TSAI - Test Selection using Artificial Intelligence for the Support of Continuous Integration

Maria Laura Brzezinski Meyer [1,2]
[1]*LAAS-CNRS, INSA, Université de Toulouse, France*
[2]*Renault Software Factory, Toulouse, France*
mlbrzezins@laas.fr

***Abstract*—The agile methodology has been increasingly deployed in the industry world, breaking the process into cycles of planning, executing, and evaluating. In the software development domain, an agile method named continuous integration is widely used to automatically integrate code changes from different developers into the same software. Then, each new build can be tested to make sure that the modifications did not interfere with the rest of the already verified code. Despite being very important, regression tests are usually the costliest part of a project. It is laborious to retest all tests of each new software version due to the time it takes to perform and often, before all tests are finished, a new software version is ready to be tested. To improve regression tests results, a selection can be done. By selecting the right tests at the right moment, the use of all test catalogs can be avoided to find faults in the software tested. The aim of this work is to develop a method to select tests to be executed for each version using artificial intelligence algorithms. Learning algorithms can find patterns and similarities between test cases to help knowing which one has a higher probability to expose a fault.**

***Index Terms*—Continuous Integration, testing, regression tests, artificial intelligence, industrial case study**

I. Introduction

The traditional "V" model is being replaced by agile methods in the industries. In a "V" model process, validation is done only at the end of development, causing faults to be discovered late. However, in agile process, validation is done for each new cycle, allowing faults to be exposed and repaired earlier. The concept of doing small cycles of release – build – test – deploy was first presented in [1] and is called continuous integration (CI). First, one or more developers write a code, then it is released and integrated into the main project. After the build, it is tested and finally, the software can be deployed. In this context, regression tests are needed to verify that new modifications did not interfere with the rest of the code. This project is inserted in the domain of the automotive industry, where the process complexity is increasing. Therefore, CI is being employed to boost the projects execution time, making it possible to care out tests daily as new realeases are added. Allowing faults to be found earlier and to be fixed before the next release, this methodology improves agility and decreases the cost of the project. Although it intends to reduce risk [2], a problem is created because the time is limited in the short continuous cycles, making it impossible to execute all the tests at each interation. At Renault Software Factory, there are two cycles of tests: a short cycle – where all the fast tests are executed, like smoke tests – and a long cycle – where longer and costlier tests are executed, like regression tests. An example is showed in figure 1. The short cycle tests are executed daily in each new version of the software. The long cycle tests are executed weekly for one version of the software. Each SWx represents a new version of the software being developed.

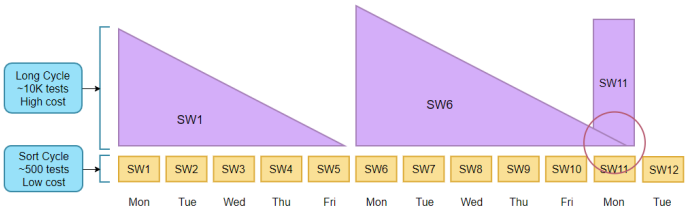

Fig. 1. Long and short test cycle with overflow example.

As can be seen in Figure 1, the long cycle may take longer than expected to be completed, which makes the process impossible as a new cycle must start. To mitigate this issue, a subset of the tests can be selected to be launched in each new build. With the increasing use of artificial intelligence (AI) methods, learning can be used to classify tests so a set can be chosen to be executed. The purpose of this work is to choose tests cases to form a test set to be performed daily into earlier steps so that faults can be revealed as soon as possible. Thus, the number of long cycles can be spaced to prevent the test campaign from not ending after a new cycle starts. Figure 2 shows the new test set included by the selection method for each software version (in blue) and the new spacing between long cycles.

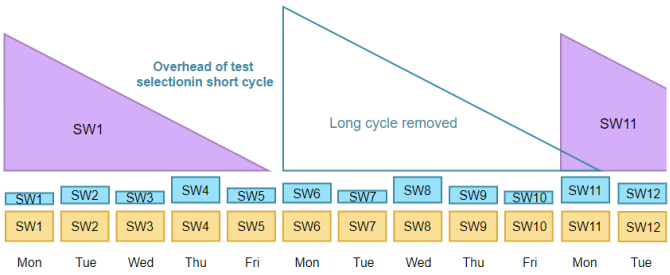

Fig. 2. Long test cycle removed and addition of selected tests example.

Therefore, this study addresses the three following research questions:

- RQ1: Which data is needed in the regression test selection process and how to obtain and use it?
- RQ2: Can AI improve the testing selection process and which algorithms can be used with the available data?
- RQ3: How to analyze the efficiency and the risks reported by the methods?

