An Industrial Application of Test Selection using Test Suite Diagnosability

Daniel Correia
daniel.b.correia@tecnico.ulisboa.pt
Instituto Superior Técnico, University of Lisbon
Lisbon, Portugal

ABSTRACT

Performing full regression testing every time a change is made on large software systems tends to be unfeasible as it takes too long to run all the test cases. The main motivation of this work was to provide a shorter and earlier feedback loop to the developers at OutSystems when a change is made (instead of having to wait for slower feedback from a CI pipeline). The developed tool, MOTSD, implements a multi-objective test selection approach in a C# code base using a test suite diagnosability metric and historical metrics as objectives and it is powered by a particle swarm optimization algorithm. This paper presents implementation challenges, current experimental results and limitations of the developed approach when applied in an industrial context.

CCS CONCEPTS

• **Software and its engineering** → *Software testing and debugging*; Software maintenance tools;

KEYWORDS

test selection, multi-objective, diagnosability, feedback

ACM Reference Format:
Daniel Correia. 2019. An Industrial Application of Test Selection using Test Suite Diagnosability. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3338906.3342493

1 INTRODUCTION AND MOTIVATION

The problem of improving the regression testing process has been extensively studied in academia and in industry. The developed approaches tend to fit into one of three categories: test suite reduction, wherein the focus is on identifying redundant test cases and permanently removing them from the test suite; test case prioritization, where we try to find an optimal ordering of execution for the test cases in order to maximize certain objectives (e.g. number of faults detected for a limited block of time); test case selection [4, 5], which consists in selecting an appropriate subset of the test suite in order to execute only the most relevant tests.

These types of techniques have been applied in industry [1, 2] to successfully reduce regression testing costs, both in terms of computational resources and developer time. However, reducing the diagnostic cost for the developers when a test fails (i.e. the cost of finding the root cause of a test failure and fixing the bug in the code) is typically not addressed by the evaluation metrics that guide these regression testing techniques.

This work targets the reduction of regression testing costs and diagnostic costs in the industrial context of OutSystems[1] wherein the build time of the CI pipeline is a problem due to the large code base and the high execution cost of the test suite. On average, a developer has to wait 40 minutes before receiving relevant feedback on a change, despite the usage of additional computational power and parallelism to speed up the testing process.

Hence, the main motivation of this work was the improvement of the development workflow by providing a faster and earlier feedback loop to the developers when a change is made. This would in turn motivate further improvements in OutSystems development processes such as pre-commit validations and migration to Git.

This paper describes the test selection approach developed in the industrial context of OutSystems and highlights several problems encountered during its implementation. Additionally, current experimental results are presented and discussed in terms of the following objectives: (1) the selected tests should correspond to the failing tests; (2) the selection process should be quick enough to be integrated into a development workflow; (3) the size of the selected subset of tests should be small enough to provide a much faster feedback loop than the original process (i.e. the CI pipeline).

2 APPROACH

Summary. The developed test selection approach was implemented in a tool called MOTSD [2] which uses a test suite diagnosability metric (DDU [3]) and historical metrics to guide the selection process. This is modelled as a multi-objective problem and a particle swarm optimization algorithm is used to find good solutions. The activity matrix (i.e. coverage data) is filtered according to the changes introduced by the commit using file-level granularity.

OutSystems Context. OutSystems' code base is built on top of a C# stack across one million lines of code and is accompanied by a few modules implemented in Typescript. This means that even though most of the files changed during development will be C# files, there is a significant amount of changes to other types of files (e.g Typescript, XAML, XML, JSON, CSS). The test suite used by OutSystems contains over 8500 tests implemented in NUnit.

[1]Introducing OutSystems blog post: https://www.outsystems.com/blog/posts/introducing-outsystems-11/ (accessed May 2019)
[2]https://github.com/danielcorreia96/MOTSD (accessed May 2019)

1-Month Period	# Commits	# Tool Executions		Tool Execution Errors			Tool Found Failing Tests?			Solution Size				Average Selection Time (s)	Average Feedback Time (s)
		Total	Total %	# No .cs Files	# No Cov. Data	# New Files	Yes, at least one	Precision	Recall	Average	Min	Max	Std		
July 2018	469	237	51%	71	146	15	32%	1%	22%	904	2	2104	744	12	801
October 2018	336	116	35%	67	133	20	31%	0%	21%	1210	5	2595	865	14	1211
February 2019	463	111	24%	126	196	30	36%	1%	23%	1447	8	2737	994	19	1211
March 2019	579	174	30%	203	153	49	46%	1%	26%	1639	3	2536	951	17	1140

Table 1: Results obtained for a set of experiments (four distinct 1-month periods) on the OutSystems code base. The objectives used for optimization were (1) Maximize DDU and (2) Maximize Total Test Failures.

Coverage Data Extraction. For coverage data collection from the C# code base, the open source tool OpenCover[3] was used. However, since OpenCover uses fine grained profiling, the overhead of extracting coverage from OutSystems' large code base and test suite is too high. To handle this issue, several optimizations were made[4] to minimize the size of the generated coverage reports and instrumentation overhead. These modifications led to a reduction of 56% in execution time (from 32 minutes to 14 minutes) and 94% in coverage report size (from 2855 MB to 168 MB).

