MuClipse Requirements Specification

v0.4
12/6/2006

Project Team:
Ben Smith

Document Author(s):
Ben Smith

Project Sponsor:
Dr. Laurie Williams

1. Introduction

There are already several open source Java mutation (JMutation) systems. MuJava and Jester are two examples of such tools. MuJava, in particular, provides both class-level and method-level mutation operators and provides a “mutant score” which shows the percentage of total mutants were killed by a user’s test set.

Jester and MuJava both have a high initial setup overhead and a configuration overhead for each project with which they are to be used. The functionality for performing Mutation Testing already exist in the open source MuJava libraries and therefore can be cleanly transformed into an Eclipse Application. MuClipse, as its name suggests, is a JMutation system which will function as an Eclipse plugin. This unique constraint will provide the system with direct access to the directory structure and environment variables, which it can pass on to the developer as an easy-to-setup Eclipse runtime configuration.

The requirements below are categorized by Functional and Non-functional but also by subcategory. They are prioritized using the following system:

#	Declaration of Meaning
1	If this requirement is not implemented in the application, the application is not complete.
2	This requirement is important, but the application can be usable without it.
3	This requirement is not important, but would improve upon the usability or appearance of the application.

2. Functional Requirements

2.1 Usability.

2.1.1

The system shall allow the developer to easily see which mutants were killed, which remain alive and the mutation score for the last execution of test cases.

Description: The results are the main point of performing mutation testing—if the developer is not able to see which mutants are not caught by the existing test set, then there is no way to improve the test set to kill the additional mutants.

Origin: The MuJava Tool.

Priority: 1

2.1.2

The system shall allow the developer to specify the location where mutants generated are to be placed, and where original source code, test cases and compiled Java classes are located.

Description: The original tool requires a very specific directory structure which it uses to look for mutants, source code, test cases and compiled Java classes (binary objects). The system should be able to take input before running which could override this structure if the developer so desires.

Origin: Ben Smith.

Priority: 1

2.2 Generating Mutants.

2.2.1

The system shall be able to generate, store and compile mutated classes.

Description: Given a set of mutation operators, the system shall perform the mutations on the various source code selections and save the results into a designated location. After it has finished, the system will compile these classes for later use.

Origin: The MuJava Tool.

Priority: 1

2.2.2

The system shall be able to perform the mutation operations the developer selects.

Description: The developer will be presented with a list of mutants that the MuJava libraries distribute. From the graphical interface, the developer can specify which mutation operators should be used with this pass of generation.

Origin: The MuJava Tool.

Priority: 3

2.2.3

The system shall mutate the source code the developer chooses.

Description: Since the mutation process can be quite time- and processor-intensive, the developer may only want to expose weaknesses for or focus on a single class. The system shall allow this option.

Origin: The MuJava Tool.

Priority: 3

2.3 Test Sets.

2.3.1

The system shall run the set of tests the developer specifies, tabulate the results of those tests and compare them with the results of each mutant.

Description: The developer will be presented with a list of choices for tests which can be run against the mutants that have been generated.

Origin: The MuJava Tool.

Priority: 2

2.3.2

The system shall be able to run tests selectively on a declared type of mutants.

Description: The developer will be presented with a class of mutant (class-level, method-level or both) choice on which to run test cases.

Origin: The MuJava Tool.

Priority: 3

2.3.3

The system shall run both MuJava tests and jUnit TestCases.

Description: These two test types (see glossary for more information) have different return types and are in fact different Object types. The system must be able to collate the test results for both.

Origin: Laurie Williams

Priority: 3

3. Nonfunctional Requirements

3.1.1

The system shall display both a textual output of its actions and an overall progress indication during its execution.

Description: Generating mutants and collating results for the various test cases takes a long time in mutation testing. The developer needs a progress indication more than the textual output provided by the tool. These textual indications do not give an endpoint that is clear enough.

Origin: Ben Smith.

Priority: 3

4. Constraints

4.1

The process of generating mutants and running test cases on those mutants shall be accessible in the form of an Eclipse Runtime Configuration.

4.2

The killed and live mutant results shall be available in the form of an Eclipse View.

4. Requirements Traceability Matrix

Requirement	->	2.1.1	2.1.2	2.2.1	2.2.2	2.2.3	2.3.1	2.3.2	2.3.3
	2.1.1	■		**X**			**X**		
	2.1.2		■						
	2.2.1			■					
	2.2.2			**X**	■				
	2.2.3			**X**		■			
	2.3.1			**X**			■		
	2.3.2						**X**	■	
	2.3.3						**X**		■

5. Development and Target Platforms

The MuClipse tool shall be developed to be a plugin which will be available for Eclipse v3.1.x, which is a multi-platform Integrated Development Environment.

6. Glossary

- class-level: a newer type of mutation which modifies method and class keywords to change scope, persistence and other object-oriented attributes
- Eclipse runtime configuration: a saved template of a runtime configuration for multiple repeated runs.
- jUnit TestCases: test objects which are of type junit.framework.TestCase. The structure is to have all methods with no return type and call static methods from the superclass to evaluate your results. If the results do not hold true, an exception of type junit.framework.AssertionFailedError is thrown.
- method-level: see traditional.
- MuJava tests: test objects created for use with the original MuJava libraries. The structure is to have a simple Java object with varying methods (which have various return types) starting with the word "test".
- mutant: a copy of the source code under test after it has undergone modification (mutations) by the mutation operators
    - live: the mutant did not cause its corresponding test case to fail
    - killed: the mutant caused its corresponding test case to fail
- mutation: the process of modifying segments of the original source
- mutation operator: the specifications for how a given class of mutation is going to be performed—implemented using a Reader and Writer for this operator
- mutation score: a percentage, given by (mutants killed) / (total mutants for this run), which acts as a metric for how effective the unit test cases are at detecting errors in the source code.
- mutation system: a program or tool which allows for both generating mutants and running a set of test cases on those mutants to compare with the original results
- runtime configuration: a set of objects on the Java classpath, parameters passed to the Java runtime environment, and other environment settings specified to allow a desired program (or tool) to run correctly
- traditional: the original mutation type, which consists of making changes to only source code which exists within a method body.
- test set: a group of Java classes which execute tests on their corresponding source classes

7. Document Revision History

Version	0.4
Name(s)	Ben Smith
Date	10/19/2006
Change Description	Dropped requirement of single-run execution

Version	0.3
Name(s)	Ben Smith
Date	10/19/2006
Change Description	Dropped requirement of single-run execution

Version	0.2
Name(s)	Ben Smith
Date	10/19/2006
Change Description	Moved many non-functional requirements into the functional section. Dropped implementation specifications that were not requirements. Added constraints section.

Version	0.1
Name(s)	Ben Smith
Date	10/6/2006
Change Description	Initial Draft