Regression Test Selection of Manual System Tests in Practice

Elmar Juergens, Benjamin Hummel
Technische Universität München
{juergens, hummelb}@in.tum.de

Florian Deissenboeck, Martin Feilkas
CQSE GmbH
{deissenboeck, feilkas}@cqse.eu

Christian Schlögel, Andreas Wübbeke
Wincor Nixdorf International GmbH
christian.schloegel@wincor-nixdorf.com
andreas.wuebbeke@wincor-nixdorf.com

Abstract—Regression testing analyzes whether software maintenance has inadvertently broken existing functionality. Since it is costly—especially for manual testing—it is typically limited to a subset of test cases. Since impact analysis of code modifications on test cases is far from trivial for real world software, regression test selection is hard. However, if it misses affected test cases, bugs may remain unnoticed. In response, the research community has proposed numerous test selection approaches. Regression test selection is especially relevant for *manual* tests, since their execution costs limit the number of tests that can be executed in practice. However, evaluations of existing work focus on *automated* tests. Its applicability to manual tests is thus unclear. We present an industrial case study that demonstrates the challenges that regression test selection techniques face when applied to manual system tests. Furthermore, we sketch how, given these challenges, manual regression test selection can be improved.

Keywords-regression test selection, software maintenance

I. INTRODUCTION

Wincor Nixdorf produces embedded systems such as ATMs, deposit systems or retail terminals. As part of the quality assurance process, Wincor Nixdorf performs system tests to validate their correct behavior. Since they involve interaction with the embedded systems, e. g. PIN entry and cash withdrawal during an ATM test, they are performed manually. Manual testing is costly—the system test suite for a single product, which is performed before each major software release, takes several person months to execute.

If a software defect occurs in the field, a repaired version of the software must be delivered rapidly. A fix for a known defect can, however, introduce new defects. To prevent new defects from being released, Wincor Nixdorf performs regression testing. Unfortunately, it is impossible to execute the entire system test suite: it would be too costly and leave the faulty version in production for too long. Test engineers thus select a subset of all tests for regression testing.

Regression test selection is hard. System tests are written on the level of application features; fixes are written on the level of source code. The detailed interplay between tests and code, however, is almost impossible to determine manually. Manual regression test selection thus necessarily involves a certain amount of guesswork. Given the importance of regression testing, how can we better support test engineers during regression test selection?

The research community has proposed several approaches for selective regression testing (SRT) [1], [2], [5], [6], [8]. SRT determines which test executions are potentially affected by a given modification (e. g., a fix) to the system, thus supporting regression test selection.

While SRT approaches are applicable to manual tests in principle, existing empirical evaluations are limited to automated tests [1], [4]. Automated and manual tests, however, differ in several aspects. First, automated tests often focus on small areas of code that implement a single feature (*cf.*, unit tests), whereas manual system tests typically cover large parts of the system in an end-to-end fashion. This can affect the savings achievable through SRT techniques. Second, an automated test driver always executes tests in the same fashion. In contrast, different testers might perform the same manual test case slightly differently, causing it to execute different code paths. Given these differences, how well can SRT techniques be applied to manual tests?

Research Problem: Evaluations of selective regression testing (SRT) focus on *automated* tests. It is unclear how well they can support test selection for *manual* system tests.

Contribution: We present an industrial case study that analyzes applicability of SRT techniques to manual system tests. We conclude that, in practice, the expected reductions in test suite size of fully automated SRT are insufficient. Furthermore, we outline a semi-automated approach that can achieve further reductions.

II. RELATED WORK

Numerous SRT approaches have been proposed [2], [3], [5]–[9]. They comprise three general steps. First, test execution *traces* are recorded that show which test cases exercise which code fragments. Second, dependencies between code fragments (e. g. due to inheritance or shared global state) are extracted. Third, all test cases that exercise modified code fragments, or code fragments that (transitively) depend on modified code fragments, are marked as *affected* test cases. The specifics of each step, e. g. the tracing granularity [1] or dependency extraction [2], [5], [8], vary across approaches. So called *safe* SRT approaches provide a guarantee that the set of selected test cases comprises all error-uncovering ones. We employ an SRT technique that marks a test case as affected, if at least one of the methods it covers was modified. It serves as a lower bound for safe approaches.

