An Empirical Study of Regression Testing Techniques Incorporating Context and Lifetime Factors and Improved Cost-Benefit Models

Hyunsook Do, Gregg Rothermel
Department of Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, Nebraska 68588-0115, USA
{dohy, grother}@cse.unl.edu

ABSTRACT

Regression testing is an important but expensive activity, and a great deal of research on regression testing methodologies has been performed. In recent years, much of this research has emphasized empirical studies, including evaluations of the effectiveness and efficiency of regression testing techniques. To date, however, most studies have been limited in terms of their consideration of testing context and system lifetime, and have used cost-benefit models that omit important factors and render some types of comparisons between techniques impossible. These limitations can cause studies to improperly assess the costs and benefits of regression testing techniques in practical settings. In this paper, we provide improved cost-benefit models for use in assessing regression testing methodologies, that incorporate context and lifetime factors not considered in prior studies, and we use these models to compare several common methodologies. Our results show that the factors we consider (in particular, time constraints and incremental resource availability) can affect assessments of the relative benefits of regression testing techniques, and suggest that particular classes of techniques may compare differently across different types of test suites.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing & Debugging—*testing tools*

General Terms

Experimentation, Measurement, Verification

Keywords

Regression testing, regression test selection, test case prioritization, evaluation schemes, economic models, empirical studies

1. INTRODUCTION

As software evolves, engineers use various approaches to assess its quality. One common approach, regression testing, involves saving and reusing (and as necessary, incrementally updating) test suites created for earlier versions of the software [2, 4, 21, 24]. By reusing test cases, this approach amortizes the costs of designing and creating test cases across a system's lifetime. Even so, reusing and maintaining test suites can be expensive, so numerous approaches to reducing or prioritizing regression testing activities have been proposed (e.g., [5, 11, 30, 32, 34]). Initially, research on regression testing — similar to research on testing in general — relied primarily on analytical approaches to assess and compare techniques (e.g. [21, 29]). Testing techniques are heuristics, however, and to properly evaluate their cost-effectiveness in practice, empirical studies are essential.

More recent research on regression testing, therefore [3, 10, 12, 14, 18, 19, 23, 25, 28, 33], has employed empirical studies. A common way to conduct such studies has been to collect one or more software systems with multiple versions, and for each system, find or create a test suite for, and locate or seed faults in, each version. Next, the techniques being studied are applied to each version and its test suite, and the results are assessed using measures of testing effort (numbers of test cases or time required) and effectiveness (rate of fault detection or numbers of faults revealed).

Empirical studies such as these have allowed researchers to compare regression testing techniques in terms of costs and benefits. However, studies to date suffer from several limitations in their abilities to assess cost-benefit tradeoffs relative to practical testing situations. These limitations involve context factors, lifetime factors, and cost-benefit models, and can be summarized as follows.

Context factors. Previous studies have considered only a few context factors when assessing techniques. Most studies have considered differences in programs and regression testing techniques, but none have considered costs of other essential testing activities such as test setup and obsolete test identification, or collection and maintenance of resources (e.g. test coverage information) needed for retesting. And only a few studies have considered the effects of time constraints on testing cost-effectiveness.

Lifetime factors. Previous studies have calculated costs and benefits independently per system version. This "snapshot" view of costs and benefits masks the fact that regression testing techniques are applied repeatedly across system lifetimes. The cost-benefit tradeoffs for techniques across entire lifetimes may be more relevant for choosing a technique than the tradeoffs on single releases.

Cost-benefit models. Previous studies have relied on limited cost-benefit models. Costs are often ignored, or calculated solely in terms of time or numbers of faults missed. Benefits are often calculated solely in terms of reduced test suite size or increased rate of fault detection. Costs of missed faults and human time, and trade-

SIGSOFT'06/FSE-14, November 5–11, 2006, Portland, Oregon, USA.

offs involving product revenue, have not been considered. Moreover, often different techniques are evaluated using different metrics, rendering their relative performance incomparable.

Limitations such as these can make it difficult to empirically compare regression testing techniques, or can lead evaluations to improperly assess the costs and benefits of techniques in practical contexts. Ultimately, this can lead to inaccurate conclusions about the relative cost-effectiveness of techniques, and inappropriate decisions by engineers relying on such conclusions to select techniques for particular situations.

It follows that researchers who empirically investigate regression testing techniques, and practitioners who might act on the results of those investigations, would be better served by empirical investigations founded on more comprehensive cost-benefit models for those techniques, that incorporate richer context and lifetime factors. In this paper, therefore, we provide such a model, and we conduct an empirical study comparing several common regression testing techniques using that model.

