Supporting Continuous Integration by Code-Churn Based Test Selection

Eric Knauss*, Miroslaw Staron*, Wilhelm Meding†, Ola Söder‡, Agneta Nilsson*, Magnus Castell†
*University of Gothenburg
name.surname@gu.se
†Ericsson AB
name.surname@ericsson.com
‡Axis Communications
ola.soder@axis.com

***Abstract*—Continuous integration promises advantages in large-scale software development by enabling software development organizations to deliver new functions faster. However, implementing continuous integration in large software development organizations is challenging because of organizational, social and technical reasons. One of the technical challenges is the ability to rapidly prioritize the test cases which can be executed quickly and trigger the most failures as early as possible. In our research we propose and evaluate a method for selecting a suitable set of functional regression tests on system level. The method is based on analysis of correlations between test-case failures and source code changes and is evaluated by combining semi-structured interviews and workshops with practitioners at Ericsson and Axis Communications in Sweden. The results show that using measures of precision and recall, the test cases can be prioritized. The prioritization leads to finding an optimal test suite to execute before the integration.**

I. Introduction

Software development organizations need to meet the demands of rapidly changing requirements and to release new products and features more often and much faster. A typical initiative to meet these demands has been the transition from traditional waterfall software development environment towards agile practices, which embraces change and emphasize customer collaboration [1]. Agile practices have shown to be helpful towards this end [2] and companies work towards increasing the velocity in order to be able to release products and features more often and faster. One of a mechanisms of agile practices is the continuous integration used in the environment of large-scale software development. A well-known bottleneck when introducing continuous integration, concerns the testing activities and the ability to choose an effective test suite for each integration cycle [3]. Efficient testing arrangements are central for achieving continuous integration, and efforts to organize these activities means dealing with high levels of complexity [4], [5].

A challenge to arrange efficient testing is to know how to select what tests to execute on what code and when [6]. There is a tendency in software development organizations to be safe rather than sorry and execute more tests than necessarily needed because they are available, and it is possible. However, this approach has a negative trade off on the velocity of the entire software development and consequently may impede the release of products and features more often and faster. Execution of unnecessary tests takes resources and can provide false confidence in the quality of the product. Hence, it would be useful to know how to identify and select the most important tests at certain situations.

Our research goal is to develop and evaluate a method for selecting a suitable set of functional regression tests on system level to support continuous integration. The method developed in our study is based on the observation that in continuous integration and testing we can develop a statistical model of connections between changed code fragments (so called code churns) and the results of test cases. In our work we use the definition of code churn by Kim et al. [7] – *Code churn* is defined as a piece of code which has been changed during a single check-in in the code repository. The statistical model is a contingency table of test case execution results and code churns, which can be visualized using the notion of a heatmap [8], [9]. The statistical model is used to suggest which test cases should be executed during the integration cycles in order to get as short feedback loops as possible.

The evaluation of the method is done based on a case study with two companies – Ericsson AB and Axis Communications. Both companies develop embedded software using different flavours of Agile software development [10], [11], [12]. The different set-up of the software development at both companies allows us to also explore how robust our method to changes in the context (e.g. the pace of continuous integration or size of the source code base/test code base).

This paper is structured as follows: in the next section, we present the most related work. In Section III, we describe the method for constructing the statistical model and suggesting the selection of test cases. In Section IV, we describe the design of the case study including the description of the system-level testing at the companies. In Section V, we present the results of the evaluation. The reported research is concluded in Section VI along with outlooks to future research.

II. Related work

We have identified the related work in three areas – test selection and test prioritization, continuous integration and visualization of large quantities of data for decision support.

A. Test selection and prioritization

Test case selection has been studied from a number of perspectives. One of the perspectives is the increase of test coverage by automated means. Arts et al. [13] and Derrick et al. [14] used formal methods to derive the minimal failing test suite in an automated way based on formally defined sets of properties. The property-based testing approach, however, requires formalization of what should be tested (the property). In our approach we reuse the existing test suites and study their sensitivity to capture source code changes and assess their quality.

