# Improving Continuous Integration with Similarity-based Test Case Selection

Francisco G. de Oliveira Neto
Chalmers | University of Gothenburg
Gothenburg, Sweden
gomesf@chalmers.se

Azeem Ahmad
Linköping University
Linköping, Sweden
azeem.ahmad@liu.se

Ola Leifler
Linköping University
Linköping, Sweden
ola.leifler@liu.se

Kristian Sandahl
Linköping University
Linköping, Sweden
kristian.sandahl@liu.se

Eduard Enoiu
Mälardalen University
Västerås, Sweden
eduard.paul.enoiu@mdh.se

## ABSTRACT

Automated testing is an essential component of Continuous Integration (CI) and Delivery (CD), such as scheduling automated test sessions on overnight builds. That allows stakeholders to execute entire test suites and achieve exhaustive test coverage, since running all tests is often infeasible during work hours, i.e., in parallel to development activities. On the other hand, developers also need test feedback from CI servers when pushing changes, even if not all test cases are executed. In this paper we evaluate similarity-based test case selection (SBTCS) on integration-level tests executed on continuous integration pipelines of two companies. We select test cases that maximise diversity of test coverage and reduce feedback time to developers. Our results confirm existing evidence that SBTCS is a strong candidate for test optimisation, by reducing feedback time (up to 92% faster in our case studies) while achieving full test coverage using only information from test artefacts themselves.

## CCS CONCEPTS

• **Software and its engineering → Software verification and validation**; *Software maintenance tools*; *Maintaining software*;

## KEYWORDS

Similarity based test case selection, Continuous integration, Automated testing

**ACM Reference Format:**
Francisco G. de Oliveira Neto, Azeem Ahmad, Ola Leifler, Kristian Sandahl, and Eduard Enoiu. 2018. Improving Continuous Integration with Similarity-based Test Case Selection. In *AST'18: AST'18:IEEE/ACM 13th International Workshop on Automation of Software Test , May 28–29, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194733.3194744


*AST'18, May 28–29, 2018, Gothenburg, Sweden*

ACM ISBN 978-1-4503-5743-2/18/05…$15.00
https://doi.org/10.1145/3194733.3194744

## 1 INTRODUCTION

Automated test optimisation has been widely researched in academia, due to their successful cost reduction by minimising, selecting or prioritising test cases [21]. There has been a recent increase in the number of studies using similarity/distance information from test cases to guide the optimisation. Numerous techniques improve diversity of varied information such as execution traces [17], model elements [4, 7, 12], test input [9], or manual tests [13, 15] across the different levels of testing.

In a recent study, Henard et al. state that the similarity-based techniques, along with combinatorial integration testing, perform the best among the black-box testing techniques [14]. But what are the key elements behind similarity-based test optimisation? The underlying assumption is that, when testing a software system, we should search for a diverse set of test cases [9, 10, 13] that are able to exercise several yet distinct parts of the code base. This ensures breadth in coverage, thus allowing the techniques to remove similar or very redundant test cases.

Then, optimisation becomes a candidate solution for companies aiming to lever their development processes by reducing redundancies in sets of test cases, while catering for high test coverage. On the other hand, discarding test case is an inherent risk of optimisation approaches [21] since defects can remain undetected when the test case supposed to reveal such defect is not exercised. Therefore, practitioners should be aware of the trade-offs when deciding to adopt automated optimisation in their test processes.

Alternatively, one can instrument continuous integration (CI) and/or delivery (CD) pipelines to provide automated and continuous testing thus mitigating the risks caused by discarding test cases. This accelerates development, since developers should receive feedback from integration level testing on built packages of the system under test (SUT) [2]. This is currently automated in servers instrumented with, for instance, Jenkins[1] or Bamboo[2].

Therefore, our goal is to investigate the synergies between CI and similarity-based test optimisation. We collect data from two companies from distinct domains and evaluate the trade-offs of introducing similarity-based test case selection (SBTCS) to CI pipelines. The

[1] https://jenkins.io/
[2] https://www.atlassian.com/software/bamboo

selection is performed on automated integration-level test cases, which are specified in natural language[3].

In a case study, we compare the performance of three different similarity functions and random test selection, with respect to two dependent variables: test coverage and time to execute the selected subset. The selection uses only the test cases themselves, such that no additional software artefact is needed such as the source code or requirements specification. Results reveal that SBTCS is able to significantly reduce the time (e.g., up to 92%, from 3.75 hours to 17 minutes) required to execute integration tests on CI servers while achieving full test coverage of different criteria from test artefacts (test requirements, dependencies and steps). This helps developers to receive faster feedback when validating builds of the SUT, during ongoing development activity.

On the other hand, limitations in our datasets hindered analysis of failure detection, hence limiting our conclusions about the effectiveness of the obtained time reduction. Nonetheless, in our case studies, the entire test suite is executed every day during the night on the CI servers, so that developers have information about detected defects at the beginning of each working day.

