through TAPE, both methods from sub classes are moved up to person class and also corresponding test method “testgetName ()” is moved to unit test of Animal which is “AnimalTest” as shown in figure 11. Thus, both source and test cases are consistent.
Like all other refactorings, PullUpMethod refactoring is started by selecting super class, sub classes and the method which is to be pulled up from the GUI. The major class which is involved in Pull up method is pull up method processor, but before applying actual refactoring of Eclipse plug-in some additional checks on source code are added. These include:

1. Does the inheritance hierarchy exist between the classes?
2. If there are more than one child classes, only then pull up method refactoring is applied.
3. Body of the methods, in all sub classes, that are to be pulled up should be same.

If these conditions are satisfied, then test methods are being searched by “ClassTypeFacade” and cycle of refactoring that is discussed previously in section 3 is applied to both source and testcases, making testcases and source synchronized.

4. Conclusion

TAPE (Test Code adaptation Plug-in for Eclipse) helps in synchronizing code and its corresponding test cases after refactoring. This automated tool of refactoring saves developer's time to make code and test cases consistent in refactoring process. Currently, only impact of refactoring on unit tests is taken into
account and client code is not incorporated. This tool can be extended to cater client code also.

5. References