Software Regression Test Recommendations using Machine Learning

Hardik Surana,Akhil M,Srishti Mishra,Ashish Sanu,Bilal Shakil Belgaumwala,Arpit Singh

Department of CSE,PES University,Bangalore,India

aashishsanu6@gmail.com,bilalshakil1443@gmail.com,aptnrg007@gmail.com

***Abstract*—This paper focuses on carrying out Regression Testing in an efficient manner using Machine Learning models. Large codebases having test suites consume more time performing checks as the size of the test suite increases. This paper describes a novel way to achieve this goal thereby saving quality code review time.**

***Keywords—Machine Learning, model,test case,test suite***

# I. Introduction

REGRESSION means retesting the unchanged parts of the application. Regression Testing is the one in which test cases are re-executed in an order to check whether the previous functionality of the application is working fine and the new changes have not introduced any new bugs.

Although software may be completely tested at some point during its development and maintenance, program changes require that parts of the software be retested. Mistaken and changed requirements cause the software to be reworked. New uses of old software yield new functionality, not originally conceived in the requirements. The management of this change is critical to the continuing usefulness of the software. The new functionality added to a system may be accommodated by the standard software development processes. Regression testing attempts to revalidate the old functionality inherited from the old version. It is the process of validating the modified parts of the software and ensuring that no new errors are introduced into the previously tested code. Even though the modified program may yield correct outputs on test cases specifically designed to test the modifications, it may produce incorrect outputs on other test cases on which the original program produced correct outputs. Thus, during regression testing, the modified program is executed on all existing regression tests to verify that it still behaves the same way as the original program, except where change is expected. This is necessary because small changes in one part of a program may have subtle undesired effects in other seemingly unrelated parts of the program.

The current challenges to regression testing range from

1) Time taken to run the whole test suite to test the entire program can be in hours or days

2) Desynchronized test suites with new releases (running tests which are no longer relevant to the current requirements of the system)

3) Short-term test suites (tests which become irrelevant fairly quickly but are not removed/modified)

4) Old and redundant test suites

Addressing points 2,3 and 4 using the selective approach outlined below can lead to a shorter running time during regression testing and catch bugs faster.

A selective approach to regression testing attempts to identify and retest only those parts of the program that are affected by a change. The three main ways to do so are – Minimization, Selection and Prioritization [6]. In this project, we propose a method for retesting of a small subset of tests which performs the checks as well as when all the tests were run which utilizes all the methods mentioned earlier.

The selective strategy uses a subset of the existing test cases to reduce the retesting cost. In this strategy, a test unit must be rerun if and only if any of the program entities, e.g. functions, variables etc., it covers have been changed. The challenge is to identify the dependencies between a test case and the program entities it covers.

A Machine Learning model trained on metrics collected from the codebase, is needed to predict the outcome of the subset of selected test cases before actually running them in the decreasing order of probability of failure.This is done so that the test case with a higher probability of failure can fail earlier, thus saving time.

II. MOTIVATIOn

As a software system ages, the cost of maintaining the software dominates the overall cost of developing the software. Regression testing is used to determine if a modified program still meets its specifications or if new errors have been introduced. Improvements in the regression testing process would help reduce the cost of software.

In this project, the selective strategy was applied. An optimal subset of test cases to test were found using machine learning algorithms by predicting the test cases that would fail the build given the changes made in a commit. Hence, these tests were more likely to find bugs in the code when a new functionality was added.

# III. LITERATURE SURVEY

There were many approaches to this problem, out of which these were of interest to us as they aligned with our problem perspective. One of them is through the use of slicing [2] which identifies affected statements and parses them to create def-use pairs for variables whose definition changes since the previous build. Another method utilizes incremental testing [3] which is used in real-time testing on production systems. In this method, the tests that need to be re-checked calculated in the background using off-line processing. More recent research has been performed which utilizes historical information of test case data such as [4]. In today's age of cloud computing, being able to run regression test analysis on the cloud is but obvious. [5] looks in this space for optimal resource utilization of cluster resources. What was observed through this survey of existing research was that historical information and static code analysis for affected changes are the two most important factors which affect the decision of which test cases must be tested again and which among them have more importance and must be tested before others. Our proposed method mentioned below highlights the steps taken for efficient implementation of the same.