Several researchers have evaluated SRT approaches. In [6], Rothermel and Harrold present a framework for qualitative comparison of SRT techniques and stress the importance of empirical comparisons. In [1], Bible et al. compare two tools that implement different SRT techniques on a set of small programs using a suite of automated tests. In [4], Graves et al. use the same small programs to compare three different SRT techniques. Furthermore, in [3], [7], [8] the authors evaluate the SRT technique they propose using automated tests. In contrast, this paper studies the applicability of SRT techniques to *manual* system tests. In addition, our study object is several orders of magnitude larger than those of previous studies [1], [3], [4], [7].

III. CASE STUDY

A. Research Questions

We analyze the following research questions to better understand the applicability of SRT to manual system tests:

RQ 1: How large is the set of test cases that SRT determines for past fixes?

SRT techniques are conservative. The set of selected test cases can thus be significantly larger than the set of test cases that actually uncover errors. This question analyzes the reduction achievable through SRT on modifications that were performed to the system in the past.

RQ 2: How large is the set of test cases that SRT selects in general?

The past behavior might not be representative, or future fixes could simply be of a different nature. This question thus analyzes the reduction that can be expected in general.

RQ 3: Are test cases deterministic?

SRT approaches assume that test cases always execute the same code. In practice, system test cases are often under-specified—full specification is often unfeasible. Missing information can, however, be handled differently by different testers. As a consequence, tests exercise different paths through the code. This question quantifies the consequences of under-specification on test determinism in practice.

RQ 4: How stable is coverage during system evolution?

To achieve coverage of modified code, testers need to execute test cases that exercise the modified methods. This question analyzes if coverage on a previous system version is a good indicator for coverage after modification.

B. Study Object

We use a system from Wincor Nixdorf as study object. Wincor Nixdorf is an international company producing IT solutions for the banking and retail industry. Its solutions are integrated products that comprise both hardware and software. The company has more than 9.000 employees and is market leader for cash handling and cash cycle solutions. The analyzed system is a client-server application that comprises several MLoC of Java code. In production,
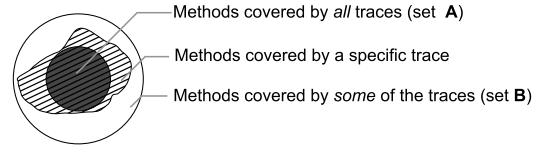

Figure 1: The sets used in RQ3 and RQ4

the server runs in a cluster that is accessed by several thousand client systems.

The system consists of the main process of the application server and several auxiliary processes. The application server itself can span multiple threads. The traces used in all research questions consist only of methods executed in threads related to HTTP requests and CORBA communication (with the client systems, e.g. an ATM) to filter out methods related to background tasks that are scheduled at certain intervals.

C. Study Design and Procedure

We analyze several consecutive system snapshots from a development branch created for bug fixing, called S_1 to S_{12} here. In addition, we use a set of test cases $T = \{T_1, \ldots, T_{53}\}$ from the set of regression tests used at Wincor Nixdorf. We created a method-level trace for a manual execution of each of these test cases using S_1. For test cases T_1 to T_3, we also created multiple traces for the three snapshots S_1, S_5, and S_{12}.

As a measure for RQ1, we compute which percentage of the test case results for S_1 is still valid after the modification that produced the next snapshot. For this, we check for each snapshot and test case, whether any of the methods that were changed in the snapshot (compared to the previous one) are contained in the trace. Any such change can potentially change the outcome of the corresponding test.

To answer RQ2, we compute the expected number of invalidated test cases, if i arbitrary methods are changed. For this, we assume the likelihood of a method change to be independent of other methods and distributed equally. We calculate the expected number of affected test cases using a random sample of 10,000 method sets of size i.

For RQ3, we execute each of the first 3 test cases in T six times, each time creating a trace for it. Each execution is consistent with the test case description, but varies non-specified properties, such as the exact steps used to navigate in the UI. For each of these test cases, we determine the number of methods executed by *all* of them (size of set A) and by only *some* of them (size of set B) (*cf.*, Figure 1). The ratio $|A|/(|A|+|B|)$ captures the probability that a method, which is *sometimes* covered by a test case, is *always* covered. It thus serves as an indicator for test case determinism.