This paper has the following structure. Section 2 provides a summary of existing techniques and studies involving SBTCS. Due to limited space, we do not present details on how the similarity measures are calculated, or how the selection is performed; instead we refer to Hemmati et al. [12] and Cartaxo et al. [4] for a detailed description of the SBTCS. Section 3 presents our case study, such as our planning, variables, factors, and details about the case companies. Results and validity threats are then discussed on Section 4, followed by Section 5 where we summarise our findings and answer the research questions. Lastly, where we draw conclusions and discuss future work.

## 2 BACKGROUND AND RELATED WORK

Similarity-based test case selection has been reported as a promising test optimisation approach for white-box and black-box approaches [5, 13, 14, 17]. The goal is to remove similar test cases in order to increase diversity of the test set, based on the underlying assumption that similar test cases exercise similar parts of the source code, hence being more likely to reveal the same defects [4, 13]. Therefore, by reducing the size of test suites and increasing diversity, we are likely to execute fewer test cases and still uncover a variety of defects.

In order to measure the similarity between two test cases, we use similarity measures, also named *similarity functions*. A similarity function receives two pieces of information as input (two test cases in our context) and returns a number (often between 0 and 1) indicating the degree of similarity between the two inputs. When two test cases have similarity 0 they are completely different test cases, whereas a similarity value of 1 indicates identical test cases. Note, however, that there is a continuum between 0 and 1, resulting in a scale, instead of a binary attribute. So, we would like to have a set of test cases with low similarity values. Alternatively, some techniques use the concept of distance between test cases [9, 10] which can be simply a complement to the similarity, i.e. $distance(T_i, T_j) = 1 - similarity(T_i, T_j)$.

Considering that the similarity is measured between pairs of test cases, the techniques usually calculates the similarity between all pairs of test cases, resulting in a similarity matrix, from which the test cases are then automatically selected by *i*) identifying the most similar pair of test cases (e.g., highest value in the matrix), and then *ii*) removing one of those two test cases. The matrix is then updated by removing information from the discarded test, and the next most similar pair is identified, hence repeating the process until the matrix is left with the desired amount of test cases [3, 4, 12].

Note that similarity can be measured in terms of different properties of a test case, such as textual steps [3, 4, 12], modifications [7], and historical failure information [17]. Therefore, one of the main elements of similarity-based test optimisation is the choice of an appropriate function able to quantify the similarity between two test cases in terms of those different properties.

Cartaxo et al. [4] and Hemmati et al. [12] investigated similarity-based test case selection in the context of model-based testing (MBT). Their techniques aim to increase diversity of model elements, such as transitions or states, that represent steps or conditions in abstract test cases. Similarly, de Oliveira Neto et al. [7] evaluated SBTCS when modifications are performed on those models, so that regression test suites include a diversity of test cases covering all model modifications. Their experiments show that SBTCS techniques perform better (especially when compared to random) in terms of coverage and defect detection, and argue that the similarity functions are quite general and applicable beyond MBT [4, 7, 12]. Our case study confirms their findings since SBTCS has good performance, even if model information is not provided.

Other studies use similarity functions to prioritise tests, instead of selecting them [13, 17]. In their case study with open source data, Hemmati et al. reveal that, for black-box testing, risk-driven selection outperforms SBTCS for rapid releases of software in terms of average percentage of fault detected (APFD), a widely used metric to evaluate test prioritisation techniques [18]. Their findings are limited to black-box approaches and test prioritisation which contrast to our study that focus on integration level test selection.

In connection to the performance of SBTCS, there are several studies evaluating the use of different similarity functions for test case selection [5, 12]. Results show that distinct functions perform significantly different in finding defects [5, 7] depending on the type of information that they are employed. For instance, the Levenshtein function [5, 12] is based on the edit distance between two string, thus being suitable for capturing similarity on text-based test scripts, whereas Jaccard Index [5, 12] is suitable for sets of test information, and the Normalised Compression Distance (NCD) [9] is more general and can be used on any type of data. Our case study confirms the differences in performance among the different similarity functions when using different types of information.

## 3 CASE STUDY

We perform a case study with the objective of exploring the benefits of using similarity-based test case selection to leverage continuous integration by providing fast and meaningful feedback on integration test activities. This case study was performed with data

[3]The test case descriptions are written using natural language but later in the CI pipeline, test frameworks are used to automatically execute the test cases

collected from two companies located in Sweden. The first is a security and video surveillance company, whereas the second is an automotive company.

Since this is an exploratory case study, our objective, at this stage, is not to assess the quality of their CI pipelines and activities, rather we use our results to describe each case and point to particular aspects of their automated testing processes that can be improved. Therefore, we focus on the following research questions:

RQ1: How can SBTCS lever test feedback on CI pipelines?
RQ2: What are the trade-offs in selecting and executing fewer test cases during software builds?
RQ3: Is there a difference between similarity functions in selecting integration-level automated test cases?

We visited each company to interview testers and understand the relations between the CI pipeline and their test activities to answer RQ1. In addition, practitioners exported test data from their CIs, including test case specifications, execution logs, and build information (time stamps, executed test cases, etc.) which we use to answer RQ2 and RQ3. Since each company provided specific archival data, we consider two distinct units of analysis to draw separate conclusions and then discuss on the overall results.