The results of our study show that the factors we consider can affect assessments of the relative benefits of regression testing techniques. In particular, the effects of time constraints in assessing techniques are large, and incremental resource availability, though less pronounced in its effects, can also effect assessments of the relative benefits of regression testing techniques. Our results also provide insights into the relative strengths and weaknesses of techniques, with consequences for their application in practice.

Overall, this work makes the following contributions. First, by providing a cost-benefit model that better captures the context and lifetime factors that affect technique cost-effectiveness, we facilitate more accurate interpretation of empirical results by practitioners and researchers. Second, by providing a mechanism for assessing the costs and benefits of various regression testing techniques in terms of a single model using shared units of comparison, we enable researchers to directly compare the cost-benefits tradeoffs between previously incomparable techniques. Third, our particular empirical results add to the growing body of knowledge about the tradeoffs between regression testing techniques, with implications for practitioners who might want to use the techniques studied.

In the next section of this paper, we review relevant previous work. Section 3 presents our cost-benefit model. Section 4 presents our study design, results, and analysis. Section 5 discusses our results, and Section 6 presents conclusions.

2. BACKGROUND AND RELATED WORK

We focus on three regression testing methodologies: retest-all, regression test selection, and test case prioritization.

Let P be a program, let P' be a modified version of P, and let T be a test suite for P. Regression testing attempts to validate P'. As a typical common practice [24], often engineers simply reuse all non-obsolete[1] test cases in T to test P' – this is known as the *retest-all technique*. Rerunning all of these test cases, however, can be very expensive; for example, Srivastava et al. [32] cite a case in which an office productivity application of 1.8 million lines of code has a test suite of 3128 test cases that require over four days to run.

When only small portions of P have been modified, a retest-all technique can involve unnecessary work. *Regression test selection techniques* (e.g., [5, 25, 28, 30], for a survey and additional references see [29]) reduce testing costs by selecting test cases that are necessary to test a modified program. Regression test selection techniques use information about P, P', and T to select a subset T' of T with which to test P'. *Safe* regression test selection techniques are those that ensure (under certain conditions) that T' detects the same faults as T, whereas non-safe techniques provide no such assurance, but rather, trade such assurances for further savings in testing cost (for an in-depth discussion of safety and classes of regression testing techniques, see [29]).

While regression test selection techniques focus on reducing the number of test cases that must be executed, *test case prioritization techniques* (e.g., [11, 32, 34]) reorder test cases, scheduling test cases with the highest priority according to some criterion earlier in the testing process. Test case prioritization techniques can yield benefits such as providing earlier feedback to testers and earlier fault detection. To date, most research on prioritization has focused on this latter goal, typically described as one of improving a test suite's *rate of fault detection*. Furthermore, when organizations cut testing processes short, prioritization can decrease the possibility that faults will have escaped into the released system.

Many regression test selection and test case prioritization techniques, including those that we consider in this work, depend on information about the coverage of code achieved by tests. Such information is obtained by inserting probes into (instrumenting) code, and this activity, along with the activity associated with collecting traces (coverage information) about test execution, are among the costs incurred by these techniques.

Empirical evaluations of the foregoing regression testing methodologies to date (as cited in Section 1) have relied, implicitly or explicitly, on relatively simple cost-benefit models. Leung and White [22] present a model that considers some of the factors (testing time, technique execution time) that affect regression testing costs, but their model does not consider benefits. Harrold et al. [15] present a coverage-based predictor of regression test selection technique effectiveness, but this predictor focuses only on reduction in numbers of test cases. Malishevsky et al. [23] extend Leung and White's work with cost models for regression test selection and test case prioritization that incorporate benefits related to omission of faults and rate of fault detection. As discussed in Section 1, however, each of these models omits many context and lifetime factors related to the costs and benefits of techniques in practice.

3. MODELLING COSTS AND BENEFITS

We now describe the cost-benefit model that we utilize. We begin our discussion by outlining the regression testing process on which our model is based. Section 3.2 describes the constituent costs of regression testing techniques that we model, Section 3.3 presents our model, and Section 3.4 describes how the model is utilized to assess and compare techniques.

3.1 Regression Testing Process

Cost-benefit models capture costs and benefits of methodologies relative to particular processes. In this work, we use a model of the regression testing process that corresponds to the most commonly used approach for regression testing at the system test level [24] — a *batch* process model — and which, though simple, is sufficient to allow us to investigate our research questions.

