Mutation Testing

CS4110 - Software Testing and Reverse Engineering

Team Powerpuff
Ioana Leontiuc - 4515218
Mengmeng Ye - 4407326
Anelia Dimitrova - 4501667

INTRODUCTION

In Mutation Testing, we mutate (change) certain statements in the source code and check if the test cases are able to find the errors. A faulty version/changed version of the program is called a mutant. These changes are called operations. The PiTest tool we use to generate the mutants offers 13 available operators , out of which 7 are default [2]. The operators are described at: http://pitest.org/quickstart/mutators/
The idea behind mutation testing is to evaluate the quality of test suites that should be robust to an extent that they will fail mutant code (faults creation in the program). If the mutant and the original code generate the same output, then the mutant is killed, otherwise it is alive. [1]

SOFTWARE PROJECTS

We analyze this approach on three different project. All the selected projects have analysis provided by SonarQube [3]. We used SonarQube in the project selection process since their projects have test coverage statistics and other valuable metrics that we could consider. We looked for Java projects with a test coverage above 90%. We selected one small and two larger projects.

	Assertj-Joda-time	Sonar-CSS	Sonar-Flex
Authors	Free Software Foundation, Inc.	Free Software Foundation, Inc.	Free Software Foundation, Inc.
Source	https://github.com/joel-costigliola/assertj-joda-time	https://github.com/SonarQubeCommunity/sonar-css	https://github.com/SonarSource/sonar-flex
What does the project do	AssertJ assertions for Joda-Time provides assertions for Joda-Time types like DateTime and LocalDateTime.	Validates CSS code, suitable for QA analysis	The plugin enables analysis of ActionScript 2 and 3 within SonarQube, and offers coverage of some CWE rules.
Lines of code	553	9311	7482
Test line coverage	91.3%	90.1%	92.8%

Number of tests	121	316	276
Condition coverage	85.7%	87.7%	87.4%

Mutation TESTING

For each project we provide an overview on how the survivor mutants are distributed.

● Assertj-Joda-time

Project mutation score (default operators): 74%
Created mutants: 182
Survivor mutants: 48

In order to create an overview of the distribution of surviving mutants we generated from the cmd mutants from each type at a time. We added the useful information in the table at https://drive.google.com/file/d/0B0f_HjNBQ0AmRnNiRy0xb1NqWDQ/view?usp=sharing The red columns show mutants than are not compatible with the project. This means that the system does not include any operation the mutant wants to change. Therefore those types are not applicable to the given system.

Type 7: return values operator most mutant survive. This is because the operator changes the return value of the method. For a boolean value it will return the opposite and for an Object it will return null, or in case the method was about to return null it will throw an exception. Most of these mutants are not relevant and there is no need for test coverage. The scenarios created in the mutants are not plausible. They ignore the contract of the method and also predefined rules of programming. A good example is the following mutant:

> **Mutant**: DateTimeAssert.java -> line 73
> **Mutation Type**: Return Values Mutator (RETURN_VALS)) - return type object
> **Suggestion**: impossible case
> **Reasoning**: This type of mutant let function `isBefore(DateTime other)` returns null. The test cases doesn't test if this function returns null. However, it is not possible that the original code in the function `return this` will return a null, therefore it is not necessary to build a test case for it.
> **Repeat Times**: 36

However some valid test gaps were found by this operator, such as:

> **Mutant**: LocalDateTimeAssert.java -> line 187
> **Mutation Type**: Void Method Call Mutator (VOIDMETHODCALLS)
> **Suggestion**: missing test
> **Reasoning**: function checks the local date and time and makes sure it is not null. If it is null, it returns an illegal argument exception. However there is no function to check if the time is indeed null or not.
> **Repeat Times:** 4

The next big survivor mutant producer is **type 3: Conditionals Boundary Mutator**. This operator is a one to one mapping for branch coverage. SonarQube said this project has a 85.7% condition coverage and it suggested 7 missing branches for each of the two classes. The mutants table shows 7 survivors for each class. In case the project has branch coverage, this operator is redundant.

Operator **type 10: inline constant mutator** is a very useful and as it turn out scalable one mutants producer. This operator is very suited for testing a good flow and logic within the implemented algorithm. The same goes for **type 11: Non Void Method Call Mutator**.