An alternative to the automated increase of test coverage is pre-selection of test cases based on program modifications and manual models for test-code relationships. Yoo and Harman [6] presented a survey of how test suites are prioritized and optimized. One of the studied problems was test case selection based on modifications of the program. According to their classification our approach is similar to text-based approach. An example of such a technique is Vokolos and Frankl's work [15]. They presented a technique for selecting test cases based on textual differences of programs reduced to symbolic representation. Although the results are very promising, they require one extra step translating the programs to symbolic code which can add additional burden on the designers.

Another approach to linking of test cases to code is partitioning of programs. An example is the work of Chen et al. [16] who used partitioning of programs to map test cases to tested parts of the source code and trace the modifications of the source code. Their approach is similar to ours but requires pre-processing in terms of partitioning of the programs into entities, which is not required for our approach.

Marculescu et al. [17] presented an approach for test suite selection based on search-based techniques. The search-based approach uses a set of predefined fitness functions to assess the suitability of test suites. Our approach is compatibles with that approach and can be used to deliver fitness functions based on source code changes.

Engström et al. [18] studied the state-of-practice in using automated test selection by conducting a customized systematic literature review. One of the important findings, which is in line with our goal, was that the automated test selection should be assisted by experienced testers the software should make a recommendation and the testers should make the decision. In our work we prepare the testers with support for simulations of what-if scenarios through test recommender.

B. Continuous integration

Nilsson et al. [4] developed a Continuous Integration Visualization Technique (CIViT) that provides an overview of end-to-end testing activities. The CIViT model serves as a solution to the lack of a holistic, end-to-end understanding of the testing activities and their periodicity in organizations. The model has proven particularly useful as a basis for discussion, which help to identify problems of current testing activities, regarding scope and periodicity, and to reason about suitable measures and to identify how to best improve their testing activities towards continuous integration. In our work, we see the need for combining this overview with support for decisions of how testing activities could be made more efficient through the use of simulations of what-if scenarios through test recommender.

Stolberg [19] described a number of experiences from introducing continuous integration into a software development organization. The experiences are aligned with many of the observed challenges in our industrial partners – need to run the complete regression suite, lack of prioritization methods that are automated. In this paper we intend to address this gap. Similar experiences were shared by Kim et al. [20] which indicates that the gap affects more than a limited number of organizations.

C. Metrics visualization

Visualization of source code changes using heatmaps was previously studied by Feldt et al. [9] in a context of daily follow up of project progress. In this paper we expand on these experiences and present a case where heatmaps can be used for visualizing dependencies between test cases and source code.

The presentation of the results from the test case prioritization requires an efficient distribution mechanism. In the studied case we investigate the use of cloud-based dissemination presented in our previous works [21]

III. CCTS: Code-Churn Based Test Selection Method

The CCTS method comprises of two parts – (i) historical analysis of code churns and test execution results, and (ii) finding optimal test suite using precision and recall metrics.

A. Historical analysis of code churns and test execution results

The historical analysis takes two inputs – the list of source code changes (e.g. results from diff in the source code repository per day) and the results of test case execution (e.g. a list of test cases executed and the results of the execution per day). The method creates a contingency table which shows how often a test case fails if there is a source code change in that particular day – for example as shown in table I.

TABLE I
Example contingency table

	Test case 1	**Test case 2**
Module 1	1	0
Module 2	1	7
Module 3	1	10
Module 4	0	0

This contingency table shows which test cases are the most sensitive to changed in the relevant source code modules. This contingency table constitutes the statistical model which is used in the next step – finding the optimal test suite.

B. *Finding optimal test suite*

The automatic recommender of optimal test suite takes as input i) the contingency table and ii) a list of recently changed modules. The contingency table is the statistical model which forms the baseline of which test cases should be recommended for which source code modules.

In order to find the optimal test suite we use the information retrieval measures – recall, precision and the f-measure. These measures are based on four categories of errors:

1) True positives: The set of tests that are both recommended by the recommender system and failed according to the ground truth.
2) False positives: The set of tests that are recommended but did not fail according to the ground truth.
3) True negatives: The set of tests that are not recommended and that did not fail according to ground truth.
4) False negatives: The set of tests that are not recommended but should have been, as they failed according to the ground truth.

