

Retesting Software During Development and Maintenance

Mary Jean Harrold and Alessandro Orso

*College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
{harrold/orso}@cc.gatech.edu*

Abstract

As most software continually evolves and changes during development and maintenance, it is necessary to test new and modified parts and retest existing parts that might have been affected by the changes. This activity is called regression testing and can account for a large percentage of the overall cost of software development. For this reason, much research has been (and is still being) performed on regression testing. This paper presents an overview of the major issues involved in software regression testing, an analysis of the state of the research and the state of the practice in regression testing in both academia and industry, and a discussion of the main open challenges for regression testing.

1 Introduction

Both during development and after deployment, software is modified for several reasons, including bug fixing, functionality enhancement, and adaptation to changes in the software's operating environment. One of the most expensive activities that occurs as the software is modified and enhanced is the (re)testing of the software after it has changed—this process is known as *regression testing*. Reports estimate that regression testing consumes as much as 80% of the overall testing budget¹ and can consume up to 50% of the cost of software maintenance [7, 36].

The frequency and time at which regression testing is performed depends on the context. For traditional development processes, regression testing can be performed after changes are made to the software, such as after nightly or regular builds, or before a new version of the software is released. For agile processes, the software (or parts thereof) can be retested every time the software is saved and compiled. For other contexts, regression testing can be applied before patches, such as security patches, are released. Regardless of the context, the goals of regression testing are the same: to improve confidence that the changes behave as intended and that they have not adversely affected unchanged parts of the software.

¹Los Altos Workshop on Testing (LAWST), held February 1-2, 1997 and reported in Reference [31].

Because regression testing is important, but expensive, much research has been performed, both in industry and in academia, on making it more effective and efficient. Researchers have developed techniques for addressing a number of issues related to regression testing, such as techniques for creating regression test suites, reusing these suites for testing subsequent versions of the software, identifying criteria for testing the modified versions that reduce risks associated with the changes, and attempting to repair test cases that have become obsolete. This research has also produced many tools and systems that have been used for empirical studies that investigate the effectiveness, scalability, and practicality of the techniques. Despite the extensive research in regression testing, there are still many challenges for making the techniques developed by researchers and practitioners usable in practice and for getting them transferred to industry.

The goal of this paper is to present a survey of the main regression-testing techniques proposed to date, so as to better understand the state of the research and the state of the practice in regression testing, along with a discussion of the current trends in both academia and industry. To this end, we first present the major issues involved in software regression testing (Section 2). We then overview the current state of the research and the state of the practice in regression testing (Sections 3 and 4, respectively). Finally, we discuss what we perceive to be the important open challenges for regression testing in the next decade (Section 5). Due to space limitations, we present a representative bibliography of regression-testing research.

To understand the impact of existing regression-testing techniques and the important open challenges for this area, and to ensure that our report represents the collective assessment of the regression-testing community, we performed an informal survey among regression-testing researchers and practitioners. We asked them (1) what they believed were the most significant contributions to testing research, (2) whether they knew of any techniques that were being used in industry, and (3) what they thought were the biggest challenges for regression testing in the future. We incorporated

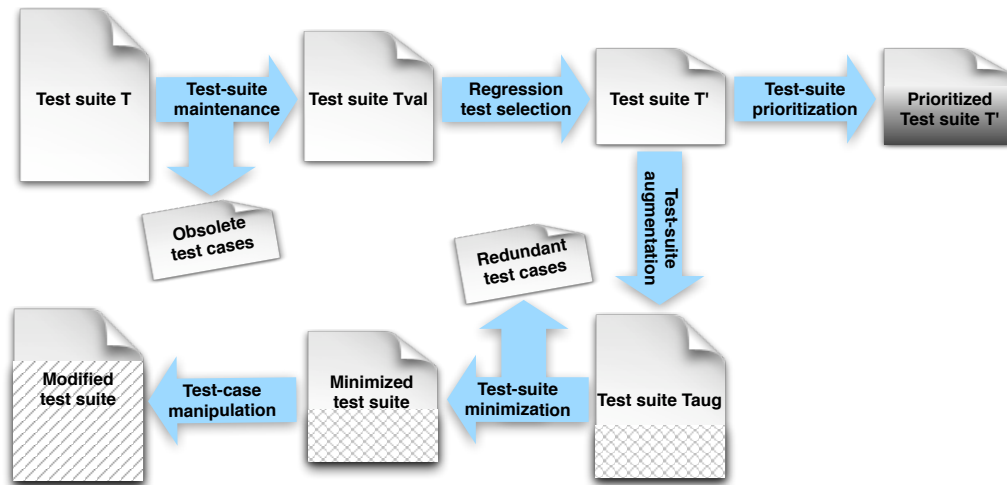


Figure 1. Intuitive view of regression testing activities.

their responses into our report and, in Section 6, we acknowledge those who contributed.

Before presenting the core of the paper, we note that there are many other techniques that are important for assessing software quality but are only indirectly related to regression testing. These techniques include: program analyses and representations used for regression-testing activities; efficient instrumentation techniques for profiling and tracing program executions and gathering data for use in regression testing; and debugging and fault-localization techniques that can be applied to find and eliminate faults uncovered during (regression) testing. Because of space constraints, we do not discuss these techniques in the paper.

2 Regression Testing

As we stated in the Introduction, regression testing is the activity of testing software after it has been modified to gain confidence that (1) newly added and changed code behaves correctly and (2) unmodified code does not misbehave because of the modifications. There are a number of issues involved in regression testing, both conceptual and engineering-related. We illustrate the most important of these issues, depicted in an intuitive way in Figure 1, by referring to a typical regression-testing scenario where there is an original version of a program, P , a modified version of P , P' , and a test suite for P , T .

An important issue is the identification, and possible fixing, of test cases in T for use on P' —we call this *test-suite maintenance*. One problem is the identification of obsolete test cases, that is, test cases that cannot be rerun on P' because of the changes from P to P' . For example, changes in an interface can prevent a test case from running or changes in functionality can change the outcome of a test case.

