Machine Learning for Software Risk Analysis

An Appsurify™ Technical White Paper

Overview

TestBrain™ is a SAAS package developed by Appsurify, Inc. that uses machine learning to make software QA testing more efficient. TestBrain reviews the software code repository, development history, and test results and applies machine learning to:

(1) for automated testing, determine a targeted set of tests to run to validate each new commit;

(2) for manual testing, predict which commits have the highest risk of containing a defect to generate a prioritized list of tests to run;

(3) identify and isolate test failures caused by flaky tests rather than code defects; and

(4) generate metrics on developer efficiency.

One important component of the machine learning is the creation of a risk model to predict the likelihood that a commit contains a defect.

TestBrain's risk model builds on academic research into techniques and metrics able to predict software defects, and adds proprietary metrics and refinements based on the company's extensive experience testing open source and commercial code bases.

This white paper describes the metrics that TestBrain uses for its risk model.

Academic Research

Academic research into techniques and metrics useful for analysing risk associated with software changes in large-scale software projects dates back at least two decades.[1] Academic models are generally classified according to whether they are based on code metrics or process metrics. Code metrics are attributes of the code itself such as the cyclomatic complexity (the number of logical paths that can be taken through a code structure), depth of inheritance (a measure of the inheritance levels from the object hierarchy top) and class coupling (the number of classes a single class uses). Process metrics focus on properties of the software development process such as the size of the changes, number of files modified, and experience of the developers.

Research comparing the correlation of code metrics and process metrics with the presence of defects has concluded that process metrics outperform code metrics, and equals or even outperforms the combination of code metrics and process metrics.[2,3] Consequently, Appsurify has focused on implementing a robust set of process metrics that provide strong predictive correlation to the likelihood of software defects.

---

[1] Khoshgoftaar, Taghi & C. Munson, John. (1990). Predicting Software Development Errors Using Software Complexity Metrics.. IEEE Journal on Selected Areas in Communications. 8. 253-261. 10.1109/49.46879.

[2] Moser, Raimund & Pedrycz, Witold & Succi, Giancarlo. (2008). A Comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. Proceedings - International Conference on Software Engineering. 181-190. 10.1145/1368088.1368114.

[3] Foyzur Rahman and Premkumar Devanbu. 2013. How, and why, process metrics are better. In Proceedings of the 2013 International Conference on

The TestBrain Solution

The TestBrain risk analysis builds upon the results of academic research and adds proprietary metrics that Appsurify has found to improve the prediction accuracy for large-scale, commercial software development projects.

The process metrics used by TestBrain fall into three subcategories: (1) commit metrics from academic research that measure the commit process; (2) developer metrics from academic research that measure the developers making the commit; and (3) TestBrain proprietary metrics.

Commit Metrics

Commit metrics measure how the code is being modified.

In general, larger changes are more likely to contain a defect than smaller changes. Changes across multiple files are more likely to contain a defect than the same number of lines modified within a single file. Frequency of change of the file has also been found to predict defects – when code has stagnated for a long time, the next change is likely to introduce a defect. But code changed by multiple developers within a short interval also increases the probability of a defect.

Specific commit metrics identified by academic research as having a high correlation with the risk of defect and utilized by TestBrain are listed in Table 1.

Developer Metrics

Developer metrics measure the attributes of the developers writing the code and making the commit, such as their experience with the code base in general and in developing the specific area of the code in the commit.

Specific developer metrics identified by academic research and utilized by TestBrain are listed in Table 2.

TestBrain Metrics

In addition to metrics identified in the academic research, TestBrain includes proprietary metrics that based on the company's experience with both open source and large scale commercial software projects, it has found to increase the accuracy of its predictions.

One set of TestBrain proprietary metrics interpret the developers' commit messages and code comments to determine what task the developers were completing and the confidence they have in their own commits.

Another set of proprietary metrics examines the details of the specific code changes for attributes such as repetition and history. For example, a single 100 line code update has a higher risk of defect than 5 lines of change repeated in 20 locations. Similarly, if the same change was made previously without causing a defect, it would indicate a low risk for subsequent updates.

TestBrain also refines the academic developer metrics by taking advantage of user defined areas and historical test results.

Software Engineering. IEEE Press, Piscataway, NJ, USA, 432-441.