Recall indicates the percentage of the tests that actually failed which were recommended. It is defined as:

$$\text{recall} = \frac{|\text{True Positive}|}{|\text{True Positive}|+|\text{False Negative}|}$$

A high recall (close to 1) is important, because otherwise tests that would have failed would have been omitted. Precision indicates the percentage of the tests that were recommended which actually failed. It is defined as:

$$\text{precision} = \frac{|\text{True Positive}|}{|\text{True Positive}|+|\text{False Positive}|}$$

An increased precision (close to 1) is important, because it corresponds to a relative speedup of testing by eliminating the need to run tests that do not provide new knowledge about the quality of the system.

Recall and precision relate to each other. The easiest way to get a high recall is to simply recommend all tests. In that case, all tests that could fail are selected but the precision is minimal and no execution time is saved. It is an open question of whether recall or precision is more important for test selection, which we evaluate in section V.

Under the assumption that both precision and recall are equally important, it makes sense to compute the f-measure. The f-measure is the geometric mean of recall and precision, defined as:

$$\text{f-measure} = \frac{2*\text{recall}*\text{precision}}{\text{recall}+\text{precision}}$$

The f-measure allows comparing the overall performance – and this allows to choose the optimal set of test cases to run given the changed source code modules and the contingency table from historical analysis.

For the continuous integration this measure is crucial as it enables automated selection of test cases. Together with the ability of automatically adjust the contingency table (along with each integration/test cycle), this approach reduces the effort for test planning and increases the chances of finding defects already during the integration.

IV. Case study design

We evaluate the CCTS method at two companies – Ericsson and Axis Communications. The goal of the evaluation is to address the following research question – *How can the CCTS method be applied during continuous integration cycles in practice?* We perform the evaluation by collecting the historical data for the integration cycles and by simulating which test cases should be selected in order to optimize the effectiveness of the test suite. We can classify this case study as ecaluative multiple case study [22], [23], consisting of two cases.

The data is collected using the following methods:

1) Document analysis – collecting the historical data about code churns and test results.
2) Semi-structured interviews – collecting the feedback on the visualization and the correctness of the contingency table (i.e. the statistical model).
3) Focus group discussions iteratively with key representatives from development and management – to inform our thematic coding and triangulate our findings from an organizational perspective, i.e. how code churn based test selection supports continuous integration.
4) Group discussions with testing experts – to triangulate our findings and verify the findings during the research process (and decide upon further directions).

A. *Collaborating companies*

Research in this paper is carried out at two companies, which we shortly introduce in the following sections.

1) System level testing at Ericsson: Ericsson AB (Ericsson) develops large software products for the mobile telecommunication network. The size of the organization during the study is several hundred engineers and the size of the projects is up to a few hundreds . Projects are increasingly often executed according to the principles of Agile software development and Lean production system, referred to as Streamline development (SD) within Ericsson [1]. In this environment, various teams are responsible for larger parts of the process compared to traditional processes: design teams (cross-functional teams responsible for complete analysis, design, implementation, and testing of particular features of the product), network verification and integration testing, etc. The organization uses a number of measurement systems for controlling the software development project (per project) described above, a number of measurement systems to control the quality of products in field (per product) and a measurement system for monitoring the status of the organization at the top level. All measurement systems are developed using the in-house methods described in [24], with the particular emphasis on models for design and deployment of measurement systems presented in [25]. The needs of the organization evolved from metrics calculations and presentations (ca. 8 years before the writing of this paper), to using predictions, simulations, early warning systems and handling of vast quantities of data to steer organizations at different levels, and providing information from project and line. These needs are addressed by the action research projects conducted in the organization, since the 2006. In this paper we discuss functional verification on system level at Ericsson.

2) System level testing at Axis Communications: Axis Communications (Axis) is the market leader in network video and a driving force behind the shift from analogue to digital video surveillance. Axis offers network video solutions for professional installations featuring products and solutions that are based on innovative and open technical platforms. Thus, Axis is developing a range of software products software embedded in network cameras, video encoders and desktop video management software. Software development processes at the company are based on the agile principles with frequent deliveries to the main branch and empowered software development teams. The testing at Axis referred to in this paper consists of system level tests using complete products such as they are delivered to the customer. The purpose of these tests is to reveal not yet known problematic interactions that cannot be found using lower level testing. The product which is studied in this company was the network camera product with a number of releases on the market. The team was multidisciplinary and comprised designers, architects and testers.