After the subset of T that is not obsolete, T_{val} , has been identified, another issue is the selection of the test cases in

T_{val} to rerun on P' —we call this activity *regression-test selection*. Although rerunning all test cases in T_{val} is a possibility, it can result in a considerable waste of testing resources because unnecessary test cases are rerun. An analysis of the changes between P and P' can help select a subset of test cases, T' , to be rerun on P' .

If the testing resources available are insufficient to rerun all test cases in T' (or T_{val} , if no selection is performed), it is useful to order the test cases being used according to some criteria—we call this activity *test-case prioritization* or *test-suite prioritization*. Test cases can be ordered using criteria such as coverage or estimated fault detection ability.

Even if all the test cases in T_{val} can be run, this set of tests may not adequately exercise P' 's new or modified functionality. In these cases, T_{val} must be assessed in an efficient way for its effectiveness in testing the changes from P to P' , and new test cases must be created and added to T_{val} —we call this activity *test-suite augmentation*. Test-suite augmentation produces an updated test suite, T_{aug} , that is adequate for P' according to some criterion.

Adding test cases to a test suite can improve its effectiveness, but it also results in the continuous growth of the test suite size. To address this problem, it might be desirable to identify and remove from T_{aug} test cases that are redundant, in that they exercise the same behaviors in the program—we call this activity *test-suite reduction* or *test-suite minimization*.

To facilitate this regression-testing process, there are a number of additional related issues to consider. One issue is the initial creation of the test suite T for use in regression testing. Test cases that are *fine-grained*, in that they target particular behaviors, improve the effectiveness of the selection, prioritization, and reduction. Test cases that are created by capturing program inputs and outputs provide *oracles* that enable automatic checking of P' 's output. An-

other issue is the availability of infrastructure, frameworks, or scripting tools for performing the activities previously discussed. These techniques provide ways of automatically executing regression-test suites and checking the outcome of the executions after the program has changed.

3 State of the Research

We now present the current state of the research in regression testing. We organize the presentation around the issues discussed in Section 2 and according to the same scenario introduced in that section: P' is the modified version of a program P , and T is a set of test cases for P .

One of the main issues in the context of test-suite maintenance is the automatic identification of obsolete test cases. The simplest case occurs when test cases become obsolete because they access an interface that has changed from P to P' . These test cases can be identified in a fully automated way and reported to the testers. In fact, for all test cases that access a module of the software directly, such as unit or subsystem test cases, a traditional compiler can identify which ones are invoking a non-existent interface. More problematic is the case of test cases that are obsolete because the functionality of the software changed from P to P' and thus, invalidated their oracle. We are not aware of any significant research that addresses this problem.

One area of regression-testing for which there has been significant research is regression-test selection (RTS), which attempts to focus the retesting around the changes from P to P' , and make use of T for retesting P' . RTS techniques use information about the execution of P with T , along with changes from P to P' , to select a subset of T to use for testing P' . The classification of test cases in T into categories of retestable, reusable, and obsolete [36] provided insights that led to many techniques for regression-test selection. One class of techniques creates models of P and P' and identifies the set of entities in the models that represent changes from P to P' . These techniques then use the identified entities to select a subset of T , T' , that exercises those entities in P and is, thus, used to test P' . Some techniques use source-code representations, with entities such as statements [70], branches [2, 60], control and data dependencies [6, 10, 20, 27], functions and non-executable components [15], and program paths [5, 8] or traces [51]. Other techniques use representations based on software artifacts other than the source code, such as UML diagrams [12, 13], architectural models [46], and program spectra [78]. Another class of techniques uses dependencies related to changes to determine the region of the program associated with the change, called a firewall, that must be re-tested [37, 75] (Section 4 provides more information about this technique).

An important aspect of an RTS technique is its safety. *Safe* RTS techniques ensure that the test suite selected, T' ,

contains all test cases in T that execute code that was modified from P to P' .² Many RTS techniques have been shown to be safe (e.g., [5, 15, 60, 70]). Other RTS techniques are not safe, in that they might omit test cases from T' that execute modifications from P to P' . However, these techniques often achieve test selection that is close to that achieved by safe techniques, and they may provide information that is useful for other regression-testing activities.

Researchers have modified and tailored RTS techniques for systems written in different types of languages, such as object-oriented languages [1, 23, 24, 34, 48, 50, 53] and aspect-oriented languages [79], and for different types of programs, such as graphical user interfaces (GUI) [3, 43, 45, 73] and component-based systems [80, 81]. Researchers have also constructed infrastructure on which to perform empirical studies [16], performed studies showing that these RTS techniques can be effective in reducing the number of test cases selected to be rerun (i.e., T') [9, 19, 26, 55, 61], and created and used cost models to compare these techniques [38, 58].

Considerable research has also been performed in the area of test-case prioritization (TCP). TCP techniques use information about P , T , and the changes from P to P' to order the test cases in T to meet some testing goal. Like RTS techniques, TCP techniques typically create a model of P and associate test cases in T with entities in P . Then, when P' becomes available, these techniques use coverage information about T on P to order test cases in T for rerunning on P' . TCP techniques leverage various kinds of information about P , P' , and T , such as requirements [67], source code [29, 63, 68], models of the software [32, 33], time to run test cases [72], and interactions among events [14]. Researchers also use techniques such as evolutionary algorithms to determine the test-case ordering [40]. Studies have shown that TCP can be quite effective in improving the regression-testing process [18, 63, 77]. Even if RTS is used to select T' (i.e., a subset of T), TCP techniques can be used to order test cases in T' so as to run the most important test cases first. There continues to be much research on test-case prioritization using different criteria and techniques for ordering the test cases.