In this case study we consider three distinct attributes in a test case: *test requirements*, *test dependencies*, *test steps*. A test requirement is an approval criterion, and therefore a functional assessment of the SUT. An example of a test requirement is a system requirement that must be validated (e.g., a hardware element should respond properly to network commands). In turn, test dependencies represent functionality required for test execution, but excluded from the approval criteria. In other words, they are setup elements required for test execution. Note that in practice we should distinguish test cases that fail due to a defect from test cases that fail due to a setup error (e.g., the hardware element being offline). The problem with test dependencies is that one failure may cause a chain reaction on the build process, and developers are unable to validate their implementation due to problems, for instance, in the environment setup. Lastly, test steps are the natural language description of user actions and expected results created by a tester. Since these are integration-level tests, this natural language description is eventually translated into executable test code.

We evaluate the different test selection techniques based on the coverage of test requirements, dependencies and steps in relation to the reduction of the test suite. The goal is to see whether fewer test cases can be executed without reducing coverage. In other words, we want to see how far can we reduce sets of test cases and still provide sustainable coverage to ensure thorough test results. Additionally, we analyse how the test selection affects the execution time on the CI servers. In order to measure effectiveness, one should ideally measure how many defects the selected subset reveals. However, the data provided had limited and incomplete failure information. Hence, we cannot draw conclusive results regarding defect detection rates of the investigated techniques. A summary of our case study planning is presented in Table 1.

**Table 1: Our case study planning according to guidelines presented by Runeson and Höst [19].**

| | |
|---|---|
| Objective | Explore |
| The context | Automated testing in CI pipelines |
| The cases | Surveillance Company<br>Automotive Company |
| Theory | Similarity-based test case selection. |
| Research questions | RQ1, RQ2 and RQ3 |
| Methods | Third-degree data collection:<br>Archival data and metrics |
| Selection strategy | Companies with instrumented CI pipelines |
| Unit of Analysis 1: | Surveillance Company:<br>Coverage and Time |
| Unit of Analysis 2: | Automotive Company:<br>Coverage |

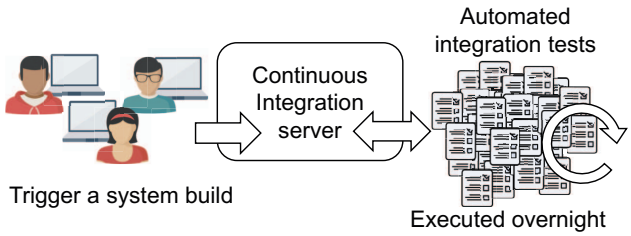



**Figure 1: Summary of the CI process in the case companies.**

## 3.1 Case companies

We conducted interviews with both companies to understand their CI activities and tools, such as how do they automate their integration tests, the bottlenecks in test executions, if and how test suites are optimised (i.e., selected or prioritised), and how do developers get feedback from this test infrastructure. For both companies, we identified a similar process, described in Figure 1.

Integration test cases are pushed to a continuous integration server that runs tests whenever building a new version of the SUT. Across its life-cycle, the test suite has grown to an extent where the execution of all test cases takes a long time (e.g., several hours), such that developers often have to optimise the test execution by selecting a subset of test cases. Alternatively, the entire test suite executes overnight.

Due to Non-Disclosure Agreements (NDA), we cannot present examples of test cases and coverage criteria provided by the companies. Instead, a summary of the data collected from both companies is given in Table 2. Since the data from each company include different information, we divide our analysis into two distinct unit of analysis, as suggested by Runeson and Höst [19].

Note that the test cases do not have a one-to-one coverage relationship with requirements, dependencies and steps. In fact, there is a lot of redundancy in the test sets, since the same requirements or dependencies are covered by different test cases. Therefore, we aim to increase diversity in coverage while removing very similar test cases.

The practical significance of our investigation is to support developers in receiving faster feedback by testing their implemented

**Table 2: Summary data from test artefacts collected from the case companies.**

| Property | Surveillance Company | Automotive Company |
|---|---|---|
| Test cases: | 1096 | 1972 |
| Test requirements: | 158 | — |
| Test dependencies: | 384 | — |
| Test steps[4]: | — | 1093 |
| Total execution time: | 225 minutes (3.7 hours) | — |

changes on the built version of the system. Therefore, instead of waiting for the entire overnight execution, we integrate the SBTCS to the CI server, and allow automatic selection of a smaller and diverse subset of test cases. Developers are then provided with faster feedback on integration and system level tests. Note that we still expect testers to test their modifications locally before pushing source code to the CI server.

## 4 RESULTS

We evaluate the selection techniques in terms of percentage of covered requirements, dependencies and steps as we reduce the size of the test suite by 5%. This allows us to see how coverage is affected as we execute fewer test cases.

