Practical Selective Regression Testing with Effective Redundancy in Interleaved Tests

Dusica Marijan
Simula, Norway
dusica@simula.no

Marius Liaaen
Cisco
Norway

ABSTRACT

As software systems evolve and change over time, test suites used for checking the correctness of software typically grow larger. Together with size, test suites tend to grow in redundancy. This is especially problematic for complex highly-configurable software domains, as growing the size of test suites significantly impacts the cost of regression testing.

In this paper we present a practical approach for reducing ineffective redundancy of regression suites in continuous integration testing (strict constraints on time-efficiency) for highly-configurable software. The main idea of our approach consists in combining coverage based redundancy metrics (test overlap) with historical fault-detection effectiveness of integration tests, to identify ineffective redundancy that is eliminated from a regression test suite. We first apply and evaluate the approach in testing of industrial video conferencing software. We further evaluate the approach using a large set of artificial subjects, in terms of fault-detection effectiveness and timeliness of regression test feedback. We compare the results with an advanced retest-all approach and random test selection. The results show that regression test selection based on coverage and history analysis can: 1) reduce regression test feedback compared to industry practice (up to 39%), 2) reduce test feedback compared to the advanced retest-all approach (up to 45%) without significantly compromising fault-detection effectiveness (less than 0.5% on average), and 3) improve fault detection effectiveness compared to random selection (72% on average).

CCS CONCEPTS

• **Software and its engineering** → *Empirical software validation*;

KEYWORDS

Test redundancy, selective regression testing, highly-configurable software, highly-interleaving tests, test optimization, continuous regression testing

ACM Reference Format:
Dusica Marijan and Marius Liaaen. 2018. Practical Selective Regression Testing with Effective Redundancy in Interleaved Tests. In *ICSE-SEIP '18: 40th International Conference on Software Engineering: Software Engineering in Practice Track, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3183519.3183532

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICSE-SEIP '18, May 27-June 3, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5659-6/18/05…$15.00
https://doi.org/10.1145/3183519.3183532

1 INTRODUCTION

As software systems evolve and change over time, test suites used to check the correctness of software change simultaneously, and typically grow in size. When new functionality corresponding to the specific set of requirements has been implemented, new tests are added to the suite covering single requirements or combinations of their interactions. To deal with large test suites, combinatorial interaction testing is often applied to highly-configurable software (HCS) for reducing costs and improving the efficiency of testing. However, depending on the target interaction strength of test configurations, repeated coverage of test requirements (and their combinations) across configurations can lead to *test redundancy*. A test suite containing redundant tests increases the cost of testing as well as test maintenance effort, which implies that for effective testing, test redundancy needs to be analyzed and removed, or maximally reduced.

In our collaboration with Cisco, we have identified the case that makes dealing with test redundancy particularly difficult. That is the case of testing HCS using *interleaved tests*. These are the tests that test high interaction degree of individual features forming chain configurations, where sub-configurations occur frequently across individual test cases. This makes it difficult to effectively reduce redundancy in the initial test suite for HCS using traditional coverage-based techniques.

Furthermore, regression tests typically executed after code changes, either due to defect fixing or other improvements, ensure that the changes do not introduce new faults (regressions). Observed from our industry experience, a common approach to regression testing in practice is retest-all, which consists in rerunning all tests, sometimes including those not specifically covering the parts of software affected by changes. However, to make regression testing more efficient only the relevant minimal set of tests that adequately cover the changes should be selected and run. While retest-all approach is easily automated, selective regression testing is mainly a manual process done by developers/testers, thus often unsystematic and liable to subjectivity. Because test suites grow in size over time, introducing redundancy, manually selecting an adequate non-redundant set of regression tests soon becomes inefficient and impractical. This is especially evident in continuous integration (CI) development, where regression testing runs as part of a timeboxed development iteration restricted to a specific duration. As these iterations are short, less time is available for testing, making it difficult to select regression tests using a manual approach. Because

regression test feedback needs to be provided rapidly, efficient regression test selection needs to be automated and further optimized for minimal test redundancy.

Existing work (detailed in a later section) has been predominantly focused on test minimization, without specific focus on redundancy analysis for highly-interleaved tests in HCS. Unlike these approaches, we focus on reducing redundancy of highly-interleaved regression tests for HCS in CI. The approach consists in, first, identifying test redundancy in a regression suite by analyzing the overlap of configuration options (features) across a test suite, relative to the target feature interaction strength. Based on this, we classify tests as unique, totally redundant, or partially redundant. Then we use historical test execution data to determine which combinations have proved high fault-revealing capabilities in previous test runs. Based on this information, we classify partially redundant tests into effective and ineffective tests. Finally, we create a regression test suite consisting of only unique and efficient partially redundant tests.

We apply the approach on an industrial video conferencing software (VCS) developed in CI, and use historical test execution data in evaluation. In further validation, we use 110 artificial HCS test suites, developed motivated by the VCS, totaling to 111 experiment subjects. We compare the approach with current industry practice, with an advanced retest-all approach, and with random test selection, in terms of fault-detection effectiveness and timeliness of regression test feedback (directly correlated with test execution time). The results show that the proposed approach can improve cost-effectiveness of regression testing of HCS compared to industry practice, as well as often used practical strategies such as random and retest-all.

The paper makes the following contributions:

1) This is, to our knowledge, the first approach to combine test coverage metrics and execution history for reducing redundancy of regression testing for HCS in CI.
2) We present a pragmatic and effective solution to a real industry problem of testing large-scale video conferencing software with interleaved tests at Cisco Systems.
3) We describe the context of the problem, focusing on challenges of optimizing interleaved test suites for testing highly configurable software, and discuss why common techniques such as manual test selection and retest-all do not produce satisfactory results.
4) We provide experimental results showing the effectiveness of the approach in testing of industrial software, and well as better performance compared to practical strategies such as random and retest-all approaches.

In the remainder of the paper we review preliminaries in Section 2. We present the running example used in the paper in Section 3, followed by the problem formulation in Section 4. We describe our solution to test redundancy detection and reduction in Section 5. We present the experimental study performed to evaluate the approach in Section 6, and discuss the results in Section 7. We discuss threats to validity in Section 8, and review related work in Section 9. Finally, we give the conclusion and highlight further research in Section 10.

2 PRELIMINARIES

This section describes the concept of test redundancy in testing HCS, and revisits the problem of regression test selection in the context of HCS testing.

Highly configurable software consists of a common code base and a set of configuration options (features) used for customizing main software functionality. For example, features in a configurable video communication software include video resolution or networking protocol type. We consider that a test case for testing of HCS is an integration test aimed at checking the correctness of interaction between involved features. Given a HCS with a set of features $FS = \{f_1, f_2, ..., f_n\}$, and a test suite $TS = \{t_1, t_2, ..., t_m\}$ for testing of HCS, we can define an association function $Cov : TS \rightarrow FS$, associating a set of covering features to each test case. $Cov(t_i) = \{f_1, f_2, ..., f_k\}, k \leq n$, represents a set of features tested by t_i. We assume that the relation between $t_i \in TS$ and $f_i \in FS$ is "many-to-many", which means that a feature can be covered by multiple test cases, and a test case can cover multiple features. We further assume that $\forall t_i \in TS, Cov\{t_i\} \neq \emptyset$, i.e. each test covers at least one feature.