Test-suite augmentation (TSA) is an area that is closely related to the selective-retest techniques just presented, but, to date, has received less attention than those techniques. TSA techniques use information about the changes from P to P' to identify criteria for retesting the changes. These criteria can then be used to (1) assess the test suite used to test P' , which consists of T' and any new test cases developed, and (2) guide the selection of new test cases that are needed to adequately test P' with respect to the changes. Like selective-retest techniques, TSA techniques create models

²Rothermel and Harrold defined *safe* regression-test selection and showed the difficulty of selecting such a subset [60].

of P and P' , assess the differences between them, and use these differences to compute what we call *change-test requirements* according to some criterion. (A test suite that satisfies the computed *change-test requirements* is said to be *change adequate*.) Many TSA techniques use differences in entities between P and P' as the criterion. These entities include data-flow [41, 49, 69], control-flow [35, 60], and both data-flow and control-flow [6, 20, 59]. More recent TSA techniques use incremental, localized symbolic execution related to the changes as a way to compute and characterize differential behavior in P and P' [4, 66]. These techniques then use this information to create change-test requirements that account for differences in both the entities and the states of P and P' . Although there have been empirical studies on the effectiveness of TSA techniques, these studies are limited, in that they have been performed on small subjects that may not be representative. Thus, additional studies are needed to demonstrate the effectiveness and efficiency of these techniques.

Researchers have extensively investigated the problem of test-suite reduction (TSR), which consists of (1) identifying test cases in T that are redundant according to some criterion and (2) removing them from T . For example, researchers have proposed TSR techniques that operate directly on P 's source code [22, 29] and leverage coverage information to identify redundancy. Although TSR techniques can significantly reduce the size of a test suite, this reduction often comes with a reduction in the fault-detection ability of the test suite. Several researchers have investigated this limitation of TSR with different results [42, 62, 77]. Note that a possible solution for this issue is to keep redundant test cases for later use (e.g., when new changes are performed) instead of actually removing them.

We now discuss several techniques that do not directly target one of the issues that we discussed in Section 2, but can support one or more regression-testing activities and have been investigated by researchers and practitioners.

A family of such techniques are capture-and-replay (CR) techniques, which allow for recording and later reproducing executions or parts thereof. Some CR techniques operate at the GUI level; they record user interactions with the GUI and produce test scripts that facilitate reproducing such interactions and can be used as automated test cases. Other CR techniques gather inputs and outputs as test cases are executed [39], so that the execution of the test cases can be automated and the output checked automatically. More recent CR techniques have investigated the use of CR technology to generate test cases at different levels of granularity (e.g., subsystem or unit) given a set of system test cases [17, 30, 47, 64].

Researchers have also proposed methods to fix regression-test suites as the software evolves [43–45]. These techniques have not yet been evaluated on large sys-

tems to assess their effectiveness and practicality. Another set of supporting techniques are those for predicting the effectiveness of regression-test selection, based on the program and its coverage [57]. Empirical evaluations of these approaches, however, has shown that the prediction varies with the program, changes, and coverage [25]. Finally, in terms of supporting infrastructure, researchers have proposed continuous testing, which runs the regression test suite automatically from within an integrated development environment every time the program is saved and compiled [65]. This approach can help regression testing by reducing the time between introduction and discovery of a fault.

4 State of the Practice

In this section, we present the current state of the practice in regression testing. One of the goals of this section is to assess which of the research techniques discussed in Section 3 have transitioned to practice and to what extent. Therefore, also in this case, we organize the presentation around the regression-testing scenario presented in Section 2.

In industry, test-suite maintenance is largely a manual process. After changes are made to the software, testers typically identify and remove obsolete test cases, fix test scripts, and repair test cases without automated support. Our interactions with industrial testers confirm that this is an expensive and time consuming task.

A few organizations do perform automated RTS. Motorola, for instance, is reported to have such a process, based on RTS techniques developed in-house. In most organizations, however, RTS is largely a manual process. One common method for selecting test cases uses the traceability between requirements and test cases that is often maintained; when changes are made to requirements, the test cases associated with changed or new requirements are selected for rerun. One issue with this approach is that it is not safe, in that it may miss test cases that are not associated with changed or new requirements but that are affected by the change. To address this problem, some techniques use additional information, such as changes in interfaces or data, historical test-case effectiveness, timing data on the last time a test case was run, or domain knowledge. Another common method for selecting test cases, even when traceability information is maintained, is to rerun some core set of test cases, a randomly selected subset of regression test suite T , or a subset of T selected based on the knowledge of the testers. Finally, many organizations simply rerun T completely (i.e., they apply a straightforward select-all RTS approach).

One notable example of RTS (and TSA) use is the Testing Firewall Technology (TFW), which has been used at ABB.³ Given a change in the software, the TFW approach

³ABB is an international corporation that provides industrial real-time control systems for diverse industry applications using configuration-based software; see <http://www.abb.com/>.

lets testers both reuse existing test cases and design, in a simple and direct way, new necessary test cases. After two years of work with ABB, researchers had applied TFW for regression testing on 18 projects (16 procedural, two object-oriented). The technology resulted in the rerunning of 40% fewer test cases than the previously-used testing process. Most importantly, those test cases found all of the faults found by the previously-used test suite, and uncovered many faults previously found only by customers after deployment. ABB is now reported to be using the TFW approach at its plants in North America and worldwide. (For details of the study, see References [74,76].) Given this success, researchers have extended the TFW approach to work for configurations, and have developed the Configuration Firewall, which is now also being applied at ABB [54, 80].

As the above discussion shows, most of the advanced RTS techniques described in Section 3 are not commonly incorporated into the regression-testing process. However, organizations are using or beginning to use TCP techniques. Microsoft, for instance, is reported to use TCP based on a number of criteria [68]. In his keynote address at OOPSLA 2002,⁴ Bill Gates discussed how Microsoft was bringing research to practice by using a TCP system that “ranks the application’s given set of tests, based on changes made to the program. [Using this system,] development teams can track in-process testing, prevent defects from entering the system through early detection and thus improve product security and reliability, and shorten the repair process.”

Several organizations are reported to be exploring the application of TCP techniques that rely on search-based optimization within their regression-testing process. These techniques allow testers to search for an optimal ordering of the test cases that considers, at once, many characteristics of the test cases, such as cost, running time, criticality, and complexity of the set-up [21].