V. Results

In this section we show the statistical models from both collaborating companies visualized as heatmaps and we summarize the results from the interviews and group discussions. Finally we summarize the results in two scenarios where the CCTS have the most of the potential.

A. Identifying efficient tests: statistical models

Figure 1 shows the contingency table (heatmap) from Ericsson. The rows are on the component level (i.e. groups of source code modules) and the columns are test suites (i.e. groups of test cases). The data used for the recommendations is on the module and test cases level, but for the purpose of efficient visualization we choose to compress the changes.

In Fig. 1, on the left hand side there are is one test suite that is sensitive to changes in a large number of modules. Then, there are a couple of tests that are not responding to any changes in modules, represented by a broad white column. On the top, there are modules that do not cause any tests to fail, represented by a broad white row. According to our interviews, the way this visualization can be interpreted depends on the level of abstraction of the tests and on the purpose of the test suites – *"The goal of our [platform tests] is to identify hidden dependencies. Fault omission and lower level faults should be found on lower levels"*.

On system level tests the more intensive colored areas in the heatmap are valuable for system and test architects when prioritizing refactoring efforts or creating a test suite with short execution time (e.g. for daily execution). There is also one test suite (on the right-hand side) which should be included in all test executions as it seems to be the most sensitive to the changed in the source code.

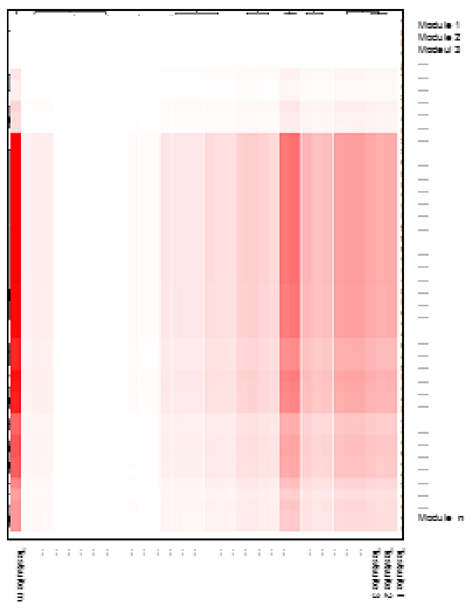

Fig. 1. Heatmap from Ericsson

Our interviewees gave us one candidate explanation for the white rows and columns in Figure 1. Accordingly, the visualization shows all tests but not all modules. Specifically, modules from one of the two major components are missing and it is likely that the tests associated with the white column are more sensitive to changes in this component.

At Axis the relationship between the source code changes and the test failures is shown in Fig. 2. This heatmap is presented on the module and test case level and shows the size of the contingency table – hundreds of rows and columns.

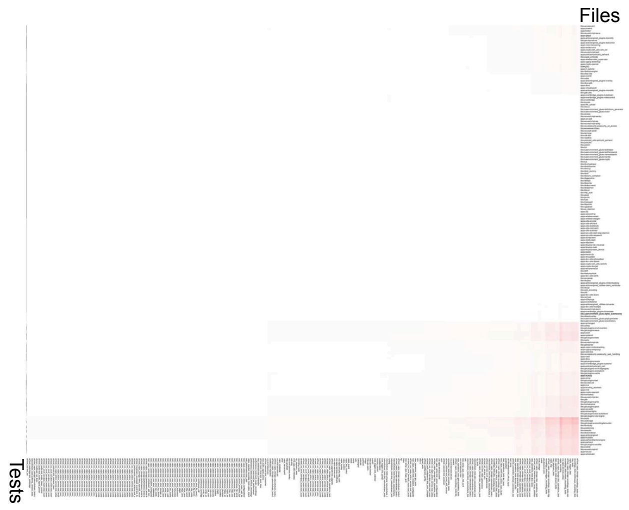

Fig. 2. Heatmap from Axis

The pattern in Fig. 2 is different from the pattern in the previous figure. For example there is no single test case which fails based on the majority of code changes. We can also observe that the majority of test cases do not fail.

B. *Simulating test selection using CCTS*

The existing heatmap concept allows a simple recommendation algorithm, which, for a given set of changed modules, selects all tests for which the heatmap reports a co-occurring test failure.