2.1 Test Case Redundancy

Test redundancy is often defined with respect to coverage metrics, such that for a given coverage criterion (for example pairwise feature coverage) if two tests t_i and t_j execute the same pair of features, one of the tests in a suite $TS = \{t_i, t_j\}$ contributes to test suite redundancy.

There are numerous causes of test redundancy, for example, test reuse in manual test specification, when existing tests are modified for testing new similar functionality, unintentionally leaving parts of already tested functionality. Other causes include incomplete requirements specification, redundancy of requirements, legacy, static test suites, parallel testing, or distributed testing [6]. In this work we focus on redundancy of integration tests, and in particular redundancy that is introduced during test case design, where test cases are developed to test a varying number of feature interactions. Redundant combinations of feature interactions in a test suite cause the same functionality being executed in multiple instances.

Formally, if we use $FSet$ to denote a feature set covered by a test suite TS, and $Cov(t)$ to denote the set of features covered by the test case $t \in TS$, then:

(1) A test case t is considered *redundant* in TS if $Cov(TS \setminus \{t\}) = FSet$,
(2) A test case t_i is considered *redundant* with respect to t_j if $Cov(t_i) \subseteq Cov(t_j)$.

2.2 Regression Test Selection

Regression test selection is a technique used to improve the cost-efficiency of software testing after a change has been made, by selecting an effective set of test cases that will check whether the change (e.g. defect fix) introduced new faults. For a given set of changes $Ch = \{Ch_1, Ch_2, ..., Ch_n\}$ made to a software system S, and an existing test suite $TS = \{T_1, T_2, ..., T_n\}$ used to test S, regression test selection determines $TS' \subseteq TS$, consisting of tests that

are relevant for Ch, to be used for testing of S'. Selection is performed according to the selection function f_{sel} that uses information about Ch_i and T_i to find those T_i that are relevant for Ch_i. Regression test selection may additionally use a set of objectives $Obj = \{Obj_1, Obj_2, ..., Obj_n\}$ to optimize the execution of TS'. Some typical objectives include high fault detection, low execution time/cost, etc. Regression test selection may also use historical test execution data to find an optimal TS' based on past test performance. Although various approaches have been proposed for regression test selection, the challenge remains to efficiently identify and reduce ineffective test redundancy for highly-interleaved tests.

2.3 Testing of Highly-Configurable Software

Configurable software typically requires sophisticated testing approaches due to the large number of configuration options found in realistic HCS, which normally causes redundancy in tests. The problem becomes especially evident in continuous regression testing, which runs frequently and is highly time-constrained. Test feedback on pass/fail regression tests needs to be provided rapidly, which necessitates that regression test suites are optimized for less redundancy. A common approach for testing of HCS has been combinatorial interaction testing (CIT), where tests are developed to cover combinations of configuration options depending on a target coverage criteria [3]. However, existing approaches predominantly consider a fixed-strength interaction coverage (for example pairwise), while we noticed in practice that realistic systems often require testing the combinations of configuration options with various degrees of interaction.

3 RUNNING EXAMPLE

We use running example derived from the industrial practice of testing highly-configurable video conferencing systems at Cisco. This example motivated our research, and the proposed solution is aimed at improving the existing practice in this domain. In the example, we introduce notations and assumptions used throughout the paper.

Consider a VCS *CBC* that consists of a set of functionality modules referred to as $FS = \{f_1, f_2, ..., f_n\}$. FS are used to build a set of products $CBCProd = \{P_1, ..., P_z\}$, which are software solutions representing various configurations of desktop or boardroom conferencing. There is a test suite $TS = \{T_1, T_2, ..., T_m\}$ used for testing of *CBC*. $Cov(T_i) = \{f_1, f_2, ..., f_k\}$, $k \leq n$ denotes a set of features tested by T_i. When TS is developed, association tags are established between pairs T_i / f_i, denoting which tests are relevant for which features. Each test $T_i \in TS$ is associated with a set of historical records $Fde(T_i) = \{Fde\_T_{i1}, Fde\_T_{i2}, ..., Fde\_T_{ij}\}$ that correspond to the execution statuses of j past test case executions (fail/pass status). $Fde\_T_{ij} = \{status_{i,j}, C_{fail}(Fde\_T_{ij})\}$ is associated with one or more configurations responsible for failure $C_{fail}(Fde\_T_{ij}) = \{C_1, C_2, ..., C_m\}$, where $C_i = \{f_1, f_2, ..., f_k\}$, and $status \in [0, 1]$.

CBC is developed and evolves incrementally and iteratively, following a CI practice, where much of the functionality is shared between different products. TS is developed to test all *CBCProd*, and because of shared functionality between products, the same parts of *CBC* are executed multiple times. These conditions are illustrated in Figure 1. Five products in the figure consist of different functionality modules $FS = \{A, B, C, D, E, ..., N\}$. In the context of *CBC*, features include an audio protocol or video resolution of a conferencing call, or network type, and they are reused across *CBC*. Tests in TS are highly-interleaved and specified with a varying degree of feature coverage, such as single-feature coverage:
$Cov(T5) = \{I\}$, $Cov(T6) = \{N\}$, $Cov(T7) = \{N\}$
or multiple-feature coverage:
$Cov(T1) = \{B, D\}$, $Cov(T2) = \{C, D\}$, $Cov(T3) = \{A, K\}$, $Cov(T4) = \{A, K, L, M\}$. TS evolves and grows, accumulating tests with different coverage criteria, which over time increases the risk of redundant testing. When using a single-feature coverage criterion, tests $T1$ and $T2$ overlap, as $Cov(T1) = \{B, D\}$ and $Cov(T2) = \{C, D\}$, contributing to *partial redundancy* in TS. We say that $T1$ is partially redundant with respect to $T2$ and vice versa. Assuming that $TS = \{T1, T2\}$, eliminating either $T1$ or $T2$ would leave a feature in *CBC* (B or C) untested. However, in a test suite containing multiple instances of partial redundancy for the same set of features, partially redundant tests can be eliminated. Contrarily, $\{T3, T4\}$ contributes to *total redundancy* in TS, as $Cov(T3) = \{A, K\}$ is a proper subset of $Cov(T4) = \{A, K, L, M\}$. We say that $T3$ is totally redundant with respect to $T4$.