We have discovered, in discussions with testing organizations, that they sometimes use or have used TSR techniques to “thin” their regression test suites. However, these organizations also report that the test cases removed by these TSR techniques are often found to be appropriate or needed for testing later versions of the code. This finding seems to indicate that TSR can be an ineffective method for managing test suites. Instead of reducing the test suite, an alternative is to use TCP to order the test cases. In this way, the most important test cases, according to some criteria, can be run first, while the other test cases are kept for consideration for future changes.

An area where the state of the practice is somewhat advanced is that of test environments and infrastructure. In terms of environments, there are many widely used platforms on which developers and testers can create test cases

⁴<http://www.microsoft.com/presspass/press/2002/Nov02/11-08OOPSLAPR.msp>

and automatically run them. IBM, for instance, started the Eclipse Project⁵ in 2001. Since then, Eclipse has been supported and extended by a consortium of software vendors, universities, and research institutions. Many tools have been created that integrate as plug-ins into the Eclipse framework, including tools that provide assistance in creating and running test cases. For example, the Test & Performance Tools Platform⁶ helps developers build testing tools by extending the platform, which includes editors, test case execution facilities, and reporting functionality.

An example of successful infrastructure is JUnit,⁷ a widely used instance of the xUnit architecture that was created to ease regression testing of Java software. JUnit, which is also supported by Eclipse through a plug-in, helps users write test cases that can then be automatically rerun each time the software is modified. Another example of regression testing infrastructure that has been widely adopted by industry are GUI-based capture-replay tools.

5 Challenges

Section 3 shows that there has been significant research in the past few decades on improving regression testing. However, as we discussed in Section 4, only a handful of the techniques and tools developed by researchers and practitioners are currently used in practice. Therefore, a first set of challenges for regression testing involves transitioning well-assessed research results into practice. A second set of challenges involves technical and conceptual issues that have not yet been investigated or that have only partially been addressed by research. In this section, we discuss some of the major practical and technical issues that we perceive as relevant for regression testing in the next decade.⁸

5.1 Transition to Practice

Based on our experience and the feedback from other researchers, we believe that the limited technology transfer for regression testing research is due to a number of factors.

One important factor is the need for additional validation of existing regression-testing techniques. Although there are a number of papers on evaluation of regression-testing techniques using controlled experiments (e.g., [9, 18, 19, 56, 62, 71, 74]), only a few of these empirical studies have been performed on real-world, large-scale systems or shown to be useful in practice. Moreover, most of these studies (necessarily) focus on a small set of programs, and it is not obvious that their findings will generalize to programs other than the ones considered. Thus, there is a need for additional, large industrial studies that may provide real users

⁵<http://www.eclipse.org/>

⁶<http://www.eclipse.org/tptp/>

⁷<http://www.junit.org>

⁸Many of these challenges were suggested by the researchers and practitioners who provided feedback to our informal survey.

in industry better confidence in the practical usefulness of existing techniques.

Another, mainly practical, factor is that little effort has been spent on investigating how to integrate regression-testing techniques into existing development, testing, and debugging processes. Some researchers have taken initial steps in this direction, by integrating their approaches into well-known and widely used integrated development environments (e.g., [52, 53, 65]), but most other work consists of stand-alone applications that may be difficult to adopt within an organization. For example, although regression-test selection has been shown empirically to be effective in many studies, there are organizations where it is not applied simply because the required coverage information is not collected during testing, and collecting it would require a change in the existing testing process.

Moreover, despite the number of techniques, tools, and infrastructure developed and (often) freely available, the issues of how differences in organizational testing and engineering processes affect tradeoffs in approaches are unknown and need to be studied. In this area, as in regression-test selection, there is a need to perform studies in industrial settings to attempt to assess the hidden costs of using these research regression-testing techniques in practice and within real, representative software development processes. These studies are also likely to help clarify which regression-testing methods are appropriate for which types of systems, which should, in turn, lead to recommendations that the industry can use.

Finally, to help facilitate the transition of research techniques to current practice, researchers should, in collaboration with industry, improve their tools so that they are as automated as possible, user friendly, and more robust than simple proof-of-concept prototypes. For instance, integrating regression-testing techniques such as regression-test selection or test-case prioritization into commonly used environments (e.g., Eclipse⁵ and Visual Studio⁹) or testing frameworks (e.g., JUnit⁷) will help increase the level of acceptance of these techniques in industry.

5.2 Open Research Issues

5.2.1 Test Suite Augmentation

Whereas topics such as regression-test selection, test-suite reduction, and test-case prioritization have been extensively studied and evaluated, the problem of test-suite augmentation has been mostly neglected by previous research. Although being able to increase the efficiency of the regression-testing process is of paramount importance, assessing and improving its effectiveness is also fundamental. In Section 3, we discussed the two main components of test suite augmentation: (1) assessing existing test suites

⁹<http://msdn.microsoft.com/en-us/vstudio/default.aspx>

and generating requirements, and (2) supporting test-case generation. Here, we discuss the open research issues for these two components.

To date, there are only a few techniques that aim to assess whether an existing test suite is still adequate for a new version of a program (we discussed these techniques in Section 3). Unfortunately, most of these techniques are limited in the kind of requirements they generate because they concentrate only on some aspects of the changes (e.g., their effect on direct data- and control-flow relations), are too expensive to scale to realistic software, or may generate too many spurious requirements. Some techniques are more promising, but they have been evaluated only on small examples and programs. More research is needed to (1) develop practical techniques for assessing the adequacy of existing test suites with respect to a set of changes and computing corresponding test-suite augmentation requirements, and (2) evaluate such techniques in realistic settings.

The issue of supporting test-case generation is strongly related to the previous one: techniques that generate requirements for change-adequate regression test suites are useful and can be used to guide testers in their activity of developing new test cases during regression testing. However, the generation of test cases that can satisfy these requirements can be extremely difficult and tedious. To the best of our knowledge, no research has specifically addressed this important issue. Although the issue is similar to the traditional test-case generation problem, there are some specific aspects that distinguish test-case generation during regression testing from its traditional counterpart. In particular, during regression testing (and maintenance in general), there is an opportunity to focus on differential behavior between two versions of the software and define specialized version of (otherwise too expensive) test-case generation techniques.