This thesis started in March 2021 within the Renault Group and the Laboratory for Analysis and Architecture of Systems - French National Centre for Scientific Research (LAAS - CNRS).

The main contributions of this work are:

- Test execution time reduction by performing smart test selection, which decreases the testers workload and costs;
- Allows to focus on tests results analysis due to this time saving;
- Early failure detection, which allows fixing the code in earlier stages of the project;
- Better compatibility of agile methodologies with an industrial sector that has long cycle testing;
- Risk management of the software quality in function of validation execution time.

The paper is structured as follows, Section II outlines the work already done in the literature that is relevant to this thesis. Then, Section III discuss the challenges faced in the theme, the work timeline and milestones are explored in Section IV and Section V concludes.

II. Related work

Yoo and Harmen [3] classified regression testing methods into three categories: minimization, selection and prioritization. In test suite minimization (TSM), the aim is to identify and remove redundant test cases from a test suite in order to reduce its size. Regression test selection (RTS) addresses the dual problem of identifying which test cases to retain from the test suite. The main focus is usually on the software changes, so tests that cover modified parts of the code are selected. The third type of methods is test case prioritization (TCP), which neither removes nor selects tests, but assigns them an order of execution seeking to maximize early fault exposure. TSM, RTS and TCP can complement each other in a global regression testing strategy. For example, test cases can be first selected to reduce the test size and then ordered to reveal faults earlier. Or a minimization technique can be applied after the selection to further reduce the number of tests. Or still, the test selection can be achieved by prioritizing the test cases, only retaining the highest-priority ones. Conceptually, the three categories of methods are closely related. All of them involve some comparative characterization of each individual test cases, based on which an optimal decision has to be taken (e.g., which test cases to remove, retain or execute first).

A. Test Suite Minimization

From a TSM perspective, the value of the tests is typically determined by referring to test requirements, like structural coverage requirements. A test case can be removed if it does not cover elements missed by the other cases. TSM then searches for the smallest possible subset of test cases such that it is no longer possible to remove test cases and preserve full coverage. In a review by Singh and Santosh [4], the techniques used in test case minimization are analyzed, which are: Heuristic-based, Genetic Algorithm based approaches, Integer Linear Programming based approaches, and Hybrid ones. The authors conclude that it is difficult to tell which technique has the best performance because each of them stands out over the other in some aspect. Note also that, irrespective of the chosen optimization technique, the effectiveness of TSM fundamentally depends on a debatable assumption: that the redundant cases in terms of coverage are also redundant for finding faults. The assumption may not hold, as evidenced by the empirical results of Rothermel et al. for code coverage [5], and Heimdahl for model coverage [6]. This yielded Marijan et al. [7] to revisit the notion of redundancy, in order to consider fine-grained coverage patterns as well as the failing history of test cases. Their work focused on feature coverage for highly configurable software products. The fine-grained minimization scheme distinguishes between totally redundant test cases (the set of features covered by the test case is included in the set covered by another test case) and partially redundant ones (the set of covered features overlap). Only the totally redundant cases are removed. The other ones are assigned a priority that depends on their history of failures, where the test cases that recently failed have a higher priority than the ones that did not. This work is an interesting example of a hybrid strategy combing RTS, TSM and TCP: first the test cases that are relevant to the changed features are selected, then totally redundant cases are removed, and finally test case prioritization is performed to run them in order until the test budget is exhausted.

B. Regression Test Selection

From an RTS perspective, a primary concern is to select test cases that are impacted by a code change. Two recent examples of tools are Ekstazi [8] and STARTS [9], respectively based on a dynamic and static analysis approach. Ekstazi dynamically collects the information about the files accessed by a test case during execution. File checksum verification allows the automated detection of changes. Tests that depend on the changed files are then selected to run. STARTS (STAtic Regression Test Selection) uses information available at compile time to build a dependency graph relating all classes, including the test classes. Like in Ekstazi, the detection of changes is done by checksum comparison and the impacted tests are selected. Beyond these change-based approaches, some authors have proposed to consider other selection criteria, like the similarity of test cases or their history of failures. Chen et al. [10] group tests into clusters based on the similarity of their execution profile. Their selection strategy randomly samples a few tests from each cluster. If any of the selected test fails, all tests in this cluster are also run. Anderson et al. [11] consider the history of failures to select tests that are likely to fail. Two

strategies are proposed. The first one simply selects tests that recently failed. The second one is more complex, by mining relations between test cases: failures in certain subsets of tests are used to determine other subsets that are likely to fail as well.