Whenever *CBC* is modified, introducing changes $Ch = \{Ch_1, Ch_2, ..., Ch_n\}$, affected products in *CBCProd* are regression tested to ensure that new modifications do not negatively affect existing functionality. Regression tests are selected mainly manually from TS based on change impact analysis and intuitive assessment of test relevance for a given Ch_i. To be able to benefit from CI, test feedback needs to be produced as quickly as possible. One way to improve the agility of test cycle is to automate the process of regression test selection (time savings compared to manual selection). Another way is to reduce the size of regression test suite by automatically identifying and removing redundancy (time savings in test execution). In this paper, we focus on the latter, enabling more cost-effective regression testing of HCS in CI, by identifying and eliminating redundancy caused by overlapped feature combinations across tests.

4 PROBLEM FORMULATION

For the purpose of illustration, in this example we use single feature coverage as a target test coverage. A test $T_i \in TS$ can be considered:

Definition 1.
T_i is totally redundant of TS, if $\exists T_j \in TS, i \neq j,\ Cov(T_i) \subseteq Cov(T_j)$.

Definition 2.
T_i is partially redundant of TS, if $\exists T_j \in TS, i \neq j,\ Cov(T_i) \neq Cov(T_j)$ and $Cov(T_i) \cap Cov(T_j) \neq \emptyset$.

Definition 3. T_i is unique, if $\neg\exists T_j \in TS, i \neq j,\ Cov(T_i) = Cov(T_j)$.

From the example given in Figure 1, $T3$ is totally redundant of $T4$ because all feature combinations covered by $T3$ are also covered by $T4$. $T5$ is unique, because it uniquely covers feature I. $T1$ and $T2$ are mutually partially redundant, with respect to single-feature coverage, as they both cover feature C, but also other different features.

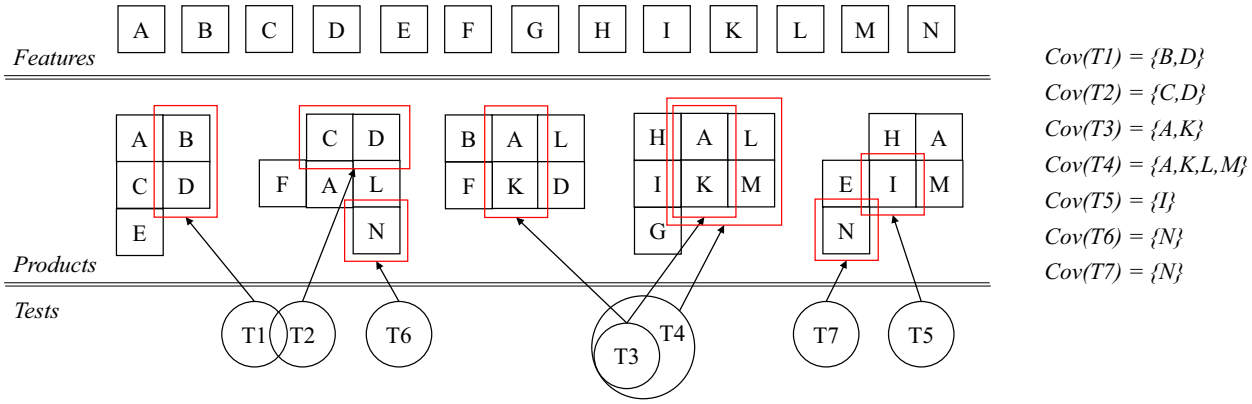

Figure 1: Test redundancy in HCS: Test cases $T1..6$ cover a set of features $A..N$ in different combinations across different products, causing partial or total test overlap.

As observed previously [19], using coverage-based metrics solely does not exhibit good performance in test redundancy detection. However, supplementing coverage metrics with the information about fault detection effectiveness of tests could provide more reliable test redundancy detection. Furthermore, assuming a fixed feature-combination coverage as a target test coverage criterion is allegedly an oversimplification for a vast majority of realistic and complex HCS systems. Features are typically integrated within different products in a varying degree of interaction. Thus, effective regression testing needs to be able to detect the cases when the same functionality (features) is executed as part of different feature combinations across different tests and products.

Based on the presented concepts, we formulate the problem of test redundancy reduction for highly-interleaved tests as follows:

For a given software change Ch, existing test suite TS, and fault detection records of each test based on execution history Fde_T_i, $T_i \in TS$, select $TS' \subseteq TS$ as a regression test suite with the following properties:

(i) *Property 1*: $\forall T \in TS'$ T includes features affected by Ch.

(ii) *Property 2*: $TS' = TS \setminus \{totally\ redundant\ tests\}$, excludes any totally redundant test in TS.

(iii) *Property 3*: $TS' \cap TS = \{unique\ tests\}$, includes all unique tests in TS.

(iv) *Property 4*: $TS' = TS \setminus \{ineffective\ tests\}$, where *ineffective* tests are tests in TS that do not contribute to increased fault detection effectiveness of TS'.

The following section provides an explanation of *ineffective* test case, i.e. when a partially redundant test case is considered *effective*, and when it is considered *ineffective*.

5 REDUNDANCY DETECTION AND REDUCTION

The major challenge in solving the above formulated problem lies in identifying *ineffective* test cases.

Given the Def 2. of partially redundant test case, for $\{T_i, T_j \in TS \mid i \neq j, Cov(T_i) \neq Cov(T_j), Cov(T_i) \cap Cov(T_j) \neq \emptyset\}$, we want to determine if exists $\{T_k \in TS \mid k \neq i, k \neq j, Cov(T_i) \setminus Cov(T_j) \subset Cov(T_k)\}$. If true, T_i is an ineffective partially redundant test that can be eliminated from TS without decreasing the fault detection effectiveness of TS, as $Cov(T_j) \cap Cov(T_k) \supseteq Cov(T_i)$. If false, T_i is a partially redundant test that executes a feature set not covered by any other test in TS. This unique set could be a single feature or a combination of features. Since our work relates a HCS with a high degree of feature interaction, together with a test suite developed over time to cover these various interactions, it is expected that combinations with varying (high) degree of feature interactions will usually exist in a test suite.

To determine whether T_i is an *effective* or *ineffective* partially redundant test case, we make the following hypothesis: Test cases covering combinations that have historically exhibited good fault-revealing performance can be classified as *effective*, and otherwise as *ineffective*. The hypothesis is based on the fact, suggested by previous studies [22], [17], [23], [5], that test execution history can help improve the effectiveness of regression testing. Therefore, to optimally detect and minimize test redundancy, we complement coverage analysis with the information indicating fault detection capability of tests exhibited in past test executions for specific configurations. $Fde(T_i) = \{Fde\_T_{i1}, Fde\_T_{i2}, ..., Fde\_T_{ij}\}$ denotes execution history for T_i over j test runs, where each failure is associated with one or more failing configurations $C_{fail}(Fde\_T_{ij}) = \{C_1, C_2, ..., C_m\}$, where $C_i = \{f_1, f_2, ..., f_n\}$ is a set of features, and $status \in [0, 1]$.

