Identification of Low-value Code to Improve Efficiency of Automated Code Analysis

Marko Ivanković
Goran Petrovic

Follow this and additional works at: https://www.tdcommons.org/dpubs_series

Recommended Citation
Ivanković, Marko and Petrovic, Goran, "Identification of Low-value Code to Improve Efficiency of Automated Code Analysis", Technical Disclosure Commons, (December 24, 2020)
https://www.tdcommons.org/dpubs_series/3922
Identification of Low-value Code to Improve Efficiency of Automated Code Analysis

ABSTRACT

Certain software testing techniques such as mutation testing are often not utilized due to the relatively high proportion of testing activity that is not deemed useful. Such techniques can benefit from identification of high-value code. This disclosure describes the classification of different parts of a codebase into high and low value code. The techniques utilize an abstract syntax tree (AST) generated from the codebase on which predefined heuristics are applied to identify low-value code. A propagation algorithm is applied to mark low value branches. Additionally, machine learning techniques are applied to classify portions of code as high or low value. By restricting mutation testing (or other testing activity) to high-value portions of code, testing can be conducted such that a high proportion of testing activity is of value in uncovering flaws.

KEYWORDS

- Software testing
- Mutation testing
- Abstract Syntax Tree (AST)
- Unit testing
- Code coverage
- Machine learning

BACKGROUND

Code coverage is a measure of the degree to which a test suite exercises a software codebase. Although code coverage is well established in software engineering research, its adoption in industry is hindered by the prohibitive computational costs of analyzing coverage at scale. Further, it is a common belief that the effort to improve coverage may not be worthwhile since it often requires testing pieces of code that are obviously not important.
Therefore, a large proportion of practitioners often deem certain analysis to be too expensive to perform. An example is mutation testing, which is a technique of establishing testing suite efficacy by inserting small faults into programs and measuring the ability of the test suite to detect them. Performing mutation testing on all lines in a codebase however would require astronomical resources, while performing mutation testing only on important lines is perfectly feasible. However, there are no available techniques to identify the important lines in a codebase.

DESCRIPTION

This disclosure describes techniques to identify low-value code in a codebase. Fig 1 illustrates an example process to predict the value of different parts of code, per techniques of this disclosure.

```
<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Parse the code into Abstract Syntax Tree (AST)</td>
</tr>
<tr>
<td>104</td>
<td>Identify portions of low value code using pre-defined heuristics</td>
</tr>
<tr>
<td>106</td>
<td>Apply propagation algorithm on the AST to mark low-value branches</td>
</tr>
<tr>
<td>108</td>
<td>Use machine learning to predict high-value and low-value parts of code</td>
</tr>
<tr>
<td>110</td>
<td>Make predicted code value available for further processing</td>
</tr>
</tbody>
</table>
```

Fig. 1: Steps for identifying low value code

The codebase that is to be analyzed is first parsed (102) into an Abstract Syntax Tree (AST). An abstract syntax tree is a tree representation of the abstract syntactic structure of source code written in a programming language. Predefined heuristics are applied (104) on the AST
representation of the codebase to identify small patterns of low-value code. For example, a large number of such heuristics can be applied to identify nodes from the AST that represent low value code. Some examples of such heuristics include logging statements, axiomatic statements for the programming language, etc. The set of heuristics to be applied can be created based on historical developer feedback.

A recursive propagation algorithm is applied on the AST to mark entire branches as low-value when the entire content of a branch is known to be low-value. Machine learning techniques, trained on previous developer behavior with reference to various codebases, are then applied to predict parts of the code that developers are likely to interact with (marking those as high-value) and parts of the code that the developers are unlikely (marking those as low-value). The model output is made available to code analysis tools for further processing such as filtering points of interest.

The techniques described in this disclosure can be used to identify low value code and improve coverage in other software testing methods such as unit testing. The identification of high-value code can provide substantial improvements in mutation testing, e.g., can improve the usefulness rate of mutation testing.

CONCLUSION

Certain software testing techniques such as mutation testing are often not utilized due to the relatively high proportion of testing activity that is not deemed useful. Such techniques can benefit from identification of high-value code. This disclosure describes the classification of different parts of a codebase into high and low value code. The techniques utilize an abstract syntax tree (AST) generated from the codebase on which predefined heuristics are applied to identify low-value code. A propagation algorithm is applied to mark low value branches.
Additionally, machine learning techniques are applied to classify portions of code as high or low value. By restricting mutation testing (or other testing activity) to high-value portions of code, testing can be conducted such that a high proportion of testing activity is of value in uncovering flaws.

REFERENCES