To understand the performance of this algorithm, we used the heatmap constructed based on data from Axis. Analysis of the heatmap reveals that 1051 tests exist of which test failures were reported for 958. Thus, a first recommendation of test selection can be to suggest removing almost 100 tests from the test suite without considering the input of changed modules.

Axis also provided us with a single observation of a list of changed modules and tests that failed, representing results of one specific run. Note that this being only one data point does not allow for a conclusive assessment. The specific observation contains a list of 142 changed modules and 326 failed tests. For this observation, the simple heatmap algorithm recommends 948 tests for which failures are known, leading to a recall of 1, a precision of 0.26, and an f-measure of 0.41, which can be considered as a weak recommendation, as the number of tests is not sufficiently reduced.

We therefore experimented with two strategies to further reduce the number of recommended tests based on the following algorithm, which requires a heatmap h and a list of recently changed modules c to recommend tests:

1) Use the heatmap h to look up n_c: how often each test failed together with a change in a module in c.
2) Order the tests in the heatmap according to this value, highest value first into the prioritized list of tests.
3) Return the top tests from the list:
    a) *Strategy 1:* Return the first n% tests from the list.
    b) *Strategy 2:* Return the first item from the list and every other test with $n_c \geq m\%$ of the first items n_c.

Results from applying these strategies on the Axis data are presented in Table II. The first row in Table II shows that if all 1051 tests are executed, there is no speedup. Columns n and m refer to Strategy 1 and 2 in the above algorithm and do not apply for this row. Because we run all tests, we also run all tests that could fail and recall is 1. However, we also run the maximum number of tests that did not fail, which results in this data set into a precision of 0.24 and a f-measure of 0.39.

For the second row, we give the same data for all 958 tests that have previously failed according to our heatmap. As we run less tests that would not have failed, precision and f-measure are slightly better and, assuming similar execution time for each test, the recommended testsuite runs 1.1 times faster than all tests.

The third row presents the data from only running step one of the heatmap algorithm, again with small improvements. The two last rows finally show the quite similar results that we can get with both selection strategies for optimal choice of n (or m respectively). These rows are quite promising, as they indicate that we can run the recommended test suits more than 3.5 times more often than all tests in the same time.

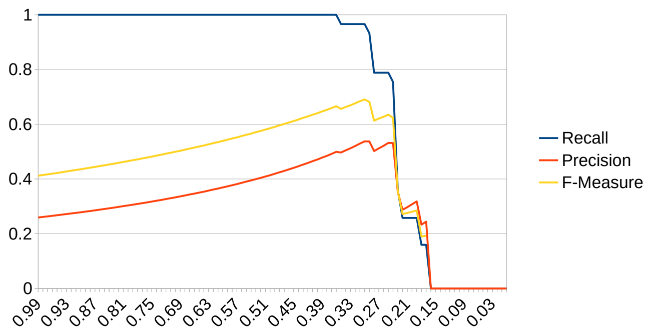

Fig. 3. Results for reporting only the best n% of recommendations.

Figure 3 and 4 show how recall, precision, and f-measure change if the values for n and m (according to both selection strategies) are tweaked.

The leftmost data point in Figure 3 shows recall, precision, and f-measure for returning all tests that have a recorded failure in the heatmap. While the list of recommended tests gets smaller, the precision slowly goes up: Less tests are recommended that do not fail. At some point around n = 35%, the recall starts to drop from 0.1, when tests that would have failed according to the Axis data set would no longer be recommended. Without further information on whether to prioritize recall or precision, the highest value of f-measure (yellow line in Fig. 3) denotes the optimal length of recommendations of $\approx 30\%$ of the tests in this case.

Figure 4 shows similar values for the second strategy. In the investigated case, all recommended tests should have at least 60% of the recorded test failures that the first test in the prioritized list of tests has for the given set of changed modules $(0.6 \leq m \leq 0.72)$.

The data to conduct the simulations for Ericsson was not available at the time of writing of this paper.