5.1 The Approach

We describe the overall approach to test redundancy reduction based on coverage and history analysis as follows:

1) Starting from Ch_i and the existing test suite TS, we identify RTS' as a set of tests affected by Ch_i, based on the associations between features covered by a test and software modules implementing these features, similarly to the associations between features and test cases. RTS' represents an initial regression test

suite that needs to be processed further to satisfy the properties $P2$, $P3$, and $P4$.

2) Next we analyze total redundancy in RTS' to satisfy the Property $P2$. We identify tests whose covering set of features is entirely covered by another tests $Cov(T_i) \subseteq RTS'$ (Figure 2a), and remove such tests from RTS'.

3) Then we look for tests in RTS' that cover features not covered by other tests in RTS' (Figure 2b), extract these tests from RTS', and form RTS, satisfying the Property $P3$.

4) At this point, RTS' consists of partially redundant tests (Figure 2c). To partition effective and ineffective tests, for each $T_i \in RTS'$ we obtain execution history $Fde(T_i) = \{Fde\_T_{i1}, Fde\_T_{i2}, ..., Fde\_T_{ij}\}$, with $j = 6$. This value has been experimentally evaluated as the optimal size of history window for test optimization for HCS [23]. Longer windows capturing more of older failure information showed to provide less accurate indication of potentially failing test cases. Next, for each $T_i \in RTS'$ we obtain a set of single-feature or multiple-feature configurations that are uniquely covered by T_i, $C_{unique}(T_i) = \{C_1, C_2, ..., C_p\}$, and we look if they belong to $C_{fail}(Fde\_T_{ij}) = \{C_1, C_2, ..., C_m\}, j = \{1..6\}$, which is a set of failing configurations in the previous j test runs.
$C_i = \{f_1, f_2, ..., f_n\}$ is a single-feature or multiple-feature configuration. If $c \in C_{unique}(T_i)$ belongs to the set of failing test configurations $C_{fail}(Fde\_T_{ij})$, the test case T_i would be weighted $\omega(T_i) = 1 \setminus j$, i.e. proportionally to the value of j indicating how recently the failure happened.
After obtaining weights for all partially redundant tests in RTS', we sort RTS' such that higher priority tests come first in a sequence. In case that one or more test cases are assigned equal weights, we check their failure rate $Fr = total\_num\_of\_failures/total\_num\_of\_executions$, and give higher priority to those tests that have failed more frequently. In case that one or more of these test cases have the same failure rate, we assign them different weights at random.

5) Given that our approach is aimed at CI testing environments with strict time constraints, we append to RTS the top n test cases from RTS' (with the highest priority) to fill the available test budget for RTS. The test budget is defined for every CI test run. n is determined by adding up test execution times of tests from RTS and RTS' as obtained from their historical execution records. At this point, RTS is the selected regression test suite that satisfies all defined properties $P1 - P4$, as the solution of coverage- and history-based test redundancy reduction problem.

The following example illustrates the described approach.

Example: Test suite TS consists of three tests $T1 = \{A, B\}$, $T2 = \{B, C\}$, $T3 = \{A, C\}$. Using only coverage analysis, all three tests are mutually partially redundant (by Definition 2), and therefore equally good candidates for ineffective partially redundant tests. However, if we complement coverage analysis with historical test execution information in Table 1, it becomes possible to assess relative fault-revealing effectiveness of these three tests. In Table 1, columns represent historical consecutive test runs, with $E1$ being the most recent run, and $E6$ the oldest test run. For each test case (shown in rows), f denotes execution result $er = fail$, and p denotes execution result $er = pass$, in the given test run. For convenience, we consider that er can be only p or f. The last two columns, show for each configuration, a weighted fault-detection effectiveness ω, and failure rate F_r, calculated as follows:

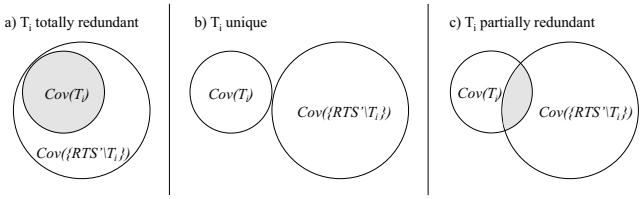

Figure 2: Test suite properties: a) T_i totally redundant, b) T_i unique, c) T_i partially redundant.

$$\omega = \sum_{i=1}^{6} status * \frac{1}{i} \quad , \quad status = \begin{cases} 0, & \text{if } er = p \\ 1, & \text{otherwise} \end{cases}$$

For the purpose of illustration, F_r calculations assume that the total number of test executions for each configuration in this example is 6.

ω indicates that T_1 is an *effective* test case, because if has historically exhibited the highest fault detection effectiveness out of all three test cases, while for T_2 and T_3 the result is inconclusive, based on just ω. Then, if we consider that $F_r(BC) > F_r(AC)$, we conclude that $T_3 = \{AC\}$ is an *ineffective* test case.

Table 1: Test execution status (p=pass, f=fail) for $T1 = \{A, B\}$, $T2 = \{B, C\}$, $T3 = \{A, C\}$ for 6 consecutive test runs. ω: weighted fault detection effectiveness, F_r: failure rate.

	E1	E2	E3	E4	E5	E6	ω	F_r
AB	f	f	p	f	f	p	1.95	4/6
BC	p	p	f	p	f	f	0.7	3/6
AC	p	f	p	p	f	p	0.7	2/6

6 EMPIRICAL EVALUATION

This section presents the set of experiments conducted to evaluate the effectiveness of the proposed approach for test redundancy reduction.

In particular, we aim to evaluate i) *fault-detection effectiveness*, and ii) *test execution time* (timeliness of test feedback) of regression test suites for which redundancy has been reduced using our approach, in comparison with i) *existing industry practice* of testing HCS in CI, ii) *retest-all approach*, and iii) *random test selection*.

The experiments were designed to answer the following research questions:

RQ1 Can regression tests selected based on the proposed approach enable faster test feedback compared to industry practice of testing HCS in CI, without compromising fault-detection effectiveness?

RQ2 Can regression tests selected based on the proposed approach reduce test feedback time compared to retest-all approach, without compromising fault-detection effectiveness?

RQ3 How do regression tests selected based on the proposed approach compare to randomly selected tests in terms of fault detection effectiveness, when controlling for the total test suite execution time?

We address the three research questions RQ1-RQ3 in experiments E1, E2, and E3, respectively.

6.1 Experimental Subjects

In evaluation, we use a realistic test suite of an existing industrial HCS. In addition, to perform more extensive evaluation of the approach on larger test instances with varied number of test cases and requirements, as well as varied failure rates and degrees of feature interaction coverage across test suites, we use 110 artificial test suites created to resemble the realistic test suite.

6.1.1 Industrial Subject. The industrial experimental subject is the video conferencing software system *CBC* introduced in Section 3 (Running Example). *CBC* contains a test suite *BCTS* consisting of 460 test cases, which cover the set of 75 features. Each test case contains historical execution records for the last ten runs. The historical information includes test result status, execution duration, required resources, test identifiers, tested software version, and execution schedulers. In total, the execution history contains results of 4600 executions (460 tests * 10 consecutive executions per test). Minimum degree of feature interaction for tests in *BCTS* varies from 1 (single features) to 3 (combination of 3 features), and maximum degree of feature interaction varies from 3 to 7.