C. Test cqse Prioritization

The last category of regression testing methods, test case prioritization, seeks the ideal order of tests cases to maximize early fault finding. A number of TCP methods have been proposed in the literature [12]. They differ in the (combination of) criteria used to predict the fault finding capability of the test cases as well in the core algorithms used to build an optimal order. Not surprisingly, the used criteria span many concerns also considered by TSM and RTS: structural coverage [13], relevance to code changes [14] [15], the similarity of tests [15] [16] [17] and their history [17] [18] [19]. The used algorithms are mostly classical optimization ones (greedy, search-based, integrated-linear-programming-based) but may also pertain to information retrieval (IR) and machine learning (ML) [12].

D. Artificial Intelligence

Indeed, techniques from data science and artificial intelligence are attracting a growing interest in the framework of regression testing methods. IR techniques are used to identify the tests impacted by a code change [14]. Clustering allows the identification of similar tests [10] [15]. It may be preceded by a pre-processing of the data for dimensionality reduction [10]. Many ML algorithms have been studied to support decision in regression testing. For example, Busjaeger et al. [20] use a Support Vector Machine method (SVM-map) to learn how to best combine five criteria for test case prioritization. Marijan et al. [7] use C4.5 to predict the effectiveness of test cases from test historical records. Spieker et al. [19] experiment with adaptive TCP strategies based on online reinforcement learning. Lachman [21] compares four machine learning algorithms: software vector machine rank (SVM rank), neural networks, K-nearest neighbor (KNN) and logistic regression. Besides, the author also proposes an ensemble algorithm to combine the four others. From their experiments, logistic regression has the best performance for test case prioritization. An interesting result is also that the features extracted from the test case descriptions – using natural language processing techniques – play a relevant role in the decision: the ML algorithms work better with than without them.

My PhD topic fits into this growing body of research, which investigates AI-based regression testing. While many approaches are being developed, there is no consensus on the combination of features to consider and the learning algorithms to use. My research will tackle the problem with the aim to provide a solution that is well suited for the industrial context at Renault Software Factory.

III. Challenges

In this section, the challenges of the Research Questions are reviewed and discussed.

A. Test Regression Data (RQ1)

As seen in Section II, there are three approaches in regression testing: minimization, selection or prioritization. In each work analyzed, some points are taken into account as selection criteria, such as test execution time, code coverage, test cost, executions history, test age, among others. Therefore, these three methods rely on having testing data to be able to reduce, select or prioritize test cases. To know which data to collect and how to have access to it is a challenge, because not all information is available. Besides that, data is heterogeneous, so it need to be careful analyzed and mined. Thus, another challenge is the data mining, so the right information is considered in the regression test algorithm. It also raises two other questions: how to extract the features - like test age, test level, tests results, test complexity, among others - to be used from the available data? And which ones are more important for choosing a test over another?

B. Artificial intelligence and test selection (RQ2)

As remarked in section II, AI is been implemented to select, prioritize or minimize tests cases. AI algorithms are very useful to find patterns and to classify data, however a lot of data is required to do so. Knowing how much data to use, which type of data is required and what features are the most important for AI processing is a big challenge. Another challenge is to know which algorithm performs better than others in the context considered. There is a comparison between methods in the literature [21] [22], but how to know which is more adapted to the automotive context of Renault Software Factory? Which AI algorithm is more adopted to the available data? Besides that, the timing is critical, it is significant to consider the time for AI to select the tests to run in a CI cycle since each cycle is short. So, the introduction of AI into a CI process needs to be carefully studied.

C. Efficacy of the methods (RQ3)

To verify the effectiveness of an algorithm, it is necessary to compare it with others to find which ones can improve the final result. RQ3 faces the challenge of how to evaluate these regression test selection algorithms and how to compare them. The use of AI makes the algorithm random, resulting in a complexity and high variance probability distribution, thus some statistical tests may be inadequate to analyze the performance of AI algorithms because they are usually based in the assumption of a normal distribution [23]. It may be necessary to adapt tests and evaluation methods to this context. It is also important to know how many experiments are needed and how they can be reliable and true to the CI reality. Besides that, the risk also needs to be measured, because maybe the time can be reduced, but it might increase the risk of not founding the faults presents in the software. Thereby, how the risk of not reveling the faults can be measured?

IV. Work plan

This research work is in its beginning at the moment of writing. The three-year schedule of this thesis can be described as follow:

1^{st} *year:*

- Bibliographical study on software testing in general, the system integration test including "risk management", as well as the existing regression test methods in a context of continuous integration.
- Test data collection, that includes test cases specifications, tests executions and bugs description.
- Analyses of the data features for identifying the set of relevant metrics for selection testing procedure.
- Setting up of the infrastructure to create a test plan, execute it, and take its results.