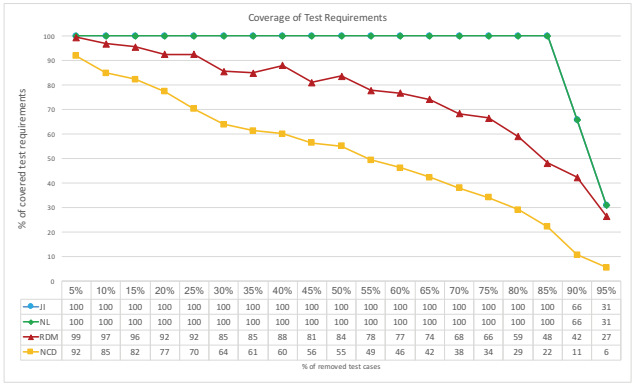
Our factor is the *selection technique*, which we assign *four levels*. We chose random selection (RDM) as a control group, whereas the remaining three levels are different similarity functions used in studies reported in literature [5, 9, 12, 13, 17]: Normalised Levenshtein (NL), Jaccard Index (JI) and Normalised Compression Distance (NCD).

### 4.1 Unit of Analysis 1

Our results for Unit of Analysis 1 are presented in Figure 2. When considering coverage of test requirements and dependencies, both the JI and NL techniques outperform RDM and NCD by providing higher and more sustainable coverage as we reduce the number of test cases. Note that both techniques are able to remove up to 85% (931/1096) of test cases and still cover all distinct test requirements (Figure 2(a)). In other words, even if a developer is working locally in an isolated test requirement, she can push her modifications to the CI server and get feedback from all test requirements by executing only 15% of the test suite.

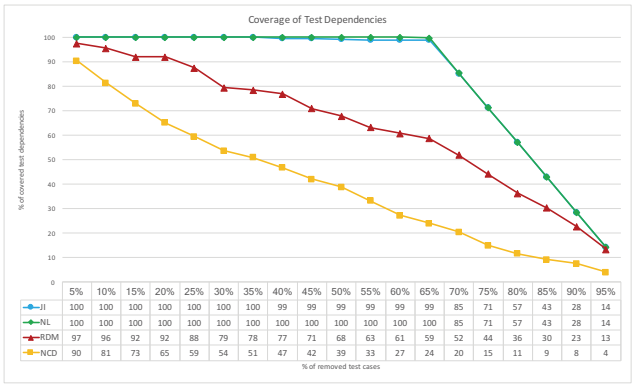
Similarly, when considering test dependencies, NL and JI outperform the other selection techniques. The main distinction between test requirements and dependencies is that each test covers a single test requirement, whereas each test case may have several dependencies. Consequently, more redundancy is expected on test dependencies. Nonetheless, JI and NL were able to remove up to 65% (712/1096) of test cases and still cover all distinct combinations of test dependencies required to execute the test cases. That means that a developer can test all combination of test setups, in the integrated system, with many less test cases.

Surprisingly, NCD has shown inferior results when compared to RDM. Studies indicate that the compression can hinder NCD when used on small strings [9]. The test requirements in our dataset are short strings, which may actually be a benefit for NL since the edit



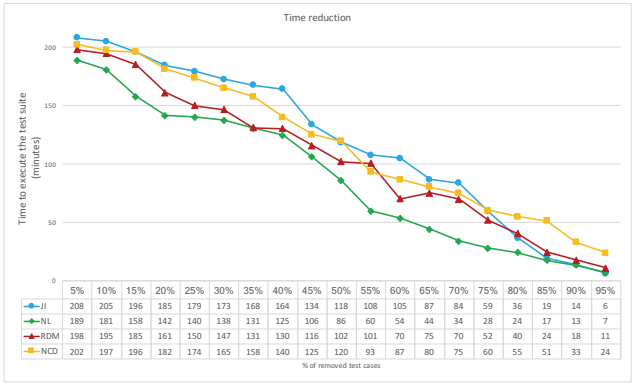

| | 5% | 10% | 15% | 20% | 25% | 30% | 35% | 40% | 45% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JI | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 66 | 31 |
| NL | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 66 | 31 |
| RDM | 99 | 97 | 96 | 92 | 92 | 85 | 85 | 88 | 81 | 84 | 78 | 77 | 74 | 68 | 66 | 59 | 48 | 42 | 27 |
| NCD | 92 | 85 | 82 | 77 | 70 | 64 | 61 | 60 | 56 | 55 | 49 | 46 | 42 | 38 | 34 | 29 | 22 | 11 | 6 |

(a)




| | 5% | 10% | 15% | 20% | 25% | 30% | 35% | 40% | 45% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JI | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 99 | 99 | 99 | 99 | 99 | 99 | 85 | 71 | 57 | 43 | 28 | 14 |
| NL | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 85 | 71 | 57 | 43 | 28 | 14 |
| RDM | 97 | 96 | 92 | 92 | 88 | 79 | 78 | 77 | 71 | 68 | 63 | 61 | 59 | 52 | 44 | 36 | 30 | 23 | 13 |
| NCD | 90 | 81 | 73 | 65 | 59 | 54 | 51 | 47 | 42 | 39 | 33 | 27 | 24 | 20 | 15 | 11 | 9 | 8 | 4 |