For instance, the approaches to characterizing differential behavior based on partial symbolic execution discussed in Section 3 could be extended to compute test cases that exercise such differential behavior. Another possibility, also mostly unexplored, would be to leverage existing test cases that exercise behaviors close to the one(s) of interest—according to some requirement r —to generate new, slightly different test cases that satisfy r . In general, the generation of change-adequate test cases is a broad area that has been only superficially investigated and is likely to become increasingly relevant in the future.

5.2.2 Test Case Maintenance and Manipulation

As we discussed in Section 3, we are not aware of any research that targets the automatic identification of obsolete test cases in the non-trivial cases of functionality changes that invalidate oracles. In general, it is not possible to automatically identify this kind of situation. Nevertheless, it may be possible to devise techniques based on heuristics

that can be successful in at least some cases. More generally, behavioral analysis performed on old and new versions of the software, such as those mentioned in Section 3, may help identify oracles that were invalidated by the changes in the software. In some cases, behavioral analysis may even be able to help fix the oracles affected by the changes in an automated or semi-automated way.

In the context of test-suite maintenance, there is also a need for additional research on how to leverage impact-analysis techniques during evolution. Previous research has shown how impact analysis can help identify those changes that cause particular test cases to fail [53], which can help testers not only to debug regression problems, but also to identify test cases that may be obsolete because of the changes. Additional research could attempt to identify, by analyzing the changes and performing a more sophisticated impact analysis, which behavioral differences are actually expected (and thus, require the affected test cases to be adapted) and which ones may be indicating regression errors instead.

A relatively recent trend in test-case maintenance research is the interest in techniques for manipulating existing test cases and transforming them into new, or simply different, test cases. (Because these techniques typically add test cases to an existing test suite, they can also be seen as a special instance of test-suite augmentation.) The goal of these techniques is to amortize the resources invested in developing test cases by reusing and generally leveraging existing test cases in as many ways as possible. Examples are techniques to derive test cases at different levels of granularity from system test cases, which may help speed up regression testing and result in more focused test cases (e.g., [17, 30, 64]). Although interesting, these techniques are just a first step. There is considerable room for improvement in the direction of manipulating test cases to reuse them or make them more effective in a specific context.

5.2.3 Leveraging Field Data and Resources

Software, especially mass-market software, is typically developed and distributed to a large number of users around the world. Because of its high dynamism, configurability, and portability, today's software may behave very differently for different users. As examples, consider Java software, which can run on completely different operating systems, or applications such as Tomcat, which have a huge configuration space. These different behaviors are often difficult to foresee and reproduce in-house (i.e., during development) but could be observed and analyzed on deployed software. For this reason, deployed software has the potential to generate a wealth of information that can give unprecedented insight to developers and be extremely useful for many development and maintenance tasks. Unfortu-

nately, in current practice, little of this information is actually collected and made available to developers.

There is a need for research that attempts to improve this situation by defining techniques that leverage information gathered from deployed applications for testing and maintenance. For instance, usage information from the field could be used to assess how a change may affect the users and how to test that change and the code affected by it. For another example, information about user executions could be leveraged to create regression test cases that are more representative of the way the software is (and will be) actually used. For yet another example, in the presence of a large user population, researchers could develop and investigate techniques that leverage part of the population to assess the behavior of a new release before its general release. This approach would be similar to traditional beta testing, but would exploit today's connectivity and ever increasing computing power to automatically perform partial update of the software and collect relevant information about the behavior of the software on the platforms that received the update.

5.2.4 Program Models for Regression Testing

There are two kinds of models that researchers should consider in the context of regression testing. The first set of models are those that can be used to represent test cases and related information (e.g., fault-detection capability, coverage, and time to run). The second set of models are traditional program models, such as UML models.

For the first set of models, one of the challenges is to build richer models than those used to date. Multi-objective techniques are becoming popular [11, 28, 40], and models that can support those techniques must take into account more and more complex information about test cases (e.g., oracle costs, subtle relationships between test cases, and constraints in terms of ordering or clustering of test cases).

For the second set of models, they contain a great deal of potentially useful information that is not readily available in the source code. Such information could be used to inform regression testing and better support techniques such as regression-test selection and test-case prioritization. Moreover, because effective and efficient test-generation techniques based on models are becoming available, there is an opportunity to leverage the automated traceability between design models and test cases [13].

Another challenge related to this context is how to maintain, in a (semi)automated way, the mapping between program models and programs during maintenance. The ability to maintain an automated mapping between these two levels may help to define code-based regression-testing methods and requirement-based regression-testing methods that can benefit from each other.

5.2.5 Regression Testing for Security

Every time a program is modified, new security vulnerabilities can be introduced as side effects. Also, every time a security hole is fixed and a patch is produced, it is important to gain sufficient confidence that the vulnerability was fixed and no other vulnerabilities were introduced. Security vulnerabilities are nothing other than faults in the code, and therefore, testing and checking for such vulnerabilities can be considered traditional regression testing. Nevertheless, there are several specific characteristics of regression testing for security that make it slightly different. First, security faults are a specific class of faults (e.g., buffer-overflow vulnerabilities), and it may be possible to devise special techniques to regression test for them. Second, when a security patch is produced, there is little time for testing it before it is released, so regression-test selection must be particularly aggressive.

5.2.6 Accommodating New Types Of Software Systems, Contexts, and Development Environments

Software is becoming increasingly parallel, partly because of the push for multi-core in the hardware world. Regression testing of parallel software is likely to become a significant challenge, with many open problems to be solved. At the same time, the parallelization of regression testing may also bring benefits and new opportunities, such as the use of parallelism to reduced testing cost.