Other representative example for this project are:

> **Mutant**: DateTimeAssert.java -> line 673
> **Mutation Type**: Return Values Mutator (RETURN_VALS)) - return type boolean
> **Suggestion**: missing test
> **Reasoning**: This mutant return true when the method is supposed to return false, and vice versa. The test is missing because it's possible that the function `haveSameYear(DateTime actual, DateTime other)` doesn't return the right value when comparing the years of two different date.
> **Repeat Times**: 36
>
> **Mutant**: LocalDateTimeAssert.java -> line 95
> **Mutation Type**: Void Method Call Mutator (VOIDMETHODCALLS)
> **Suggestion**: missing test
> **Reasoning**: the function of checking whether a parameter is null has been removed in the mutant, which suggest the test is missing for testing the function's null parameter.
> **Repeat Times**: 4

● Sonar-CSS

Values computed by the overall summary provided by running of all the default mutants.
Some of the subprojects had errors in the original version and we could not compile them.

Subprojects	checks	squid	plugin	**Total**
Mutants created	1279	125	26	1430
Mutation score	91%	58%	65%	71.3%

This Maven project consists of 4 subproject, out of which only 3 of them can have mutants generated on them. This project is substantially larger than the previous one and we look at the distribution of mutants per class file. For the squit subproject we were able to run all the individual operators from the cmd. Due to Mavel related issues we could only run the default operators on the other subprojects. We summarize the surviving mutants in the following table, by using the same approach:
https://drive.google.com/file/d/0B0f_HjNBQ0AmS1ZkcVQ0WVVuY3M/view?usp=sharing

Aside from all the observations made for the previous project from the table we can see that the mutants are spread over all the classes and very few duplicate hinting to the same class. This demonstrates that this method is very comprehensive.

Looking more closely to the mutants generated for the same class we noticed that some of them indicate to the same problem. To this end we propose bundle mutants. A bundle mutant is formed of two or more survivor mutants created from the same or different operator that indicate to the same problem. Some examples from the CSS project are:

Mutant: BewareOfBoxModel -> 90,94
Mutation Type: void method call mutator, negated conditional
Suggestion: missing test
Reasoning: both mutant are related to a larger if statement that resembles a switch structure. These mutants I believe indicate deadcode within the two of the cases. Apparently adding the WIDTH_FOUND combination does not affect the code neither does calling the clear method within the IS_BOX_SIZING case.

Mutant: BewareOfBoxModel -> 162,166,181

Mutation Type: negated conditional
Suggestion: missing test
Reasoning: all mutant point to the conditions that take into account the PercentageValueElement .

Mutant: CommentContainsPatternChecker -> 57,57,58
Mutation Type: negated conditional, changed conditional boundary
Suggestion: not relevant
Reasoning: this mutant bundle shows that changing the logic in the way the 2 boolean values are computer do not change the outcome, the OR value between the two booleans.

● Sonar-Flex

Values computed by the overall summary provided by running of all the default mutants. Some of the subprojects had errors in the original version and we could not compile them.

Subprojects	checks	squid	plugin	toolkit	Total
Mutants created	1115	156	66	2	1339
Mutation score	93%	55%	79%	50%	69.25%

Due to Mavel related issues we were unable to run operators individually from the cmd, except for the checks subproject. Therefore the overview provided in the table contains the default operators, the purple column header indicate the operators we did not generate.
https://drive.google.com/file/d/0B3pfB7dP7Gf3Tk96OTNZdGI3bGc/view?usp=sharing

We noticed that just as for the CSS project, there is no pair of operators that flag the same classes. To exemplify the need for human evaluation we found the following special case:

Mutant: FlexGrammar -> 22 different survivor mutants
Mutation Type: Void Method Call Mutator
Suggestion: complex human evaluation needed
Reasoning: this mutant bundle shows that eliminating the call to skipIfOneChild is not detected by the test suite. This may either be a missing test for the case where there is only one child or a flaw in the logic implemented. It might turn out there is no need to skip if there is only one child.

CONCLUSIONS

We consider mutation testing to be of great value in practice because of the fine grained faults in logic it can find. Except for the type 3: Conditionals Boundary Mutator all operators are worth looking into. Being that the mutant approach is very costly because it needs human evaluation, for type 3 operator we suggest as a better alternative traditional branch coverage analysis since this operator is a one to one mapping for conditional coverage. Another suggestion would be treating some mutants as part of a bundle. Realizing that more independent mutants are related to the same problem makes the whole process faster. For every project we had more mutants or no mutant for certain operators. This shows a direct dependency between the type of operators one should use within the project context. Taking into account all these we consider mutation analysis to be doable and worthwhile

REFERENCES

[1] http://www.guru99.com/mutation-testing.html
[2] http://pitest.org/
[3] https://nemo.sonarqube.org/