TABLE II
RESULTS FROM ASSESSMENT OF DIFFERENT CONFIGURATIONS OF TEST CASES

Case	**No. of Tests**	**Speed-up**	**n**	**m**	**f-m.**	**prec.**	**rec.**
All tests	1051	0	–	–	0.39	0.24	1.00
Previously failed tests	958	1.10	–	–	0.41	0.26	1.00
Heatmap algorithm	948	1.11	1.00	0.00	0.41	0.26	1.00
Optimal case 1 (Fig. 3)	287	3.68	0.30	–	0.69	0.54	0.97
Optimal case 2 (Fig. 4)	283	3.71	–	0.65	0.69	0.54	0.97

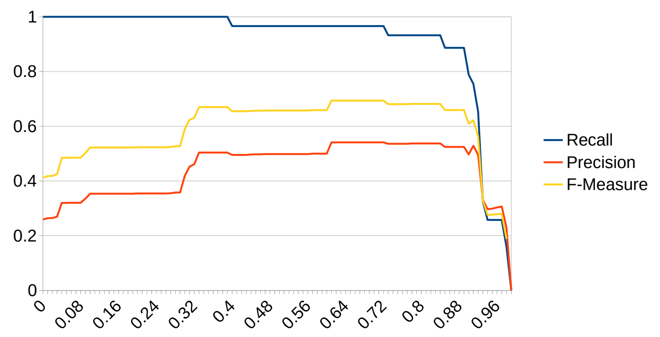

Fig. 4. Results for reporting only the recommendations within m% of the best recommendation's quality.

C. *Scenarios for Applying CCTS*

With respect to research goal of refining scenarios on how risk based test selection can offer value in daily work, we found that heatmaps enable refactoring and restructuring of the test landscape. In our interviews at Ericsson we identified two scenarios, which were confirmed by interviewees at Axis: i) Re-distributing effort over test scope levels and ii) restructuring test suites.

Scenario 1: Re-distributing effort over test scope levels. It is a goal to find problems as early as possible, i.e. to find all the problems inside of a unit with unit tests. The heatmap supports test managers in this, because they can start an analysis from the dark red areas. The goal then would be to reduce the darkness in the heatmap, i.e. having less test failures on this abstraction level.

Scenario 2: Re-structuring test suites. Interviewees at both companies indicated that tests should be selected and scheduled based on different scopes, e.g. for hourly, daily, and weekly execution. Thus, it seems like a good strategy to select tests for these test suites dynamically based on the changes to software artifacts (at least in the case of the hourly tests).

Both scenarios are in line with our research goal to develop a method which will help to select a suitable set of functional regression tests on system level that balances the need to find integration problems with the need to execute the tests quickly enough to support the fast pace of continuous integration. By dynamically constructing test suites for XFTs, obsolete tests are removed from their test suites. By taking into account all available tests and their history, tests that were previously missing in a XFTs test suite can be included if they promise enough value based on historical data and current source code changes.

VI. Conclusions

Continuous integration requires new ways of selecting test cases and requires more automation in that area, but this automation provides also new possibilities. One of such possibilities is automated collection of statistics of test case executions and collection of code churn statistics. These statistics allow for developing statistical models over which test cases are the most sensitive to changed in the source code – both in the total code base and in selected areas.

In this paper we explored the possibilities for using the new data as input to an automated test selection algorithm. The algorithm uses historical data and measures from the information retrieval area (precision, recall and f-measure) to find the optimal test suite to execute given a set of code churns.

The initial evaluation of the method at Ericsson and Axis Communications showed that the approach is feasible and can be used in practice in two scenarios. The first scenario is the restructuring of the test suites and the other redistribution of the test scope levels.

In our future work we intend to develop a tool for orchestrating test cases to further refine our method and provide a smarter test infrastructure.

Acknowledgment

This research has been carried out in the Software Centre, Chalmers, Göteborgs Universitet and Ericsson AB, AB Volvo, Saab AB, Volvo Car Group, Sony Mobile, Grundfos, Axis Communications, Jeppesen, Malmö University

This work has been partially supported by the Swedish Strategic Research Foundation under the grant number SM13-0007.

References

[1] P. Tomaszewski, P. Berander, and L.-O. Damm, "From traditional to streamline development - opportunities and challenges," *Software Process Improvement and Practice*, vol. 2007, no. 1, pp. 1–20, 2007.