(b)



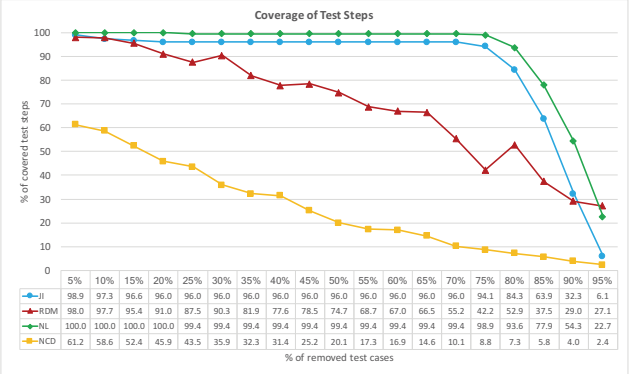

| | 5% | 10% | 15% | 20% | 25% | 30% | 35% | 40% | 45% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JI | 208 | 205 | 196 | 185 | 179 | 173 | 168 | 164 | 134 | 118 | 108 | 105 | 87 | 84 | 59 | 36 | 19 | 14 | 6 |
| NL | 189 | 181 | 158 | 142 | 140 | 138 | 131 | 125 | 106 | 86 | 60 | 54 | 44 | 34 | 28 | 24 | 17 | 13 | 7 |
| RDM | 198 | 195 | 185 | 161 | 150 | 147 | 131 | 130 | 116 | 102 | 101 | 70 | 75 | 70 | 52 | 40 | 24 | 18 | 11 |
| NCD | 202 | 197 | 196 | 182 | 174 | 165 | 158 | 140 | 125 | 120 | 93 | 87 | 80 | 75 | 60 | 55 | 51 | 33 | 24 |

(c)

**Figure 2: Results from test selection using data from the Surveillance Company. Test requirements (a) and dependencies coverage (b) and time reduction (c).**

distance provides a fine-grained similarity value. However, we are only able to draw conclusive results when executing the techniques on a variety of datasets.

In terms of time (Figure 2(c)), the first conclusion is that none of the techniques affect time reduction, since, as expected, time




| | 5% | 10% | 15% | 20% | 25% | 30% | 35% | 40% | 45% | 50% | 55% | 60% | 65% | 70% | 75% | 80% | 85% | 90% | 95% |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| JI | 98.9 | 97.3 | 96.6 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 96.0 | 94.1 | 84.3 | 63.9 | 32.3 | 6.1 |
| RDM | 98.0 | 97.7 | 95.4 | 91.0 | 87.5 | 90.3 | 81.9 | 77.6 | 78.5 | 74.7 | 68.7 | 67.0 | 66.5 | 55.2 | 42.2 | 52.9 | 37.5 | 29.0 | 27.1 |
| NL | 100.0 | 100.0 | 100.0 | 100.0 | 99.4 | 99.4 | 99.4 | 99.4 | 99.4 | 99.4 | 99.4 | 99.4 | 99.4 | 99.4 | 98.9 | 93.6 | 77.9 | 54.3 | 22.7 |
| NCD | 61.2 | 58.6 | 52.4 | 45.9 | 43.5 | 35.9 | 32.3 | 31.4 | 25.2 | 20.1 | 17.3 | 16.9 | 14.6 | 10.1 | 8.8 | 7.3 | 5.8 | 4.0 | 2.4 |

**Figure 3: Results from Automotive company regarding coverage of test steps as we select fewer test cases.**

decreases linearly as we remove test cases. On the other hand, by combining results from time and coverage we see the benefits of SBTCS in CI pipelines. Note that the entire test suite executes overnight, since developers cannot wait, on average, 225 minutes to receive feedback from testing on a system build. In practice, Figure 2(a) shows us that a developer can validate all test requirements with only 15% of the test cases, which requires 17 minutes to execute. In case developers want to make sure all of the test setup *and* requirements are exercised at least once, Figure 2(b) shows us that an average of 44 minutes is needed, by executing only 35% of test cases. In addition to the benefits of full coverage with less time, the selection of test cases (i.e., up to 80% using NL, JI and NCD), itself, with respect to test requirements and test dependencies takes less than one minute (an average of 20 seconds on our implementation). We believe that scalability is feasible, since they run on quadratic time[5] because of the similarity matrices, but we plan to explore further the extent of that scalability in future work.

Unfortunately, we could not assess this size reduction with respect to failure detection, since the provided data lacked reliable defect information. Certainly there is the risk that some of the discarded tests would reveal important defects, but remember that all of the test cases will still execute overnight. Therefore, developers will have feedback on exhaustive coverage the next day, in case some of the defects slip through.

## 4.2 Unit of Analysis 2

Our results for Unit of Analysis 2 are presented in Figure 3. Unfortunately, the case company could only provide information on test steps at this point. Nonetheless, the results are insightful, particularly when comparing the results between both units of analysis.