Coming to machine learning based implementations, in one of the papers the authors used Genetic Algorithm [1] on the regression test suite to prioritize test cases dynamically on the basis of complete code coverage during time-constrained

execution. They implemented two main selection algorithms –PMX and cyclic crossover. The optimization of test cost was

done for a software project with 10 test cases. They assigned a

random cost to each test case and performed the analysis based on this. They considered a random population, minimum test cost as fitness function, 10 as the population size and ran the algorithm for 100 iterations, first choosing PMX crossover and next choosing cyclic crossover and compared the two. They calculated process cost and test cost in each case. The performance comparison of PMX and the Cyclic Crossover in case of a suite of 10 test cases (100 generation) was plotted. Cyclic crossover was found to be less effective in terms of time. The authors also found that cyclic crossover was more effective in terms of test cost optimization. They concluded that the genetic algorithm can be used effectively to perform the work on test case generation but it can be used effectively if there is large number of test cases and large number of possible test sequences. This approach just provides a comparison between the two algorithms. The cost assigned was random. The paper

does not talk about real applications where more complex metrics are involved. In [7] the authors proposed to to get

maximum fault coverage in minimum execution time in

regression testing by using Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Hybrid Particle Swarm

Optimization (HPSO). The algorithm proposed in this research paper executes in polynomial time with complexity of O(n2 )

where, n is the population generated. The evaluation metric chosen is APFD (Average Percentage of Faults Detected). 75.6% of fault coverage has been done by HPSO. The automation of algorithm has provided solid base for its effectiveness. In [8] the authors designed a test suite minimization technique consisting of two modules -The Analyzer module and the Optimizer module. The Analyzer module consists of evolutionary algorithms like Random forest/ Genetic Algorithm. Optimizer module consists of Greedy Algorithm. The N test cases that are to be recommended will be derived upon executing the analyzer. These algorithms are fed with historically derived inputs such

as test case execution frequency, test case failure pattern, change feature pattern and bug fixes & associations. The authors proposed a relevancy based metric that is derived

from historical executions of the test cases. Instead of adopting most literature referred methods such as coverage based and fault counts, the method used in this paper tries to automatically associate important test cases to file changes across builds. From this the most suitable test cases can be found for the regression build.

[9] prioritizes test cases based on total coverage of code components (covering previously uncovered code components), estimated ability to show faults (code that might show a lot of bugs) and compares the rate of fault detection to the fault rates of untreated, randomly ordered, and optimally ordered test suites. Priority of execution of test suites was ordered according to fast rate of code coverage, frequently used features, historically buggy code, increased rate of finding high risk faults, increased likelihood of revealing faults, related to specific code changes in the regression testing process.

# IV. METHODOLOGY

## *Data Collection*

In this project, we have proposed a method which utilizes historical test and source code analysis based features and uses machine learning to generate test case recommendations. These are the test cases which must be prioritized. The data is collected for the metrics mentioned below. To generate this historical data, we chose a sample repository called wicket-bootstrap [10] and collected the metrics by checking out to the various git commits of the repository. An open-source plugin called Clover [11] which is a plugin for maven-based java projects was used to collect data for code coverage and complexity. The data is made available by the plugin through an HTML report which is later scraped and stored in a CSV file. Another java based testing framework called Junit testing was use to find test case run time and the type of error. This generates an XML file which is scraped to store into a csv file.

In the case of test case failure, the category/type of error was identified by scraping the Surefire reports generated during testing.

Information about files changed (number of insertions and deletions) and time of commit was scraped from Github. This was then used to find code maturity per class (more changes within a short duration implies less stable code).

*Metrics :* The metrics were collected for each build of the repository by automating the deployment and testing process over 2000 commits.

The metrics which stood out as most important and which we have collected for the machine learning model based on recent research are:

* Code Coverage - Statement and Branch Level
* Test Case Failure Rate
* Cyclomatic Code Complexity
* Test Case run time
* Type of Error
* Maturity of source code
* Outcome (Pass or Fail)

These metrics are then aggregated into a dataset.There is a need to downsample the metrics as there is a majority of passed testcases and a minority of failures. If not done so, it would lead to an unbalanced dataset. The Github repository (wicket-bootstrap) which it is built from consists of about 2000 commits which approximates to about 8 lakh rows in the dataset.The size of the datasets validates the use of the machine learning model.

## *Machine Learning Model*

The machine learning models we here want to implement includes Random Forest Classifiers, XG Boost, Logistic Regression and Naive Bayes with SMOTE(Synthetic Minority Over-sampling Technique) and Cohen Kappa Score and Probabilistic Neural Networks.

For Random Forest Classifiers, XG Boost and Probabilistic Neural Networks,each of these models are trained on the downsampled set of metrics and their performance is evaluated.

Furthermore, class wise datasets are created. Following this,Logistic Regression and Naive Bayes are trained and evaluated on aggregated class metrics, aggregated source code metrics for every test class, the same is repeated for aggregated source code metrics in which the minority is oversampled using the SMOTE technique.

Each of these models are trained on the metrics and their performance is evaluated. Since the performance of Random Forest Classifiers was the best, it was decided to go ahead with it.