Test Selection Pipeline. The core component of MOTSD is the test selection pipeline that, given a set of code changes from a commit, provides a subset of tests to be executed based on coverage data and historical metrics mined from build history. The pipeline consists of three main steps: (1) filter the coverage data according to the files changed in the commit; (2) obtain solutions for the multi-objective problem; (3) return a solution (i.e. a selected subset of tests) using total ordering on the objectives.

3 RESULTS

Table 1 shows the results obtained for 4 different 1-month periods of development using a test suite of around 8000 tests. The performance of the approach was evaluated using solution size, selection time, feedback time and recall. The recall metric was chosen since this problem is similar to an Information Retrieval problem where a query is provided (i.e commit) to the system and it returns a set of documents (i.e. set of selected tests) that are relevant. Hence, recall evaluates how many failing tests are selected.

Limitations. The results highlight several limitations when applied to an industrial context both in terms of how often it can produce results (on average, MOTSD executes on less than 50% of the commits) and how often it can actually find failing tests. Regarding cases where the tool does not execute, 3 error cases were detected: (1) the commit changed no C# files (i.e. .cs files); (2) the activity matrix did not have coverage data for the changed files; (3) the commit only added or changed new files.

The first error is caused by the fact that the activity matrix only has coverage data from C# code. The second error stems from the assumption that the activity matrix covers all of the system's components, which is unrealistic since covering all possible code execution paths in a large system is unfeasible. The third error reveals a lack of adaptability to new files, which could be solved by frequently updating the activity matrix. However, this is impractical due to the high overhead of generating a new activity matrix.

Performance and Objectives Regarding the ability to select failing tests, MOTSD's performance is lower than would be desirable (max. 26% recall) and current investigation work has shown that in most cases either the commit was innocent (i.e. tests were broken by a previous commit) or there was no coverage data available to perform the desirable selection. In addition, if we consider that finding at least one of the failing tests is enough for the selection to be successful, then the results are a bit more positive (reaching a max of 46%).

MOTSD is fast enough to be integrated into existing development workflows and it supports the initial motivation of shorter feedback loops since the average selection time is under 10 seconds and the number of selected tests is much smaller than the original test suite. This is supported even further by the newly observed feedback time, i.e. the time to actually execute the selected tests and receive feedback on them. On average, MOTSD was able to provide a feedback time of 1200 seconds (20 minutes). This is significantly better than the original feedback time - 90 minutes.

Future Work. This work revealed several concerns regarding applicability to cross-language code changes and the scalability bottleneck of coverage data extraction. One possible improvement would be to use some other source of data than coverage information. For example, the activity matrix could be modelled using a dependency graph built by analyzing the version history and linking files that are changed in the same commit[5]. This graph could be further enriched with build history information by linking changed files with tests that failed in the respective commit.

4 CONCLUSIONS

This paper presents a multi-objective test selection approach developed in the industrial context of OutSystems. The developed tool uses a recently proposed test suite diagnosability metric, DDU, instead of a classical code coverage metric in order to tackle the diagnostic cost of the selected subset of tests. Several challenges were revealed when implementing this type of tool in a large scale project both in terms of the code coverage tools overhead and the limitations of the proposed test selection approach.

ACKNOWLEDGMENTS

This work was supported by OutSystems under a master thesis collaboration project.

[3]https://github.com/OpenCover/opencover (accessed May 2019)

[4]Comparison of changes with original OpenCover: https://github.com/OpenCover/opencover/compare/4.6.519...danielcorreia96:oc_2016_merged (accessed May 2019)

[5]For example, this could be done using hercules https://github.com/src-d/hercules

REFERENCES

[1] Mateusz Machalica, Alex Samylkin, Meredith Porth, and Satish Chandra. 2018. Predictive Test Selection. *CoRR* abs/1810.05286 (2018). arXiv:1810.05286 http://arxiv.org/abs/1810.05286

[2] Atif Memon, Zebao Gao, Bao Nguyen, Sanjeev Dhanda, Eric Nickell, Rob Siemborski, and John Micco. 2017. Taming Google-scale Continuous Testing. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP '17)*. IEEE Press, Piscataway, NJ, USA, 233–242. https://doi.org/10.1109/ICSE-SEIP.2017.16

[3] Alexandre Perez, Rui Abreu, and Arie van Deursen. 2017. A Test-suite Diagnosability Metric for Spectrum-based Fault Localization Approaches. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 654–664. https://doi.org/10.1109/ICSE.2017.66

[4] Gregg Rothermel and Mary Jean Harrold. 1996. Analyzing Regression Test Selection Techniques. *IEEE Trans. Softw. Eng.* 22, 8 (Aug. 1996), 529–551. https://doi.org/10.1109/32.536955

[5] Kaiyuan Wang, Chenguang Zhu, Ahmet Celik, Jongwook Kim, Don Batory, and Milos Gligoric. 2018. Towards Refactoring-aware Regression Test Selection. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 233–244. https://doi.org/10.1145/3180155.3180254