Similarly to results from Surveillance Company, NL outperforms the remaining similarity functions by providing high and sustainable test step coverage. Note that, for Automotive, test cases are also integration-level test cases described using natural language. Figure 3 shows that NL can consistently cover 99% (1082/1093 steps) of distinct test steps even after removing 75% (1479/1972) of the most similar test cases. If compared to the random selection at this threshold of removal, more than half (58% = 634 steps) of unique test steps would be discarded. In turn, steps are short strings hindering again NCD's performance.

Unfortunately, we lack complementary results from Automotive (e.g., time and failure) to contrast our coverage results. Nonetheless, the value is again in the prospects in using SBTCS to allow developers to get faster feedback from CI servers with reliable and thorough test coverage.

[5]For instance, Marzal and Vidal [16] show that for strings of length $m$ and $n$, NL seems to have time complexity $O(m * n^2)$.

# 5 DISCUSSION

One of the main shortcomings of our case study is the focus on coverage, without contrasting information on detected defects. Certainly, coverage information alone leads to limited conclusions, since we cannot assess the quality of the coverage achieved without seeing how many defects are detected. On the other hand, we focus on CI and the time required to provide test feedback from CI servers back to develop. In practice, developers often perform and test their changes locally, but those changes cannot be integrated in the master branch unless they pass all of the tests on a CI build.

The value of our approach is not to replace the overnight exhaustive test execution. Rather, we aim to provide developers with a preview of that execution, allowing them to prioritise code changes (i.e., which modifications should be pushed to the master branch) throughout their development day. Note that for the Surveillance Company, the SBTCS using NL reduced the feedback time by 92% and 80% while covering the complete sets of test requirements and setup configurations respectively.

In addition, developers can choose to be even safer by selecting higher thresholds for subset sizes (e.g., 50% of test cases) that take an average of one hour to execute which developers can spare, for instance, during lunch time. So, even though there is a benefit in having thorough coverage, we also enable control of systematic testing using the CI servers. That allows stakeholders to reduce risks with redundant test cases, which is a benefit seen in other studies using SBTCS [4, 5, 7]. Based on the data analysis and the discussion, we answer our research questions.

Another interesting finding is the consistence of the SBTCS selection techniques, even though they were used in different case companies. In fact, the results align with findings from other studies SBTCS that shows coverage advantages even after removing more than half of the test suite [4, 7]. For instance, in one of their case studies, Cartaxo et al. reduced by 50% the test execution time while still exercising all of the model's transitions, i.e., full test coverage. They used however a different similarity function, named a basic counting function [3, 4, 7, 12, 17], whereas our best results are with the Normalised Levenshtein function.

*RQ1: How can SBTCS lever test feedback on CI pipelines?* Our case study shows that SBTCS provides stakeholders with a sustainable test coverage even when test sets are significantly reduced. Therefore, we can instrument CI servers to include a test selector interface where developers can input a desired subset size that suits their time constraints. The goal is to execute fewer test cases, so that developers can get faster feedback on integration tests before, for instance, pushing their code to a master branch.

*RQ2: What are the trade-offs in selecting and executing fewer test cases during software builds?* Test selection techniques in general are susceptible to the risk of discarding test cases, since ideally one should execute all the test cases [21]. Even though SBTCS provides complete coverage even with smaller subsets, the complete coverage does not imply that all defects will be detected. In summary, the trade-off is to, on one hand, decrease the feedback time with sustainable coverage, while, on the other, mitigate the risk of discarding test cases. Our approach assumes that the entire test suite is executed overnight. Hence, failures should be reported the next day so that developers can debug and revert their changes if necessary. Unfortunately, our data lacked failure information to properly assess this conclusion, but we are currently investigating these trade-offs with the case companies by collecting more data.

*RQ3: Is there a difference between similarity functions in selecting integration-level automated test cases?* Yes. Our results from both case studies confirm that similarity functions perform differently depending on the type of property analysed in the test case [5]. For instance, NCD performed worse than NL and JI since data extracted from all of the three properties (steps, requirements and dependencies) were small string descriptions. We believe that using longer strings (e.g., a concatenation of all three properties) can yield better results with NCD. Alternatively, the conclusions would be different since the properties would not be evaluated in isolation anymore.

One of the caveats when using string distances is that they are based only on lexicographic information, and do not capture semantics [15]. For instance, in our case studies, both JI and NL performed similarly with very few differences in terms of coverage. Coutinho et al. measured a statistically significant difference between JI and NL on their three case studies [5], which contrast to our results. This contrasting conclusion can be attributed to the labels in the test cases. When looking at the data provided by authors[6], we noted that their test descriptions are transition labels, from synthesised models [6] using two real applications as references. Consequently, their test sequences are arbitrary labels (e.g., "A", "AA", "ABX"), whereas ours are actual natural language descriptions written by a tester when designing the test case. The semantic content of the test cases is not captured in any of the approaches (ours included), but lexicographic information such as length or the different labels used can yield differences in performance. Therefore, we intend to include more similarity functions and distinct artefacts in future work, in order to collect more evidence regarding the trade-off in applying different similarity functions on a variety of artefacts.