6.1.2 Artificial Subjects. The artificial experimental subjects are developed to resemble the described industrial test suite *BCTS* as follows. We created 110 test suites, larger in terms of the number of test cases and covered features compared to *BCTS*. The artificial suites are grouped by size in 11 groups $\{ATS_1, ATS_2, ..., ATS_{11}\}$, and each ATS_i consists of ten test suites $ATS_i = \{ATS_{i\_1}, ATS_{i\_2}, ..., ATS_{i\_10}\}$. The test suites cover a set of 150 features. Association tags are established between features and test cases, maintaining feature interactions coverage as found in *BCTS*. Minimum degree of feature interaction per ATS_i varies from 1 (single features) to 3 (combination of 3 features), and maximum degree of feature interaction varies from 3 to 7 features. To enable more thorough evaluation, the size of each test suite ATS_{i_j} is increased compared to *BCTS*. The size of test suites varies from 500 to 1000 test cases with an increment of 50. Each ATS_{i_j}, $i = 1..11, j = 1..10$ contains test execution history that was developed according to the execution history of *BCTS*. The history consists of test identifiers, software versions, test result status (pass/fail), and test execution duration. The history is developed for six consecutive runs. For all artificial test suites, failure rates and the degree of feature interactions are developed to resemble those observed in the *BCTS*. In particular, for each ATS_i, failure rates range from 29% to 63%, with deviation from 2 to 11. Table 2 provides summary information about the artificial test suites.

6.2 Methodology and Measures

In E1, we compare RHS with current industry practice (CIP), in terms of total test execution time and fault-detection effectiveness of regression suite. At the first step of the experiment, we retrieve modified features from history, and select a set of tests from *BCTS* affected by the modifications. The resulting test suite represents the initial regression test suite, which we refer to as *BC_RHS*. Then we apply RHS approach to *BC_RHS*, and using six historical execution records of *BCTS*, we analyze which feature combinations showed good fault-detection performance in the past. Based on this information, we eliminate totally redundant and ineffective partially redundant tests from *BC_RHS*, creating the final regression test suite. Further, we measure the loss of fault-detection of the final "non-redundant" test suite, as the percentage of failures that were caused by configurations not covered by tests in *BC_RHS*, due to reductions, and covered by *BCTS*. Finally, for the final "non-redundant" test suite, we measure the percentage of reduction of the test suite size compared to the size of *BCTS*. As we have available test execution history from ten consecutive runs for *BCTS*, and our approach uses the history window of size 6 (explained previously), we run the experiment 5 times. We start from the oldest six historical execution records, and in every experiment replace the oldest record with one newer, until we have used all the available records.

In E2, we compare the proposed approach for test redundancy reduction based on coverage and history analysis (RHS) with retest-all approach (RA) in terms of total test execution time and fault-detection effectiveness of a "non-redundant" regression suite. We compare with RA because RA is a common approach to regression testing used in practice, instead of tedious manual regression test selection/reduction. However, to allow for fairer comparison, we implemented a modified retest-all (MRA), which subsets the original test suite that would normally be executed by RA such that only the tests affected by a change are selected as regression tests. This was implemented using the association tags between features covered by a test and software modules implementing these features, similarly to the associations existing between test cases and their covering features.

At the first step of the experiment, we randomly choose 15 features as modified (10% of the total number of features in the artificial test suites). Then, from each test suite $ATS_{i\_j}, i = 1..11, j = 1..10$, we automatically select a set of tests affected by changed features. The selection is based on the existing association tags between test cases and their covering features. The resulting test suites represent the initial regression test suites, which we refer to as $InitRHS_{i\_j}, i = 1..11, j = 1..10$. Then we apply RHS approach to each $InitRHS_{i\_j}$. We use six consecutive historical execution records of each corresponding test suite indicating which feature combinations exhibited good fault-revealing capability, and according to the described approach, from each $InitRHS_{i\_j}$ we eliminate any redundant tests, as well as tests contributing to partial ineffective redundancy. The resulting test suites represent the final selected regression test suites, which we refer to as $FinRHS_{i\_j}, i = 1..11, j = 1..10$. Finally, for each $FinRHS_{i\_j}$ we measure the loss of fault-detection compared to the fault-detection capability of its corresponding $InitRHS_{i\_j}$. The loss is measured as the percentage of failures that were caused by configurations not covered by test cases in $FinRHS_{i\_j}$, due to reduction,

Table 2: Experimental subjects ATS_1-ATS_{11} each represents a set of 10 test suites of the same size but with varying failure rate and feature interaction coverage.

	ATS_1	ATS_2	ATS_3	ATS_4	ATS_5	ATS_6	ATS_7	ATS_8	ATS_9	ATS_{10}	ATS_{11}
Number of test cases	500	550	600	650	700	750	800	850	900	950	1000
Min % of failing test cases	29	29	29	29	29	29	29	29	29	29	29
Max % of failing test cases	63	63	63	63	63	63	63	63	63	63	63
Average % of failing test cases	35	29	43	58	33	37	45	63	58	36	51
Avg min feature interact. cov.	1	2	1	2	1	2	3	2	1	2	1
Avg max feature interact. cov.	5	6	6	5	7	4	4	3	5	7	5

and covered by $InitRHS_{i\_j}$. Then for each $FinRHS_{i\_j}$ we measure the percentage reduction of a test suite size compared to the size of its corresponding $InitRHS_{i\_j}$.

In E3, we compare RHS with random test selection (RS) in terms of fault-detection effectiveness, while controlling for the total execution time of regression test suite. The motivation to compare with RS comes from the observation that random selection is sometimes used as an alternative to automated regression test reduction, to reduce the cost of regression testing in cases where running the whole regression suite selected after code modifications would exceed test budget.

At the first step of the experiment, we obtain $InitRHS_{i\_j}, i = 1..11, j = 1..10$ and $FinRHS_{i\_j}, i = 1..11, j = 1..10$ created in E1, and for each $FinRHS_{i\_j}$ we measure total test execution time $ET(FinRHS_{i\_j})$. Then we start randomly selecting tests from $InitRHS_{i\_j}$, and when each test is selected we accumulate its execution time $ET(InitRS_{i\_j})$. We continue randomly selecting tests until $ET(InitRS_{i\_j})$ equals $ET(FinRHS_{i\_j})$. We refer to the resulting test suites as $FinRS_{i\_j}$. Then for each $FinRS_{i\_j}$ we measure the loss of fault detection compared to the fault-detection capability of its corresponding $InitRHS_{i\_j}$. The loss is measured as the percentage of failures that were caused by configurations not covered by test cases in $FinRS_{i\_j}$, and covered by $InitRHS_{i\_j}$. Finally, we compare *RHS* and *RS* in terms of the loss of fault detection effectiveness. To account for randomness in RS, we repeat each experiment 100 times and determine the statistical significance of the results in non-parametric Mann-Whitney U-Tests, with the significance level 0.01.