2^{nd} *year:*

- Improvement of the data mining study.
- Study and implementation of AI algorithms, as well as determination of scenarios for using the learning algorithms in the decision process.

3^{rd} *year:*

- Execution of tests selection methods controlled experiences to evaluate each algorithm.
- Study and implementation of evaluation methods.
- Finalization of experiences and writing of the manuscript.

V. Conclusion

The increased use of technologies and the evolution of the industry bring challenges to be explored and overcome. For software testing, the use of continuous integration methodology makes it possible to test each new version of the code daily. However, the implementation of sort cycles of development-validation-deploy restricts the number of tests to be executed to validate the current version. Selecting the right tests to be executed for each version becomes a challenge wide explored over the years. Furthermore, with the growing wave of the use of artificial intelligence in the technological world, new approaches are been developed.

This work aims to study the use of AI algorithms in the process of selecting test cases to be executed. The expected contribution of this thesis is an algorithm able to take information about test cases and the software under test, process these inputs using learning algorithms to have a list of tests to be executed in the output. Besides that, after the automatic execution of the tests, all the new information about test executions will feedback the algorithm.

Acknowledgment

This project has received funding from Renault Group. The author would like to thank the director Hélène Waeselynck from LAAS and the industrial advisor Fernand Cuesta for their guidance during the first months of this PhD.

References

[1] G. Booch, *Object-Oriented Analysis and Design with Applications (3rd Edition)*. USA: Addison Wesley Longman Publishing Co., Inc., 2004.

[2] M. Fowler, "Continuous integration," May 2006. [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html

[3] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, p. 67–120, Mar. 2012. [Online]. Available: https://doi.org/10.1002/stv.430

[4] R. Singh and M. Santosh, "Test case minimization techniques : A review," *IJERT*, vol. 2, pp. 1048–1056, 12 2013.

[5] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 219–249, 2002. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.256

[6] M. P. E. Heimdahl and D. George, "Test-suite reduction for model based tests: Effects on test quality and implications for testing," in *Proceedings of the 19th IEEE International Conference on Automated Software Engineering*, ser. ASE '04. USA: IEEE Computer Society, 2004, p. 176–185.

[7] D. Marijan, A. Gotlieb, and M. Liaaen, "A learning algorithm for optimizing continuous integration development and testing practice," *Software: Practice and Experience*, vol. 49, no. 2, pp. 192–213, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2661

[8] M. Gligoric, L. Eloussi, and D. Marinov, "Practical regression test selection with dynamic file dependencies," 07 2015, pp. 211–222.

[9] O. Legunsen, A. Shi, and D. Marinov, "Starts: Static regression test selection," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 949–954.

[10] S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng, "Using semi-supervised clustering to improve regression test selection techniques," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, 2011, pp. 1–10.

[11] J. Anderson, S. Salem, and H. Do, "Improving the effectiveness of test suite through mining historical data," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 142–151. [Online]. Available: https://doi.org/10.1145/2597073.2597084

[12] Y. Lou, J. Chen, L. Zhang, and D. Hao, "Chapter one - a survey on regression test-case prioritization," ser. Advances in Computers, A. M. Memon, Ed. Elsevier, 2019, vol. 113, pp. 1–46. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245818300615

[13] G. Rothermel, R. Untch, C. Chu, and M. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[14] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "An information retrieval approach for regression test prioritization based on program changes," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 268–279.

[15] R. Haraty, N. Mansour, L. Moukahal, and I. Khalil, "Regression test cases prioritization using clustering and code change relevance," *International Journal of Software Engineering and Knowledge Engineering*, vol. 26, pp. 733–768, 06 2016.

[16] C. Fang, Z. Chen, K. Wu, and Z. Zhao, "Similarity-based test case prioritization using ordered sequences of program entities," *Software Quality Journal*, vol. 22, 06 2014.

[17] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 2015, pp. 58–68.

[18] Y. Fazlalizadeh, A. Khalilian, M. Abdollahi Azgomi, and S. Parsa, "Incorporating historical test case performance data and resource constraints into test case prioritization," vol. 5668, 07 2009, pp. 43–57.

[19] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 12–22. [Online]. Available: https://doi.org/10.1145/3092703.3092709

[20] B. Busjaeger and T. Xie, "Learning for test prioritization: an industrial case study," 11 2016, pp. 975–980.

[21] R. Lachmann, "12.4 - machine learning-driven test case prioritization approaches for black-box software testing," 01 2018, pp. 300–309.

[22] K. Bhatnagar, "Regression test case selection using machine learning," March 2020. [Online]. Available: https://medium.com/analytics-vidhya/regression-test-case-selection-using-machine-learning-241ded86f559

[23] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Software Testing*, vol. 24, 2014.