Despite the limitations in our data, the analysis show consistent benefits when using SBTCS in CI pipelines. In addition, this is an exploratory study with practitioners, with whom more promising studies are planned. We hope to collect more data and include companies from a variety of domains, to better understand the boundaries of the applicability of SBTCS.

We aim to foster technology transfer of our approach by using the technology transfer model introduced by Gorschek et al. [11], composed of 7 steps (Figure 4). Our case study includes results from

[6] http://sites.google.com/site/distancefunctions/

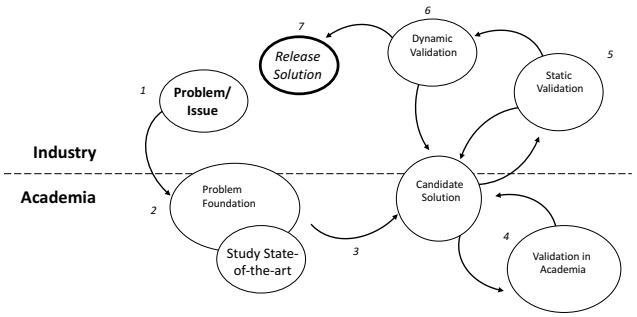



**Figure 4: Technology transfer model proposed by Gorschek et al. [11].**

*step 4* (validation in academia), and studies are currently moving towards *step 5* (static validation), where we intend to introduce the candidate solution to stakeholders (e.g., testers and developers) and perform interviews, to assess our instrumentation of the CI servers.

## 5.1 Threats to validity

We discuss threats to validity in terms of conclusion, construct, internal and external validity threats, in accordance to empirical software engineering guidelines [20]. The different types of validity explore distinct limitations in our case study, for each we present a mitigating strategy.

Our construct validity is limited by the choice of dependent variables. Ideally, test optimisation is assessed in terms of failure information, even if mutation is employed on the source code to indicate defect detection [1]. However, our data did not include source code or reliable failure information. We mitigate this threat by limiting our findings to coverage and time, which are two desired properties in any CI pipeline [8]. Additionally, we chose only a limited set of similarity functions as levels to our factor. We argue that including more similarity functions, at this point, would not add much value to our findings, particularly since our case study is exploratory with respect to SBTCS applied to CI.

Most of the internal validity threats are related to the execution of the test selection (i.e., implementation of the techniques), as well as the consistency of data collected from practitioners. We mitigate the first threat by thoroughly testing the implementations of the similarity functions with unit tests, whereas the second threat comprise data automatically generated by CI servers that monitor test activity. Nonetheless, two of the authors checked the data for inconsistencies, such that the inconsistencies of failure data were detected in those sessions.

Our conclusion validity is connected to the findings regarding our research questions. We use actual time and coverage data exported from CI servers with our collaborating practitioners, so we mitigate the threat to draw conclusions on outdated information. In addition, we presented and validated the results to practitioners that confirmed our findings. Nonetheless, this is an exploratory case study, which, at this point, discouraged us to pursue statistical tests to compare the different levels of our factor. However, a more thorough quantitative analysis is planned for future work.

Lastly, the external validity of our case study is limited by the amount of companies involved. Once more practitioners join the studies, we can extend the reach of our findings by including more CI pipelines and different types of test cases. However, such investigation would be risky without the initial results, presented in this paper. Still, we use actual data from industry whereas other studies have similar findings using data from open source projects [13, 17] or synthesised models [5, 6].

## 6 CONCLUDING REMARKS

In this paper we report on a case study that investigates the trade-offs in using similarity-based test case selection in continuous integration pipelines. We use three different similarity functions (Jaccard Index, Normalised Levenshtein, and Normalised Compression Distance) on integration level test cases described in natural language. The test cases automatically check builds of a system under test on CI servers.

Our results with two industrial partners reveal a significant reduction in time (up to 92% faster) required to get feedback from tests on a built version of the SUT. Despite reducing the size of the test suite by 85%, 65% and 70%, we are still able to provide full coverage of, respectively, test requirements, dependencies and steps.

In addition, our findings reveal differences and similarities with respect to usage of similarity functions on different types of test artefacts, which confirm results by some previous studies [7, 9] and provide a contrast to the study by Coutinho et al. [5]. Nonetheless, the benefits of using SBTCS to increase the diversity of a test set is confirmed by our results, hence contributing empirical evidence on the performance of similarity-based test optimisation.

On the other hand, the effects of discarding test cases on the defect detection rates are not explored in our study, and are planned for future work. Currently, that information is not a big risk to our practitioners, since the CI servers execute the entire test suite overnight. Therefore, developers are able to identify any slipped bugs the next day, while obtaining the benefit of faster feedback provided by the SBTCS techniques that are instrumented on the CI servers.