## *Selection of test cases*

For generating the test-case recommendations, we have

optimized the results by using statically generated dependency graphs. For this purpose, we have used an open-source maven plugin called STARTS [12] which stands for Static Regression Test Selection. This tool generates type-dependency-graphs to figure out impacted tests. Therefore, this performs selection and minimization of the test cases. The recommendations are later prioritized by the machine learning model.

Each of the selected test cases is then associated with the most recent metrics from the dataset. This is done by searching through the dataset, commit wise in reverse chronological order. If it is a new test case, it is assigned a set of random test metrics.

Following this, the outcome of each selected test case is predicted by the model, and the test cases are ranked in decreasing order of their probability of failure.

## *Running the test cases*

The test cases, ranked according to decreasing order of their probability of failure, are run using the Surefire plugin. The results of these test cases are collected using the Clover plugin, using which the misclassifications are displayed.

V. RESULTS AND DISCUSSION

The results produced by the Random Forest classifier have an accuracy of 99.99% leading us to believe that there is some amount of overfitting in the model. We believe that working more on the model and making it predict better will lead to better results.

The misclassifications are most likely due to the fact that the most recent metrics of a test case are older.

VI. FUTURE WORK

As more test cases are added to the code and run, its metrics are appended to the dataset. After a certain threshold (that was chosen) is passed,the model is trained on these additional test cases. This eventually makes the model better and will serve as a ground for the future way of Regression Testing, faster and more efficient.

##### ***Acknowledgment***

We would like to thank Mr. Dinkar Sitaram,Head of CCBD,PES University for his help and guidance throughout this project.

##### ***References***

1. A Genetic Algorithm for Regression Test Sequence Optimization - Suman, M.Tech. Student (Computer Engineering) Yadavindra College of Engineering, - Seema, Assistant Professor (Computer Engineering) (Supervisor) Yadavindra College of Engineering
2. Gupta, Rajiv, Mary Jean Harrold, and Mary Lou Soffa. "An approach to regression testing using slicing." In Software Maintenance, 1992. Proceedings., Conference on, pp. 299- 308. IEEE, 1992.
3. Agrawal, Hiralal, Joseph Robert Horgan, Edward W. Krauser, and Saul A. London. "Incremental regression testing." In Software Maintenance, 1993. CSM-93, Proceedings., Conference on, pp. 348-357. IEEE, 1993.
4. Kim, Jung-Min, and Adam Porter. "A history-based test prioritization technique for regression testing in resource- constrained environments." In Software Engineering, 2002. ICSE 2002. Proceedings of the 24th International Conference on, pp. 119-129. IEEE, 2002.
5. Orso, Alessandro, Nanjuan Shi, and Mary Jean Harrold. "Scaling regression testing to large software systems." In ACM SIGSOFT Software Engineering Notes, vol. 29, no. 6, pp. 241-251. ACM, 2004.
6. Yoo, Shin, and Mark Harman. "Regression testing minimization, selection and prioritization: a survey." Software Testing, Verification and Reliability 22, no. 2 (2012): 67-120.
7. Hybrid Particle Swarm Optimization for Regression Testing - Written by Dr. Arvinder Kaur and Divya Bhatt, University School of Information Technology.
8. Recommendation and Regression Test Suite Optimization Using Heuristic Algorithms - Satya Sai Prakash K, Prasad S U M Dhanyamraju, Gopi Krishna D (HCL Technologies, Hyderabad)
9. Prioritizing Test Cases For Regression Testing - Gregg Rothermel, Member,Roland H. Untch, Member, Chengyun Chu, Mary Jean Harrold, Member, IEEE Computer Society
10. https://github.com/l0rdn1kk0n/wicket-bootstrap
11. https://docs.atlassian.com/clover-maven-plugin/4.1.2/usage.html
12. Legunsen, Owolabi, August Shi, and Darko Marinov. "STARTS: STAtic regression test selection." In Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 949-954. IEEE Press, 2017.
13. ”STARTS : STAtic Regression Test Selection.” Owolabi Legunsen, August Shi, Darko Marinov, Department of Computer Science University of Illinois at Urbana-Champaign, USA{legunse2,awshi2,marinov}@illinois.edu
14. ”An Extensive Study of Static Regression Test Selection in Modern Software Evolution” Owolabi Legunsen, Farah Hariri, August Shi, Yafeng Lu,Lingming Zhang, and Darko Marinov ,Department of Computer Science, University of Illinois at Urbana-Champaign{legunse2,hariri2,awshi2,[marinov}@illinois.edu2Departmen](mailto:marinov%7D@illinois.edu2Departmen) of Computer Science, The University of Texas at Dallas{yxl131230,lingming.zhang}@utdallas.edu