[2] M. Pikkarainen, O. Salo, R. Kuusela, and P. Abrahamsson, "Strengths and barriers behind the successful agile deploymentinsights from the three software intensive companies in finland," *Empirical software engineering*, vol. 17, no. 6, pp. 675–702, 2012.

[3] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[4] A. Nilsson, J. Bosch, and C. Berger, "Visualizing testing activities to support continuous integration: A multiple case study," in *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2014, pp. 171–186.

[5] N. Mellegård and M. Staron, "Characterizing model usage in embedded software engineering: a case study," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. ACM, 2010, pp. 245–252.

[6] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[7] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of refactoring challenges and benefits at microsoft," 2014.

[8] M. Staron, J. Hansson, R. Feldt, A. Henriksson, W. Meding, S. Nilsson, and C. Hoglund, "Measuring and visualizing code stability–a case study at three companies," in *Software Measurement and the 2013 Eighth International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2013 Joint Conference of the 23rd International Workshop on*. IEEE, 2013, pp. 191–200.

[9] R. Feldt, M. Staron, E. Hult, and T. Liljegren, "Supporting software decision meetings: Heatmaps for visualising test and code measurements," in *Software Engineering and Advanced Applications (SEAA), 2013 39th EUROMICRO Conference on*. IEEE, 2013, pp. 62–69.

[10] M. Staron, W. Meding, and K. Palm, "Release readiness indicator for mature agile and lean software development projects," in *Agile Processes in Software Engineering and Extreme Programming*. Springer, 2012, pp. 93–107.

[11] M. Staron, W. Meding, J. Hansson, C. Höglund, K. Niesel, and V. Bergmann, "Dashboards for continuous monitoring of quality for software product under development," *System Qualities and Software Architecture (SQSA)*, 2013.

[12] M. Staron and W. Meding, "Monitoring bottlenecks in agile and lean software development projects–a method and its industrial use," *Product-Focused Software Process Improvement*, pp. 3–16, 2011.

[13] T. Arts, J. Hughes, J. Johansson, and U. Wiger, "Testing telecoms software with quviq quickcheck," in *Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*. ACM, 2006, pp. 2–10.

[14] N. Walkinshaw, K. Bogdanov, J. Derrick, and J. Paris, "Increasing functional coverage by inductive testing: a case study," in *Testing Software and Systems*. Springer, 2010, pp. 126–141.

[15] F. I. Vokolos and P. G. Frankl, "Pythia: a regression test selection tool based on textual differencing," in *Reliability, quality and safety of software-intensive systems*. Springer, 1997, pp. 3–21.

[16] Y.-F. Chen, D. S. Rosenblum, and K.-P. Vo, "Testtube: A system for selective regression testing," in *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press, 1994, pp. 211–220.

[17] B. Marculescu, R. Feldt, and R. Torkar, "Practitioner-oriented visualization in an interactive search-based software test creation tool," in *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*. IEEE, 2013, pp. 87–92.

[18] E. Engström, R. Feldt, and R. Torkar, "Indirect effects in evidential assessment: a case study on regression test technology adoption," in *Proceedings of the 2nd international workshop on Evidential assessment of software technologies*. ACM, 2012, pp. 15–20.

[19] S. Stolberg, "Enabling agile testing through continuous integration," in *Agile Conference, 2009. AGILE'09.* IEEE, 2009, pp. 369–374.

[20] S. Kim, S. Park, J. Yun, and Y. Lee, "Automated continuous integration of component-based software: An industrial experience," in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2008, pp. 423–426.

[21] M. Staron and W. Meding, "Metricscloud: Scaling-up metrics dissemination in large organizations," *Advances in Software Engineering*, vol. 2014, 2014.

[22] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical software engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[23] R. K. Yin, "Discovering the future of the case study method in evaluation research," *Evaluation Practice*, vol. 15, no. 3, pp. 283–290, 1994.

[24] M. Staron, W. Meding, and C. Nilsson, "A framework for developing measurement systems and its industrial evaluation," *Information and Software Technology*, vol. 51, no. 4, pp. 721–737, 2008.

[25] M. Staron and W. Meding, "Using models to develop measurement systems: A method and its industrial use," vol. 5891, pp. 212–226, 2009.