Another shift that is occurring is from traditional desktop applications to increasingly complex web applications, which are distributed in nature and highly dynamic. Additionally, because they run remotely and, thus, do not need to be distributed in a traditional way, web applications tend to be updated more often than traditional applications. One challenge is, therefore, to define daily maintenance cycles that are reliable, secure, safe, and efficient. Finally, web applications integrate with other web applications or web services, which can happen dynamically. In this context, one challenge is how to check whether the use of a new service, for instance, will not introduce side effects or simply produce wrong results.

Analogous challenges are present in the context of component-based software. A software component's correct behavior in one environment is no guarantee that it will behave correctly in a new environment. Although this issue is not new, and there has been significant research on this topic, most of the resulting techniques are of limited practical usefulness; they tend to make unrealistic assumptions, such as that developers will write specifications, provide their test cases, and release software with no cyclic dependencies. Researchers need to define techniques that help developers and testers document how a component has been tested in other environments and decide how and how much to test it in the new environment.

Systems created by distributed development teams generate problems similar to those resulting from development of component-based systems; testers need to test these systems, which consist of individual subsystems, with their own regression-test suite. Researchers need to create techniques that will determine how these regression-test suites interact with each other and will specify how to integrate them for system-level regression testing.

5.2.7 Integrating Regression Testing Into Computer Science Education

The last challenge that we mention is not technical, but educational. Educators must provide sufficient coverage of regression testing in the undergraduate computing curriculum. Many computer science graduates have no knowledge of regression testing (or sometimes even testing), which can be quite limiting in a world where it is increasingly common to integrate and retest, rather than develop and test. Related to this point, another challenge is how to teach students to be integrators, rather than programmers. Many developers nowadays use some integrated development environment or framework (e.g., Eclipse⁵ and JBoss¹⁰) to integrate many different parts of the system. Educators rarely teach these issues in their curricula and seldom investigate or discuss research issues involving integration. Unfortunately, many educators and researchers tend to ignore, often simply due to inertia, that software development is completely different today from what it was 20 or even 10 years ago. This difference may be an additional reason why companies are reluctant to fund research in this area and college freshmen are reluctant to enroll in computer science degree programs.

6 Acknowledgements

We thank the many colleagues who contributed to this work: James H. Andrews, Lionel Briand, Mei-Hwa Chen, Sebastian Elbaum, Cem Kaner, Bogden Korel, David Kung, Mark Harman, Hareton K. N. Leung, Aditya P. Mathur, A. Jefferson Offutt, David S. Rosenblum, Gregg Rothermel, Antanas Rountev, Barbara G. Ryder, Frank Tip, John Van-Leeuwen, Lee J. White, and Tao Xie.

References

- [1] K. Abdullah and L. J. White. A firewall approach for the regression testing of object-oriented software. In *Proc. of Qual. Week*, May 1997.
- [2] H. Agrawal, J. R. Horgan, E. Krauser, and S. London. Incremental regression testing. In *Proc. Conf. on Softw. Maint.*, pages 1–10, Sept. 1993.
- [3] J. R. Andreas. Automated regression testing of graphical user interface based applications. In *Proc. Hawaii Int'l Conf. on Sys. Sci.*, page 101, Jan. 1991.