For RQ4, we execute each of the three test cases that were run on S_1 in RQ3 on the snapshots S_5 and S_{12} to produce three more traces for each test case and snapshot combination. For each of these combinations, we then calculate the
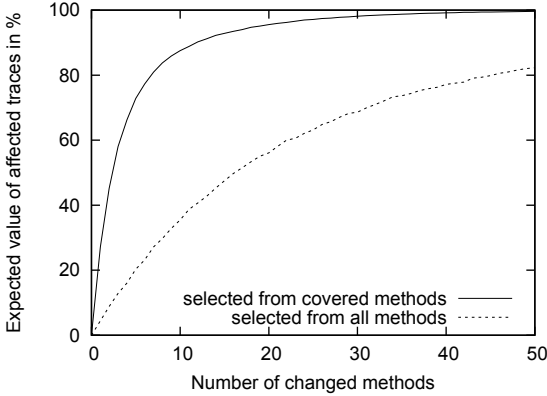
Figure 2: Expected number of invalidated tests by number of changed methods

Table I: Sizes of the sets determined in RQ3

test case	$\lvert A\rvert$	$\lvert B\rvert$	$\lvert A\rvert/(\lvert A\rvert+\lvert B\rvert)$
T_1	3692	325	92%
T_2	4211	314	93%
T_3	1605	3295	33%

Table II: Number of elements staying in the sets A resp. B during system evolution

	$S_1 \to S_5$		$S_1 \to S_{12}$	
	$A \to A$	$B \to B$	$A \to A$	$B \to B$
T_1	99.1%	63.4%	99.0%	61.5%
T_2	99.6%	70.4%	98.5%	65.9%
T_3	100%	10.0%	99.9%	29.2%

sets A and B as in RQ3. To measure the stability of both sets, we compute the number of methods that are still in these sets for the traces taken from later snapshots.

D. Results and Discussion

For RQ1, we found that only the changes performed in S_6, S_8, and S_9 affected methods from any of the test traces. The methods modified between the other snapshots are not covered by any of our 53 tests. The changes from S_5 to S_6 modified 41 methods (from which 13 were covered by the tests); to S_8, 12 methods were modified (4 covered); and to S_9, 40 methods were modified (only 2 covered). From the perspective of SRT, both the deltas to S_6 and S_8 invalidate the results of 33 of our 53 test cases (about 62%), while the changes for S_9 only affect 3 of the tests (6%).

A plot of our data for RQ2 is shown in Figure 2. The expected values are given for both the case of picking methods only from those covered by our traces, and for picking arbitrarily from *all* methods. To invalidate more than 50% of the test results, only 3 *covered* methods have to be changed. This is consistent with the results from the history analysis, where changing 4 covered methods invalidated 62% of the tests. If not only covered methods are selected, the 50% mark is reached at 17 changed methods.

The results of RQ1 and RQ2 demonstrate that we cannot expect SRT approaches to determine a set of regression tests that is much smaller than the set of all regression tests, if modifications are not limited to a very small number of methods. If, e.g., only 10% of all test cases can be executed within the time window available to test a repair release, we cannot expect SRT approaches to reliably produce a sufficiently small set of tests: already for 3 or more changes to *covered* methods, we must expect to have 50% or more of the test cases in the selected set.

The results for RQ3 are summarized in Table I. For two of the tests, more than 90% of the methods executed during any test execution are found in all of them. This also means that

by looking at only one of the execution traces, we potentially miss up to 10% of methods that might be executed. However, the results for T_3 indicate that the stable fraction can be significantly smaller than 90%.

Table II shows our results for RQ4. Nearly every method that was in all traces (set A) for a test case taken on S_1 is also in all its traces for both later snapshots. Stability for the set B is much weaker, which confirms the results of RQ3.

The results of RQ3 show that under-specification of manual tests substantially impacts trace stability. This has two implications for SRT. First, the fundamental assumption underlying safe SRT approaches—that the execution path of the test case through the software is controlled [6]—is thus violated. This substantially reduces the conclusiveness of the results of safe SRT for manual tests. Second, we cannot be sure that a test case that SRT determines as affected by a modification really traverses the modified code—it might be part of the methods it does not always visit. This suggests that, after regression test selection, testers should trace the execution of the regression tests to make sure that the modifications have really been covered. If not, additional test cases may need to be executed.

E. Threats to Validity

Our traces are collected on the level of methods. More fine grained tracing, e.g. on the level of statements, could lead to less test cases being selected. However, statement level tracing has a higher performance impact and can hinder manual testing. Moreover, we expect the gain of more fine grained tracing to be small, as [1] shows tracing below the method/function level only in few cases lead to significantly better results. On the other hand, we did not use a safe SRT technique—we, e.g., ignored dependencies implied by inheritance or the database. Using a safe SRT technique would increase the number of selected test cases.