7 RESULTS AND ANALYSIS

In this section we present the results of the experiments E1, E2, and E3, addressing research questions RQ1, RQ2, and RQ3, respectively. We present the results of the experiment E1 in Figure 3, the results of the experiment E2 in Figure 4 and Figure 5, and the results of the experiment E3 in Figure 6.

7.1 Industry Practice

In experiment E1, we evaluate whether RHS is able to reduce regression test feedback while achieving equal fault-detection effectiveness compared to CIP. In terms of fault-detection effectiveness, RHS exhibited equal performance compared to CIP for all five experiments. In terms of total regression test execution time, RHS showed improvements over CIP by 30% on average (15% in worst case, and 39% in best case). The results of time-effectiveness are shown in Figure 3. X-axis represents five experiments corresponding to different historical execution data, and y-axis denotes the percentage of test suite total execution time, with CIP as baseline (100%).

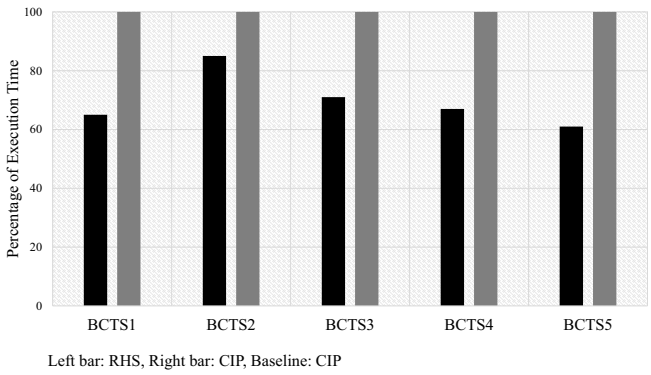

Figure 3: Percentage reduction of regression test execution time for RHS compared to CIP.

In summary, the results of the experiment E1 indicate that the proposed approach for redundancy reduction based on coverage and history analysis successfully reduces regression test feedback compared to industry practice, without compromising fault-detection effectiveness.

7.2 Time Effectiveness (Test Feedback Time)

Figure 4 shows time-effectiveness of RHS compared to MRA for each subject. X-axis represents test suites, and y-axis is the percentage reduction of total regression test execution time for RHS, compared to MRA. The results show that RHS can reduce total regression test execution time by 31% on average compared to MRA. In the best case, using RHS led to reducing the execution time by 45%, and in the worst case by 15%. Higher reduction is achieved for test suites with a higher degree of feature interaction, and vice versa. This is because many of the feature combinations with a lower interaction degree are subsumed within the ones covering larger combinations, which simplifies detecting redundant tests. We consider that execution time is directly proportional to the size of

a test suite, because tests have similar execution time. Shorter test suite execution time leads to shorter test feedback in continuous integration testing.

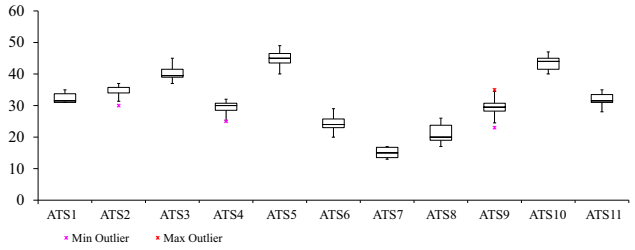

Figure 4: Percentage reduction of the total regression test execution time for RHS compared to MRA.

7.3 Fault Detection Effectiveness

Figure 5 shows fault detection effectiveness of RHS compared to MRA for each subject. X-axis represents test suites, and y-axis is the percentage of fault-detection effectiveness loss for RHS, compared to MRA. The results indicate that RHS exhibits comparable performance to MRA, with less than 0.5% loss in fault-detection effectiveness on average. For about the half of experiment subjects, RHS achieved equal performance to MRA. The worst performance is exhibited for ATS_8, where using RHS led to max 2% fault-detection loss for some test suites. We explain this behavior by a low degree of max feature interaction coverage of ATS_8, which made it difficult to satisfy all interactions while eliminating redundant tests. Lower fault-detection effectiveness of ATS_8 is compensated with its shorter execution time. As can be seen in Figure 4, ATS_8 whose max feature interaction coverage is 3-wise, takes less execution time compared to ATS_7, whose max feature interaction coverage is 4-wise, but which on the other hand exhibits less loss in fault-detection effectiveness.

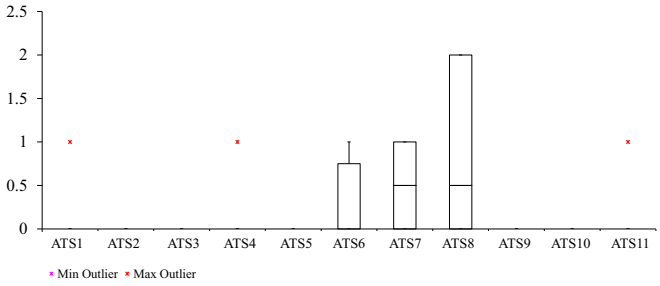

Figure 5: Percentage reduction of fault detection effectiveness for RHS compared to MRA.

In summary, the results of the experiment E2 show that the proposed approach for redundancy reduction based on coverage and history analysis can reduce test feedback compared to an advanced retest-all approach without significantly compromising the fault-detection effectiveness of a regression suite.

In the experiment E3, we compare RHS and RS in terms of fault detection effectiveness, while controlling for the total test execution time of regression test suites. We present the results of the experiment in Figure 6. The boxplot shows the distribution of the percentage of fault detection effectiveness loss of RS compared to RHS for each subject, averaged over ATS_i. The results indicate that RHS achieves significant improvement in fault-detection effectiveness of regression test suites over RS. RHS consistently exhibits better performance across all subjects, producing test suites capable of detecting 72% more faults on average compared to RS, with a deviation of about 10 on average. The statistical Mann-Whitney U-tests showed that the improvements of fault-detection effectiveness are statistically significant.

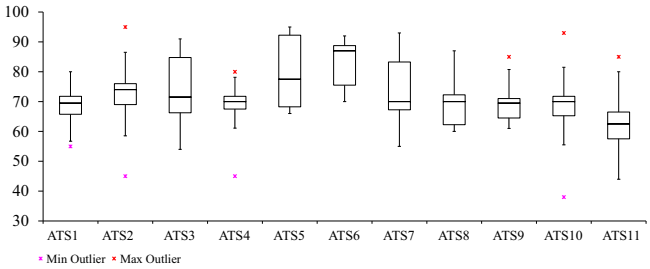

Figure 6: Distribution of the percentage of fault detection effectiveness loss of RS compared to RHS.