¹⁰<http://jboss.com>

- [4] T. Apiwatnapong, R. Santelices, P. K. Chittimalli, A. Orso, and M. J. Harrold. MaTRIX: Maintenance-Oriented Testing Requirements Identifier and Examiner. In *Proc. Wkshp. on Testing: Academic & Ind. Conf., Practice and Research Techniques*, pages 137–146, Aug. 2006.
- [5] T. Ball. On the limit of control flow analysis for regression test selection. In *Proc. Int'l Symp. on Softw. Testing and Analysis*, pages 134–142, Mar. 1998.
- [6] S. Bates and S. Horwitz. Incremental program testing using program dependence graphs. In *Proc. Symp. on the Prin. of Prog. Lang.*, pages 384–396, Jan. 1993.
- [7] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, New York, NY, 1990.
- [8] P. Benedusi, A. Cimitile, and U. D. Carlini. Post-maintenance testing based on path change analysis. In *Proc. Conf. on Softw. Maint.*, pages 352–361, Oct. 1998.
- [9] J. Bible, G. Rothermel, and D. Rosenblum. A comparative study of coarse- and fine-grained safe regression test selection. *ACM Trans. Softw. Eng. Meth.*, 10(2):149–183, Apr. 2001.
- [10] D. Binkley. Semantics guided regression test cost reduction. *IEEE Trans. Softw. Eng.*, 23(8):498–516, Aug. 1997.
- [11] J. Black, E. Melachrinoudis, and D. Kaeli. Bi-criteria models for all-uses test suite reduction. In *Proc. Int'l. Conf. on Softw. Eng.*, pages 106–115, May 2004.
- [12] L. C. Briand, Y. Labiche, and S. He. Automating regression test selection based on UML designs. *ACM Trans. Softw. Eng. Meth.*, to appear.
- [13] L. C. Briand, Y. Labiche, and L. O'Sullivan. Impact analysis and change management of UML models. In *Proc. Int'l Conf. on Softw. Maint.*, pages 256–265, Sept. 2003.
- [14] R. C. Bryce and A. M. Memon. Test suite prioritization by interaction coverage. In *Proc. Wkshp. on Domain-Specific Approaches to Softw. Test Automation*, Sept. 2007.
- [15] Y. F. Chen, D. S. Rosenblum, and K. P. Vo. TestTube: A system for selective regression testing. In *Proc. Int'l Conf. on Softw. Eng.*, pages 211–222, May 1994.
- [16] H. Do, S. Elbaum, and G. Rothermel. Infrastructure support for controlled experimentation with software testing and regression testing techniques. In *Proc. Int'l Symp. on Empirical Softw. Eng.*, pages 66–70, Aug. 2004.
- [17] S. Elbaum, H. N. Chin, M. Dwyer, and J. Dokulil. Carving differential unit test cases from system test cases. In *Proc. Symp. on the Foundations of Softw. Eng.*, pages 1253–264, Sept. 2006.
- [18] S. Elbaum, A. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.*, 28(2):159–182, Mar. 2002.
- [19] T. Graves, M. J. Harrold, J. M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. In *Proc. Int'l Conf. on Softw. Eng.*, pages 188–197, Apr. 1998.
- [20] R. Gupta, M. J. Harrold, and M. L. Soffa. Program slicing-based regression testing techniques. *Journal of Softw. Testing, Verif., and Rel.*, 6:83–111, 1996.
- [21] M. Harman and P. McMinn. A theoretical and empirical analysis of evolutionary testing and hill climbing for structural test data generation. In *Proc. Int'l Symp. on Softw. Testing and Analysis*, pages 73–83, July 2007.
- [22] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Trans on Softw. Eng. and Methodology*, 2(3):270–285, July 1993.
- [23] M. J. Harrold, J. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. Spoon, and A. Gujarathi. Regression test selection for Java software. In *Proc. Conf. on OO Prog., Sys., Lang., and Appl.*, pages 312–326, Oct. 2001.
- [24] M. J. Harrold, J. D. McGregor, and K. J. Fitzpatrick. Incremental testing of object-oriented class inheritance structures. In *Proc. Int'l Conf. on Softw. Eng.*, pages 68–80, May 1992.
- [25] M. J. Harrold, D. Rosenblum, G. Rothermel, and E. J. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE Trans. Softw. Eng.*, 27(3):248–263, Mar. 2001.
- [26] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical evaluation of program spectra. In *Proc. Wkshp. on Prog. Analysis for Softw. Tools and Eng.*, pages 83–90, June 1998.
- [27] J. Hartmann and D. J. Robson. Techniques for selective revalidation. *IEEE Software*, 16(1):31–38, Jan. 1990.
- [28] H.-Y. Hsu and A. Orso. MINTS: A general framework and tool for supporting test-suite minimization. Poster at the Int'l Symp. on Softw. Testing and Analysis, July 2008.
- [29] J. A. Jones and M. J. Harrold. Test-suite reduction and prioritization for modified condition/decision coverage. *IEEE Trans. Softw. Eng.*, 29(3):195–209, Mar. 2003.
- [30] S. Joshi and A. Orso. SCARPE: A Technique and Tool for Selective Record and Replay of Program Executions. In *Proc. Int'l Conf. on Softw. Maint.*, pages 234–243, Oct. 2007.
- [31] C. Kaner. Improving the maintainability of automated test suites. In *Proc. of Qual. Week 1997*, May 1997.
- [32] B. Korel, L. Tahat, and M. Harman. Test prioritization using system models. In *Proc. Intl Conf. Softw. Maint.*, pages 559–568, Sept. 2005.
- [33] B. Korel, L. Tahat, and B. Vaysburg. Model based regression test reduction using dependence analysis. In *Proc. Int'l Conf. on Soft. Maint.*, pages 214–223, Oct. 2002.
- [34] D. Kung, J. Gao, P. Hsia, Y. Toyoshima, and C. Chen. Class firewall, regression testing, and software maintenance of object-oriented systems. *Journal of OO Prog.*, pages 51–65, May 1995.
- [35] J. Laski and W. Szermer. Identification of program modifications and its applications in software maintenance. In *Proc. Conf. on Softw. Maint.*, pages 282–290, Nov. 1992.
- [36] H. K. N. Leung and L. J. White. Insights into regression testing. In *Proc. Conf. on Softw. Maint.*, pages 60–69, Oct. 1989.
- [37] H. K. N. Leung and L. J. White. A study of integration testing and software regression at the integration level. In *Proc. Conf. on Softw. Maint.*, pages 290–300, Nov. 1990.
- [38] H. K. N. Leung and L. J. White. A cost model to compare regression test strategies. In *Proc. Conf. on Softw. Maint.*, pages 201–208, Oct. 1991.
- [39] R. Lewis, D. W. Beck, and J. Hartmann. Assay - A tool to support regression testing. In *Proc. of the European Softw. Eng. Conf.*, pages 487–496, Sept. 1989.
- [40] Z. Li, M. Harman, and R. M. Hierons. Search algorithms for regression test case prioritization. *IEEE Trans. Softw. Eng.*, 33(4):225–237, Apr. 2007.
- [41] U. Linnenkugel and M. Mullerburg. Test data selection criteria for (software) integration testing. In *Proc. Int'l Conf. on Sys. Integration*, pages 709–717, Apr. 1990.
- [42] S. McMaster and A. M. Memon. Fault detection probability analysis for coverage-based test suite reduction. In *Proc. of the Int'l Conf. on Softw. Maint.*, Sept. 2007.