Table 1: Commit Metrics Used by TestBrain

- Lines of code in the file before the change
- Number of lines of code added to the file in the commit
- Normalized number of lines of code added to the file in the commit
- Number of lines removed from the file in the commit
- Normalized number of lines removed from the file in the commit
- The number of characters in the commit message
- Number of modified directories
- Number of previous commits for the file
- Number of times the file has been modified alone
- Number of active developers who previously modified the file
- Total number of distinct developers who contributed to the file
- Whether or not the commit was done by the owner of the file
- Number of developers who contributed less than 5% of the file
- Number of files modified by the committer
- Number of developers who modified each file in the commit.
- Total number of distinct developers who modified each file in the commit.
- Number of previous commits made to files in the commit
- Number of other files modified by the developer in commits where the same file was modified
- Entropy of changes of the file

Table 2: Developer Metrics Used by TestBrain

- Total number of commits made on the file by the prime author
- Percentage of lines written by the prime author in the project
- Number of commits made on the file by the prime author in the previous month
- Number of commits made by the developer in the package containing the file
- Average number of commits made by all developers in a commit
- Average amount of time between commits
- Time of day when the commit was made

TestBrain Learning Process

Upon initial configuration where TestBrain is connected to the project code repository, TestBrain creates an initial risk profile by reviewing the complete commit history. From that time, TestBrain integrates with the test infrastructure to review all new test results to continue to refines its risk analysis.

Initial Learning from Code Repository

When initially configured, TestBrain connects to a Git-based project repository. For existing projects with a substantial development history, it analyzes the history of commits to build a learning database and determine where defects were previously created.

TestBrain analyzes the text within the commit messages to identify the commits made to fix defects. Once TestBrain has identified a “fixing commit”, it reviews the code history using the *git blame* function to find the prior commit that caused the defect. TestBrain then uses these “defect commits” as historical data to predict future defects.

To correct any mistakes caused by missing or misinterpreted commit messages, users are encouraged to verify the defects identified by TestBrain and the commits that caused them. However, any errors here will only impact the initial settings that will be gradually be superseded by the live test data.

For new projects or small projects without a large historical data set, TestBrain uses a general model based on prior analysis of a collection of open source projects. However, since the general model does not include metrics for the new code base and development team, it is less accurate. Once a large enough data set is available, TestBrain updates the model based on the project code base.

Subsequent Learning from Live Test Results

Once the initial learning is complete, TestBrain integrates with the test frameworks and continues to learn and refine its risk analysis by reading the results of all new tests and associating them with each new commit. Since this process does not depend on the completeness and accuracy of commit messages, it is more accurate. However, the learning process takes place gradually over time as it collects data from new tests and failures. The more tests and the more failures that TestBrain sees, the more accurate its results become.

Prediction Accuracy

A large number of factors affect the accuracy of the predictions made by TestBrain. The more historical data and live test data it has, the more accurate it becomes. It also increases in accuracy when the development team is stable and continuing to work on similar code areas and tasks.

Because the initial learning depends on commit messages that include text “fixed” or “resolved”, the completeness and accuracy of the commit messages will have a large impact on the initial metrics. Users can improve the accuracy my manually correcting any mistakes in this initial defect analysis.

TestBrain has been tested on large open source repositories to gauge its accuracy. For the AngularJS project, TestBrain was able to predict 82% of all defects fixed.

Future Enhancements

Appsurify’s R&D team is continuously testing additional metrics and models to improve the accuracy of the models. A recent release added integration with the Jira bug tracking tool to make use of the historical defect information cataloged there to improve the reliability of the initial learning process as well as automatically populating the

results of new tests. Future releases will add integration with additional bug tracking and test case management tools to take advantage of the defect information available.

About Appurify

Appsurify applies machine learning to make software testing smarter, faster, and cheaper. By improving the efficiency of software testing, Appsurify allows you to release your products faster with fewer defects.The Appsurify team builds upon decades of experience in software testing to overcome today's limitations. Appsurify was founded in 2017 and is headquartered in Santa Monica, CA, with development based in Auckland, New Zealand.

Appsurify, Inc.
appsurify.com
info@appsurify.com
1.650.402.1400
Nashville, TN | Auckland, NZ