Furthermore, we continue to collect data from industrial partners interested in levering their CI pipelines through similarity-based test optimisation. In addition to the smaller subsets, the similarity-based optimisation is able to point out redundancies in a test set, thus being a candidate to support other test activities such as test design and maintenance. In summary, our contributions are a stepping stone to test automation on companies adopting lean and agile methodologies, since SBTCS provide faster development feedback by removing waste (i.e., accumulated redundancies) from test repositories.

## ACKNOWLEDGMENTS

We thank the participants in our reported case study for their availability to clarify questions about the dataset, and the engaging and insightful discussions. This work was supported by the Chalmers Software Center[7] Project 30 on Aspects of Automated Testing.

[7] https://www.software-center.se/

## REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. 2005. Is Mutation an Appropriate Tool for Testing Experiments?. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05)*. ACM, New York, NY, USA, 402–411. https://doi.org/10.1145/1062455.1062530

[2] Jose Campos, Andrea Arcuri, Gordon Fraser, and Rui Abreu. 2014. Continuous Test Generation: Enhancing Continuous Integration with Automated Test Generation. In *IEEE/ACM Int. Conference on Automated Software Engineering (ASE)*. ACM, New York, NY, USA, 55–66.

[3] Emanuela Gadelha Cartaxo, Francisco Gomes de Oliveira Neto, and Patrícia DL Machado. 2007. Automated Test Case Selection Based on a Similarity Function. *GI Jahrestagung (2)* 7 (2007), 399–404.

[4] Emanuela G. Cartaxo, Patrícia D. L. Machado, and Francisco G. de Oliveira Neto. 2011. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability* 21, 2 (2011), 75–100. https://doi.org/10.1002/stvr.413

[5] Ana Emília Victor Barbosa Coutinho, Emanuela Gadelha Cartaxo, and Patrícia Duarte de Lima Machado. 2016. Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing. *Software Quality Journal* 24, 2 (01 Jun 2016), 407–445. https://doi.org/10.1007/s11219-014-9265-z

[6] Francisco Gomes de Oliveira Neto, Robert Feldt, Richard Torkar, and Patrícia D. L. Machado. 2013. Searching for Models to Evaluate Software Technology. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE '13)*. IEEE Press, Piscataway, NJ, USA, 12–15. http://dl.acm.org/citation.cfm?id=2662572.2662578

[7] Francisco Gomes de Oliveira Neto, Richard Torkar, and Patrícia D.L. Machado. 2016. Full modification coverage through automatic similarity-based test case selection. *Information and Software Technology* 80 (2016), 124 – 137. https://doi.org/10.1016/j.infsof.2016.08.008

[8] P.M. Duvall, S. Matyas, and A. Glover. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley. https://books.google.se/books?id=MA8QmAEACAAJ

[9] R. Feldt, S. Poulding, D. Clark, and S. Yoo. 2016. Test Set Diameter: Quantifying the Diversity of Sets of Test Cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. 223–233. https://doi.org/10.1109/ICST.2016.33

[10] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal. 2008. Searching for Cognitively Diverse Tests: Towards Universal Test Diversity Metrics. In *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*. 178–186. https://doi.org/10.1109/ICSTW.2008.36

[11] T. Gorschek, P. Garre, S. Larsson, and C. Wohlin. 2006. A Model for Technology Transfer in Practice. *IEEE Software* 23, 6 (Nov 2006), 88–95. https://doi.org/10.1109/MS.2006.147

[12] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. 2013. Achieving Scalable Model-based Testing Through Test Case Diversity. *ACM Trans. Softw. Eng. Methodol.* 22, 1, Article 6 (March 2013), 42 pages. https://doi.org/10.1145/2430536.2430540

[13] H. Hemmati, Z. Fang, and M. V. Mantyla. 2015. Prioritizing Manual Test Cases in Traditional and Rapid Release Environments. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. 1–10. https://doi.org/10.1109/ICST.2015.7102602

[14] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon. 2016. Comparing White-Box and Black-Box Test Prioritization. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 523–534. https://doi.org/10.1145/2884781.2884791

[15] Yves Ledru, Alexandre Petrenko, Sergiy Boroday, and Nadine Mandran. 2012. Prioritizing test cases with string distances. *Automated Software Engineering* 19, 1 (01 Mar 2012), 65–95. https://doi.org/10.1007/s10515-011-0093-0

[16] Andres Marzal and Enrique Vidal. 1993. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15, 9 (Sep 1993), 926–932. https://doi.org/10.1109/34.232078

[17] T. B. Noor and H. Hemmati. 2015. A similarity-based approach for test case prioritization using historical failure data. In *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*. 58–68. https://doi.org/10.1109/ISSRE.2015.7381799

[18] G. Rothermel, R. H. Untch, Chengyun Chu, and M. J. Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering* 27, 10 (Oct 2001), 929–948. https://doi.org/10.1109/32.962562

[19] Per Runeson and Martin Höst. 2008. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14, 2 (2008), 131–164. https://doi.org/10.1007/s10664-008-9102-8

[20] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[21] S. Yoo and M. Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability* 22, 2 (2012), 67–120. https://doi.org/10.1002/stvr.430