- [43] A. M. Memon. Using tasks to automate regression testing of GUIs. In *Proc. of the Int'l Conf. on Artificial Intelligence and Appl.*, Feb. 2004.
- [44] A. M. Memon. Automatically repairing event sequence-based GUI test suites for regression testing. *ACM Trans. Softw. Eng. Meth.*, 2008.
- [45] A. M. Memon and M. L. Soffa. Regression testing of GUIs. In *Proc. of the European Softw. Eng. Conf. and the Int'l Symp. on Foundations of Softw. Eng.*, pages 118–127, Sept. 2003.
- [46] H. Muccini, M. Dias, and D. J. Richardson. Software architecture-based regression testing. *Journal of Sys. and Softw.*, 79(10):1379–1396, Oct. 2006.
- [47] A. Orso and B. Kennedy. Selective Capture and Replay of Program Executions. In *Proc. of the Int'l Workshop on Dynamic Analysis*, pages 29–35, May 2005.
- [48] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. In *Proc. of the Int'l Symp. on the Foundations of Softw. Eng.*, pages 241–252, Nov. 2004.
- [49] T. J. Ostrand and E. J. Weyuker. Using dataflow analysis for regression testing. In *Pacific NW Softw. Qual. Conf.*, pages 233–247, Sept. 1988.
- [50] D. E. Perry and G. E. Kaiser. Adequate testing and object-oriented programming. *Journal of OO Prog.*, 2(1):13–19, Jan./Feb. 1990.
- [51] M. K. Ramanathan, A. Grama, and S. Jagannathan. Sieve: A Tool for Automatically Detecting Variations Across Program Versions. In *Proc. Int'l Conf on Automated Softw. Eng.*, pages 241–252, Sept. 2006.
- [52] X. Ren, B. Ryder, M. Stoerzer, and F. Tip. Chianti: A change impact analysis tool for Java programs. In *Proc. Int'l Conf. on Softw. Eng.*, pages 664–665, May 2005.
- [53] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley. Chainti: A tool for change impact analysis of Java programs. In *Proc. Conf. on OO Prog., Sys., Lang., and Appl.*, pages 432–448, Oct. 1996.
- [54] B. Robinson. *A firewall model for testing user-configurable software systems*. PhD thesis, Case Western Reserve University, May 2008.
- [55] D. Rosenblum and G. Rothermel. A comparative study of regression test selection techniques. In *Proc. Int'l Workshop on Empirical Studies of Softw. Maint.*, Oct. 1997.
- [56] D. Rosenblum and E. J. Weyuker. Lessons learned from a regression testing case study. *Empirical Softw. Eng. Journal*, 2(2):188–191, 1997.
- [57] D. Rosenblum and E. J. Weyuker. Using coverage information to predict the cost-effectiveness of regression testing strategies. *IEEE Trans. Softw. Eng.*, 23(3):146–156, Mar. 1997.
- [58] G. Rothermel, S. G. Elbaum, A. G. Malishevsky, P. Kallakuri, and X. Qiu. On test suite composition and cost-effective regression testing. *ACM Trans. Softw. Eng. Meth.*, 13(3):277–331, July 2004.
- [59] G. Rothermel and M. J. Harrold. Selecting tests and identifying test coverage requirements for modified software. In *Proc. Int'l Symp. on Softw. Testing and Analysis*, pages 169–184, Aug. 1994.
- [60] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Meth.*, 6(2):173–210, Apr. 1997.
- [61] G. Rothermel and M. J. Harrold. Empirical studies of a safe regression test selection technique. *IEEE Trans. Softw. Eng.*, 24(6):401–419, June 1998.
- [62] G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong. An empirical study of the effects of minimization on the fault detection capabilities of test suites. In *Proc. Int'l Conf. on Softw. Maint.*, pages 34–43, Nov. 1998.
- [63] G. Rothermel, R. Untch, C. Chu, and M. J. Harrold. Prioritizing test cases for regression testing. *IEEE Trans. Softw. Eng.*, 27(10):929–948, Oct. 2001.
- [64] D. Saff, S. Artzi, J. H. Perkins, and M. D. Ernst. Automatic test factoring for Java. In *Proc. Int'l Conf. on Automated Softw. Eng.*, pages 114–123, Nov. 2005.
- [65] D. Saff and M. Ernst. Continuous testing in Eclipse. In *Proc. Int'l Conf. on Softw. Eng.*, pages 668–669, May 2005.
- [66] R. Santelices, P. K. Chittimalli, T. Apiwattanapong, A. Orso, and M. J. Harrold. Test-suite augmentation for evolving software. In *Proc. Int'l Conf. on Automated Softw. Eng.*, Sept. 2008.
- [67] H. Srikanth, L. Williams, and J. Osborne. System test case prioritization of new and regression test cases. In *Proc. Int'l Symp. on Empirical Softw. Eng.*, Nov. 2005.
- [68] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In *Proc. Int'l Symp. on Softw. Testing and Analysis*, pages 97–106, 2002.
- [69] A. B. Taha, S. M. Thebaut, and S. S. Liu. An approach to software fault localization and revalidation based on incremental data flow analysis. In *Proc. Int'l Comp. Softw. and Appl. Conf.*, pages 527–534, Sept. 1989.
- [70] F. Vokolos and P. Frankl. Pythia: A regression test selection tool based on text differencing. In *Proc. Int'l Conf. on Rel., Qual., and Safety of Softw. Intensive Sys.*, May 1997.
- [71] F. Vokolos and P. Frankl. Empirical evaluation of the textual differencing regression testing techniques. In *Proc. Int'l Conf. on Softw. Maint.*, pages 44–53, Nov. 1998.
- [72] K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, and R. S. Roos. Time aware test suite prioritization. In *Proc. Int'l Symp. on Softw. Testing and Analysis*, pages 1–12, July 2006.
- [73] L. J. White, A. Almezen, and S. Sastry. Firewall regression testing of GUI sequences and their interactions. In *Proc. Conf. on Softw. Maint.*, pages 262–270, Sept. 2003.
- [74] L. J. White, K. Jabar, B. Robinson, and V. Rajlich. Extended firewall for regression testing: An experience report. *Journal of Softw. Maint. and Evolution: Research and Practice*, 2008.
- [75] L. J. White and H. K. N. Leung. A firewall concept for both control-flow and data-flow in regression integration testing. In *Proc. Conf. on Softw. Maint.*, pages 262–270, Nov. 1992.
- [76] L. J. White and B. Robinson. Industrial real-time regression testing analysis using firewalls. In *Proc. Int'l Conf. on Softw. Maint.*, pages 18–27, Sept. 2004.
- [77] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. Int'l Symp. on Softw. Rel. Eng.*, pages 230–238, Nov. 1997.
- [78] T. Xie and D. Notkin. Checking inside the black box: Regression testing by comparing value spectra. *IEEE Trans. Softw. Eng.*, 31(Oct.):869–883, 2005.
- [79] J. Zhao, T. Xie, and N. Li. Towards regression test selection for aspect-oriented programs. In *Proc. Wkshp. on Testing Aspect-Oriented Programs*, pages 21–26, July 2006.
- [80] J. Zheng, B. Robinson, L. Williams, and K. Smiley. Applying regression test selection for COTS-based applications. In *Proc. Int'l Conf. on Softw. Eng.*, pages 272–277, May 2006.
- [81] J. Ziegler, J. Grasso, and L. Burgermeister. An Ada based real-time closed-loop integration and regression test tool. In *Proc. Conf. on Softw. Maint.*, pages 81–90, Oct. 1989.