Design and Software Quality Metrics

Exploring the Relationship between Design Metrics and Software Diagnosability using Machine Learning (2018)

Thomas Dornberger   Sofie Kemper

2018-05-24

1 Metrics

1.1 Test Suite Characteristics

The following metrics aim to quantitatiely analyse a project's test suite. We use basic metrics provided via Gzoltar (see 2.1) as well as test suite metrics proposed by Perez et al in "A Test-suite Diagnosability Metric for Spectrum-based Fault Localization Approaches" and shown to be good indicators of spectrum-based fault localisation accuracy. These metrics are computed based on the coverage data provided by Gzoltar (see 2.1).

On top of that, we use the number of test cases and test size normalised via the number of source lines of code as a coarse approximation of test coverage which would be infeasible to calculate with our limited computation power. Other metrics such as the number of relevant components (i.e., all components tested by at least one of the relevant tests) were rejected since equivalent information is already contained in the dynamic metrics.

1.1.1 Number of Relevant Passed Test Cases (T-NP)

The total number of test cases out of the test suite that were passed, i.e., executed successfully. We measure this based only on the relevant test classes, i.e., all test classes which load any of the changed components. This metric is calculated via Gzoltar (see 2.1).

1.1.2 Proportion of Passed Test Cases (T-PP)

The proportion of passed test cases out of all relevant test cases.

1.1.3 Number of Failed Test Cases (T-NF)

The total number of test cases out of the relevant test suite that were failed, i.e., that could not be executed successfully. This number can be extracted by Gzoltar (see 2.1) or accessed via `http://program-repair.org/defects4j-dissection/#!/`.

1.1.4 Proportion of Failed Test Cases (T-PF)

The proportion of failed test cases out of all relevant test cases. This number can be extracted by Gzoltar (see 2.1) or accessed via `http://program-repair.org/defects4j-dissection/#!/`.

1.1.5 Number of Test Cases (T-N)

This metric counts the total number of test cases in the given test suite.

1.1.6 Number of Relevant Test Cases (T-RN)

The number of test cases that are relevant for the buggy version, i.e., the number of test cases in the classes in which at least one test case fails because of the bug. This measure is provided by gzoltar (see 2.1).

1.1.7 Proportion of Relevant Test Cases (T-RP)

This metric determines which proportion of all test cases in the test suite are considered relevant, i.e., load at least one of the changed methods during execution.

1.1.8 Test Case Density (T-DSLOC)

This metric measures number of test cases per 100 source lines of code, i.e., it gives an indicator of how well-tested the source code is.

1.1.9 Test Size (T-SLOC)

This metric indicates the size of the tests by measuring the total number of lines of code (excluding blank lines and comments) of tests.

1.1.10 Relative Test Size (T-RSLOC)

The relative test size quantifies the size of the test suite (measured via source lines of code) normalised by the size of the project (also in source lines of code) since we expect a strong positive correlation between these two features/

1.1.11 Test Density (T-D)

The test density is a measure that ensures that components are frequently involved in tests. It is computed as T-D= $\sum A[i,j]/(N \cdot M)$ and then normalised, s.t., a value of 1.0 is the target.

1.1.12 Test Diversity (T-G)

The test diversity tries to quantify to what extent components are tested in diverse combinations. We compute it as T-G$= 1 - \sum n \cdot (n-1)/(N \cdot (N-1))$ which yields values in the interval $[0;1]$ where 1 is seen as the optimal value.

1.1.13 Test Uniqueness (T-U)

The test uniqueness tries to measure to what extent components are distinguishable based on the test results. It is computed as T-U=|T-G|/T-M.

1.1.14 Perez et al's diagnostic predictor (T-DDU)

This metric tries to combine the previous ones in a way that provides the maximum amount of information concerning the test suite's influence on software diagnosability. It is partly based on the notion of entropy. The metric is computed via the formula T-DDU = T-P $\cdot$ T-G $\cdot$ T-U and, thus, produces values in the interval $[0;1]$ where a value of 1 is regarded as ideal.

2 Tools

We have tried a multitude of different tools (SourceMeter, LOCC, SonarQube, CCCC, USC CodeCount, CLOC, etc.) and chosen the following for the interesting metrics they provide as well as the possibility to automate their usage, i.e., the possibility to execute them on the command line and output formats that can be easily parsed and used for our purposes.

2.1 Gzoltar

Gzoltar is a tool used for spectrum-based fault localisation, i.e., for automatic testing and debuging. It is provided as a command-line tool as well as an Eclipse plugin. It produces coverage information as well as suspiciousness scores for different artifacts. More information can be found via `http://gzoltar.com`.