In summary, the results of the experiment E3 indicate that the proposed approach for redundancy reduction based on coverage and history analysis can significantly improve fault detection effectiveness of a test suite compared to random test selection, for an equal test suite execution time.

8 THREATS TO VALIDITY

External Validity: A threat to external validity of our results is that the used experiment subjects may not be sufficiently representative. However, the approach was developed to address the problem of test redundancy detected in a particular setting of highly configurable video conferencing software. Therefore, we could have hardly found other appropriate experimental subjects to validate the proposed approach. Still, to mitigate this threat, we developed and used in validation larger artificial subjects resembling the realistic one, while varying different parameters such as the number of test cases, degree of feature interaction coverage, and number of failing tests.

Internal Validity: One threat to internal validity are potential faults in our implementation of RHS, MRA, and RS. To mitigate this threat, we have carefully analyzed and thoroughly tested all implementations used in experimentation. Another threat to internal validity may be the effect of randomness in evaluation. To mitigate this threat, we repeated experiments 100 times and used non-parametric Mann-Whitney U-Tests, with the significance level 0.01, to confirm the statistical significance of the results.

Construct Validity: A threat to construct validity is the choice of experimental measures. We believe that we have used the measures commonly used in evaluation of regression test selection, such as execution time reduction and fault detection effectiveness of a test suite.

9 RELATED WORK

Our work touches upon and extends the following two principal research areas.

Test Redundancy. Numerous approaches to test redundancy detection are based on different test coverage metrics, such as statement coverage [25], branch coverage [29], decision coverage, condition coverage, or path coverage. However, Koochakzadeh et al. report that coverage based metrics are imprecise in test redundancy detection and should be combined with additional information [19], [18]. Fraser et al. present an approach where redundancy in tests generated with model checker is identified based on the analysis of paths of Kripke structure [7]. In a later stage, the algorithm modifies redundant parts of tests to eliminate redundancy. However, the authors report high run-time complexity of the algorithm as a drawback. Jeffrey et al. propose to use a modified HSG heuristic [11] for selectively retaining redundant tests in a suite based on two sets of test requirements, branch coverage and all-uses coverage, while preserving fault-detection effectiveness [15], [16]. However, the approach has limitations due to a large size of a reduced test suite. Test redundancy has been intensively studied in the context of test suite minimization techniques [2, 9, 13, 14, 24, 25, 29, 33], which aim to reduce the size of the original test suite (remove redundant tests) but preserve its fault detection capabilities. However, it was shown that test minimization techniques often compromise fault detection effectiveness of a test suite. Rothermel et al. analyze the effect of test suite minimization on its fault detection capability, and show that minimizing a test suite can significantly lower its effectiveness in detecting faults [30], [29]. Heimdahl et al. find that test suite reduction based on structural coverage exhibits high loss in fault detection capability [12]. These findings imply that the performance of test suite reduction could be improved by improving the effectiveness of precisely identifying redundant tests. This is particularly important for HCS, where redundancy incurs high testing and maintenance costs.

Regression Test Selection. Different approaches have been proposed to improve the effectiveness of regression test selection. Ren et al. use change impact analysis to identify regression tests affected by a change, by constructing a call graph for each test that consists of interdependent changes at the method level [27]. Orso et al. use change impact analysis and field data, although the approach is considered unsafe [26]. Other techniques include regression test selection based on specification changes [4], [21], [31], as well as code changes [8], [4], [10], [28], or adaptive test prioritization [32]. Several researchers have propose to use model-based techniques for regression test selection. Korel et al. use an Extended Finite State Machine model dependence analysis for identifying modifications [20]. Andrews et al. apply model-based selective regression testing to a railroad crossing control system based on behavioral model of a system and a behavioral test suite [1]. However, common limitation of model-based regression testing techniques is an ambiguous quality of regression test suites, because these techniques use only a modified system model instead of an original [20]. Despite numerous approaches proposed for improving regression test selection, limited attention has been given to the problem of regression testing of highly configurable software developed in continuous integration, especially in the case of highly-interleaved tests.

10 CONCLUSION AND DISCUSSION

This paper addresses the problem of efficient regression test selection in software environments characterized by high degree of configurability and high need for short test feedback (typically HCS systems in CI environments). We present a practical approach for improving the efficiency of regression test selection based on identifying and eliminating test redundancy. The approach uses coverage metrics and history analysis to determine an effective regression test suite, normally consisting of fewer tests and exhibiting comparable performance compared to retest-all, often used in regression testing in practice. This is achieved by analyzing which combinations of features, out of those tested multiple times across a test suite, have historically proved high-fault detection effectiveness. The approach has been evaluated using a large set of subjects, including one industrial HCS and eleven artificial HCS systems resembling the industrial one in structure and complexity. The results show that the proposed approach improves current industry practice, enabling faster regression test feedback, without compromising fault-detection effectiveness. The results further show that our proposed approach can reduce test feedback compared to an advanced retest-all approach without significantly compromising the fault-detection effectiveness of a regression suite. The results also show that the proposed approach can significantly improve fault detection effectiveness compared to random testing, which is often used to reduce the effort of regression testing in time- and resource constrained environments.

Further Work: In further work, we will improve the precision of the approach by enhancing the analysis of partial redundancy, introducing weightening of feature interaction failures across historical executions. The idea is that failures occurred in older test runs are worse predictors of failures in the current run, compared to failures occurred in more recent runs. The assumption is that associated errors are more likely to be fixed the more time has elapsed from failures. However, this assumption needs to be shown with empirical evidence, to be able to effectively use weightening of historical failures for test redundancy detection. In further work, we will also evaluate the approach on more industrial instances of regression testing from different CI domains, which will improve the generalizability of our approach. However, it is normal to expect that our approach will need tailoring to the requirements and constraints of the specific domain, such as feature interaction degree of configurable software, frequency of code commits in CI, size of test suites, etc.

ACKNOWLEDGMENT

This work is supported by Certus SFI project, funded by the Norwegian Research Council. We also thank various people at Cisco QA Team for their input and contribution to this work.

REFERENCES

[1] A. Andrews, S. Elakeili, and A. Alhaddad. 2015. Selective Regression Testing of Safety-Critical Systems: A Black Box Approach. In *Software Quality, Reliability and Security - Companion (QRS-C), 2015 IEEE International Conference on.* 22–31.

[2] H. Baller, S. Lity, M. Lochau, and I. Schaefer. 2014. Multi-objective Test Suite Optimization for Incremental Product Family Testing. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. 303–312. https://doi.org/10.1109/ICST.2014.43

[3] K. Z. Bell and M. A. Vouk. 2005. On effectiveness of pairwise methodology for testing network-centric software. In *2005 International Conference on Information and Communication Technology*. 221–235.