For RQ2, we assumed an equal distribution of change probability over all methods. In practice, the probability will follow a different distribution, which, however, is not known *a priori*. Additionally, we selected methods for a change set independently from each other, which is not faithful,

System & Modifications

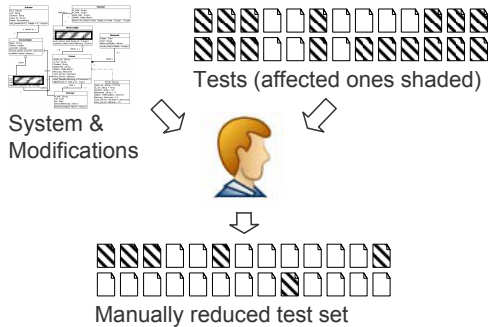Tests (affected ones shaded)

Manually reduced test set

Figure 3: Trace-guided test case selection

as we expect changed methods to be either technically or functionally related. Thus, the numbers reported might be overly pessimistic if we count the number of changed methods. But they can also be interpreted as a lower bound when we pick not individual methods but rather entire changes. For example, 3 changes that affect covered methods are expected to invalidate more than 50% of the tests.

Finally, the study is limited to a single system written in Java. Further research is required to understand how it transfers to other systems or other programming languages.

IV. TRACE-GUIDED TEST CASE SELECTION

If a repaired release must be delivered rapidly, the expected reductions of SRT are too small. How can we further reduce the test set size?

We qualitatively investigated this question on several pairs of consecutive system versions. The later version was derived from the earlier version through a modification that inadvertently broke a regression test. On these pairs, we made two observations. First, when inspecting code changes between the versions, the software engineers could make good guesses, which of the affected test cases were especially likely to uncover introduced bugs. For example, if a modification to ATM code affected currency handling, test cases that explicitly deal with different currencies were preferred. This suggests that knowledge about the modifications can be used to reduce the set of affected test cases.

Second, when we ran an SRT tool, it suggested test cases that, at first glance, would not have been considered by the engineers, since the engineers where unaware that they also exercised the modified code. An example are ATM update jobs that (among other things) update currency exchange rates. This suggests that SRT tools can indicate hidden system dependencies to the tester.

Based on these observations, we suggest a semi-automated process (depicted in Figure 3). In the first step, an SRT tool determines the system modifications and infers the test cases that are likely to exercise the modified code—as RQ4 shows, test cases that always covered the modified methods in the past are good candidates. Second, the software engineers inspect the modifications and decide, for each test case, if

its execution is likely to uncover a bug that this modification could have introduced, and that is unlikely to be uncovered by the test cases already contained in the regression test set.

This approach allows engineers to make conscious decisions about which test cases to include. Entirely manual regression test selection, as is currently done, faces higher risks of omissions due to hidden dependencies that are unknown to the testers. However, future work is required to show that trace-guided test case selection misses less error-uncovering test cases than entirely manual test selection.

V. CONCLUSION

We have presented an industrial case study on the applicability of existing SRT approaches to manual system tests. It demonstrates that, except for very small modifications, the number of test cases selected by SRT approaches must be expected to exceed the test effort available to retest a repair release. Furthermore, the under-specification of manual tests—which cannot be completely avoided in practice—causes unstable traces that reduce conclusiveness of results of safe SRT approaches. To compensate these obstacles, we suggest two strategies. First, to employ a semi-automated approach that combines SRT output with the knowledge and experience of test engineers. Second, to trace the execution of test cases during regression testing to assure that the modified code fragments have been covered.

REFERENCES

[1] J. Bible, G. Rothermel, and D. Rosenblum. A comparative study of coarse-and fine-grained safe regression test-selection techniques. *ACM TOSEM*, 2001.

[2] V. Channakeshava, V. Shanbhag, A. Panigrahi, R. Sisodia, and S. Lakshmanan. Safe subset-regression test selection for managed code. In *Proc. of ISEC'08*, 2008.

[3] Y. Chen, D. Rosenblum, and K. Vo. TestTube: A system for selective regression testing. In *Proc. of ICSE '94*, 1994.

[4] T. Graves, M. Harrold, J. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM TOSEM*, 2001.

[5] M. Harrold, J. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. Spoon, and A. Gujarathi. Regression test selection for Java software. *ACM SIGPLAN Notices*, 2001.

[6] G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE TSE*, 1996.

[7] G. Rothermel and M. Harrold. A safe, efficient regression test selection technique. *ACM TOSEM*, 1997.

[8] D. Willmor and S. Embury. A safe regression test selection technique for database-driven applications. 2005.

[9] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization: a survey. *Softw. Test. Verif. Reliab.*, 2007.