[4] Yanping Chen, Robert L. Probert, and D. Paul Sims. 2002. Specification-based Regression Test Selection with Risk Analysis. In *Proc. of the 2002 Conf. of the Centre for Advanced Studies on Collaborative Research*. IBM Press. http://dl.acm.org/citation.cfm?id=782115.782116

[5] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for Improving Regression Testing in Continuous Integration Development Environments. In *ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 235–245. http://doi.acm.org/10.1145/2635868.2635910

[6] Emelie Engstrom and Per Runeson. 2013. Test Overlay in an Emerging Software Product Line - An Industrial Case Study. *Information and Software Technology* 55, 3 (March 2013), 581–594. http://dx.doi.org/10.1016/j.infsof.2012.04.009

[7] Gordon Fraser and Franz Wotawa. 2007. Redundancy Based Test-suite Reduction. In *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering (FASE'07)*. Springer-Verlag, Berlin, Heidelberg, 291–305. http://dl.acm.org/citation.cfm?id=1759394.1759425

[8] M. Gligoric, L. Eloussi, and D. Marinov. 2015. Ekstazi: Lightweight Test Selection. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. 713–716. https://doi.org/10.1109/ICSE.2015.230

[9] Arnaud Gotlieb and Dusica Marijan. 2014. FLOWER: Optimal Test Suite Reduction As a Network Maximum Flow. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA 2014)*. ACM, New York, NY, USA, 171–180. http://doi.acm.org/10.1145/2610384.2610416

[10] Rajiv Gupta, M. Jean Harrold, and Mary Lou Soffa. 1996. Program Slicing-Based Regression Testing Techniques. *Software Testing, Verification and Reliability* 6, 2 (1996), 83–111.

[11] M. Jean Harrold, Rajiv Gupta, and Mary Lou Soffa. 1993. A Methodology for Controlling the Size of a Test Suite. *ACM Transactions on Software Engineering and Methodology* 2, 3 (July 1993), 270–285. http://doi.acm.org/10.1145/152388.152391

[12] M. P. E. Heimdahl and D. George. 2004. Test-suite reduction for model based tests: effects on test quality and implications for testing. In *International Conference on Automated Software Engineering 2004*. 176–185.

[13] Aymeric Hervieu, Dusica Marijan, Arnaud Gotlieb, and Benoit Baudry. 2016. Practical Minimization of Pairwise-covering Test Configurations Using Constraint Programming. *Information Software Technology* 71, C (March 2016), 129–146. https://doi.org/10.1016/j.infsof.2015.11.007

[14] H. Y. Hsu and A. Orso. 2009. MINTS: A general framework and tool for supporting test-suite minimization. In *International Conference on Software Engineering*.

[15] D. Jeffrey and Neelam Gupta. 2005. Test suite reduction with selective redundancy. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*. 549–558.

[16] Dennis Jeffrey and Neelam Gupta. 2007. Improving Fault Detection Capability by Selectively Retaining Test Cases During Test Suite Reduction. *IEEE Transactions on Software Engineering* 33, 2 (Feb. 2007), 108–123. http://dx.doi.org/10.1109/TSE.2007.18

[17] Jung-Min Kim and Adam Porter. 2002. A History-based Test Prioritization Technique for Regression Testing in Resource Constrained Environments. In *International Conference on Software Engineering*. ACM, New York, NY, USA, 119–129. http://doi.acm.org/10.1145/581339.581357

[18] Negar Koochakzadeh and Vahid Garousi. 2010. A Tester-assisted Methodology for Test Redundancy Detection. *Advanc. Software Engineering* 2010, Article 6 (Jan. 2010), 13 pages. http://dx.doi.org/10.1155/2010/932686

[19] N. Koochakzadeh, V. Garousi, and F. Maurer. 2009. Test Redundancy Measurement Based on Coverage Information: Evaluations and Lessons Learned. In *International Conference on Software Testing Verification and Validation*. 220–229.

[20] B. Korel, L. H. Tahat, and B. Vaysburg. 2002. Model based regression test reduction using dependence analysis. In *Software Maintenance, 2002. Proceedings. International Conference on*. 214–223.

[21] Chengying Mao and Yansheng Lu. 2005. Regression testing for component-based software systems by enhancing change information. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*. 8 pp.–.

[22] D. Marijan, A. Gotlieb, and S. Sen. 2013. Test Case Prioritization for Continuous Regression Testing: An Industrial Case Study. In *International Conference on Software Maintenanse 2013*. 540–543.

[23] D. Marijan and M. Liaaen. 2016. Effect of Time Window on the Performance of Continuous Regression Testing. In *International Conference on Software Maintenance and Evolution*. 568–571.

[24] W. Masri, A. Podgurski, and D. Leon. 2007. An Empirical Study of Test Case Filtering Techniques Based on Exercising Information Flows. *IEEE Transactions on Software Engineering* 33, 7 (July 2007), 454–477.

[25] A. Jefferson Offutt, Jie Pan, and Jeffrey M. Voas. 1995. Procedures for reducing the size of coverage-based test sets. In *International Conference Testing Computer Software*. 111–123.

[26] Alessandro Orso, Taweesup Apiwattanapong, and Mary Jean Harrold. 2003. Leveraging Field Data for Impact Analysis and Regression Testing. In *Proceedings of the 9th European Software Engineering Conference (ESEC/FSE-11)*. ACM, New York, NY, USA, 128–137. http://doi.acm.org/10.1145/940071.940089

[27] Xiaoxia Ren, Fenil Shah, Frank Tip, Barbara G. Ryder, and Ophelia Chesley. 2004. Chianti: A Tool for Change Impact Analysis of Java Programs. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA '04)*. ACM, New York, NY, USA, 432–448. http://doi.acm.org/10.1145/1028976.1029012

[28] Gregg Rothermel and Mary Jean Harrold. 1997. A Safe, Efficient Regression Test Selection Technique. *ACM Transactiona on Software Engineering and Methodology* 6, 2 (April 1997), 173–210. http://doi.acm.org/10.1145/248233.248262

[29] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong. 1998. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *International Conference on Software Maintenance*. 34–43.

[30] Gregg Rothermel, Mary Jean Harrold, Jeffery von Ronne, and Christie Hong. 2002. Empirical Studies of Test-Suite Reduction. *Journal of Software Testing, Verification, and Reliability* 12 (2002), 219–249.

[31] A. S. M. Sajeev and B. Wibowo. 2003. Regression test selection based on version changes of components. In *Software Engineering Conference, 2003. Tenth Asia-Pacific*. 78–85.

[32] Amanda Schwartz and Hyunsook Do. 2016. Cost-effective regression testing through Adaptive Test Prioritization strategies. *Journal of Systems and Software* 115 (2016), 61 – 81. https://doi.org/10.1016/j.jss.2016.01.018

[33] Lingming Zhang, Darko Marinov, Lu Zhang, and Sarfraz Khurshid. 2011. An Empirical Study of JUnit Test-Suite Reduction. In *Proceedings of the 2011 IEEE 22Nd International Symposium on Software Reliability Engineering (ISSRE '11)*. IEEE Computer Society, Washington, DC, USA, 170–179. http://dx.doi.org/10.1109/ISSRE.2011.26