Figure 1 presents a timeline depicting the maintenance, regression testing, and post-release phases for a single release of a software system following a batch process model. Time *t1* represents the time at which maintenance (including all planning, analysis, design, and implementation activities) of the release begins. At time *t2* the release is code-complete, and regression testing and fault

[1] A test case is obsolete for P if it can no longer be applied to P (e.g. due to changes in inputs), is no longer needed to test P (e.g. due to being designed solely for code coverage of P, and now on P' redundant in coverage) or if its expected output on P' differs (e.g. due to specification changes) from its expected output on P.

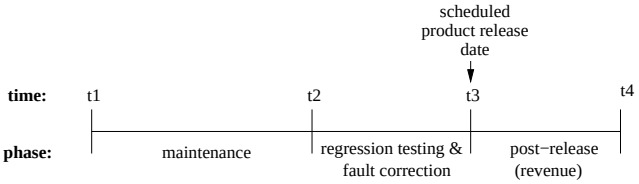

Figure 1: Maintenance and regression testing cycle.

correction begin. (These activities may be repeated and may overlap within time interval *t2*-*t3*, as faults are found and corrected.) When this phase ends, at time *t3*, product release can occur – at this time, revenue associated with the release (together with associated increases in the company's market value) begin to accrue. In a perfect world, actual product release coincides with scheduled release time, following completion of testing and fault correction activities, and this is the situation depicted in the figure.

This process model relates to the regression testing techniques we wish to investigate as follows. Suppose the timeline shows the situation in which the retest-all technique is employed. In this case, regression test selection techniques attempt to reduce time interval *t2*-*t3* by reducing testing time, with, for non-safe techniques, a possible increase in the number of faults that slip through testing and are detected in the post-release phase. Test case prioritization techniques attempt to reduce time interval *t2*-*t3* by allowing greater portions of the fault correction activities that occur in that period to be performed in parallel with testing, rather than afterward. If either of these techniques succeeds, the software can be released prior to its scheduled release date, and overall revenue can increase. If prioritization is unsuccessful and fault correction activities cause time interval *t2*-*t3* to increase, then the release will be delayed and revenue can decrease.

We also use this model to explore one further dimension of regression testing that occurs commonly in practice, involving the interaction between resource availability and process decisions related to product release and revenue. Organizations that create software for sale regression test it with the goal of improving its dependability and attracting greater revenue by reducing the costs of post-release fault correction and increasing the perceived value of the released software and the market value of the company [6]. The cost of this testing activity competes, however, with the desire to field the software earlier, which itself can also result in greater revenue and company market value. Releasing software at time *t2* in the timeline can increase revenue due to the benefits of timeliness, but potentially increases costs due to missed faults.

In practice, pressure to release software and preserve revenue may cause organizations to terminate testing early. In this case, also, revenue may increase but with potential for costs downstream due to missed faults. An analogous situation occurs when the maintenance period runs long and the organization terminates testing early in order to meet scheduled release dates, although in this case the focus is on not losing revenue. Note that in such cases, test case prioritization can decrease the degree to which such costs occur, by increasing the likelihood that faults are detected prior to the termination of testing. In our empirical study we investigate the effects of "early" regression testing termination.

The process model we have just described makes several assumptions. For example, organizations may also create software for reasons other than to create revenue. Organizations that complete testing early could in theory spend additional time performing other forms of verification until the scheduled release date arrives, and this could lead to increased revenue via reduced fault cost downstream. Moreover, revenue itself is not the sole measure of benefit, because market value is also important.

There are also many other regression testing process models that could be considered. For example, some organizations use incremental testing processes, in which test cases are run each night as maintenance proceeds.

These differences noted, this process model does allow us to investigate a cost-benefit model that is much more complex than those used in research on regression testing to date. More important, we believe that the cost-benefit model we present here can be adjusted to accommodate relaxations in the foregoing assumptions, as well as process differences.

3.2 Costs Modelled

We now describe the constituent costs of regression testing techniques that we consider in this work. In this section we focus on what these costs are, not on methods for measuring or estimating them; discussion of measures is provided in Section 4.2.

To model the costs and benefits of regression testing, we consider nine constituent cost components. Here we describe each component and some of the factors that cause it to vary.

Test setup (CS). CS includes the cost of activities required to prepare to run tests, such as setting up the testing environment (hardware and software) and arranging for the use of resources. Thus, CS varies with characteristics of the system under test, such as whether it exists standalone, in a distributed environment, or in an environment involving special hardware or human interaction.

Identifying obsolete test cases (CO_i). CO_i represents the cost of determining which of the test cases in a test suite are still applicable to a new system version to be tested. This cost varies with the type of test cases (e.g., specification-based, code-based, system, unit), the amount of change occurring between consecutive versions, and the availability of documentation or engineer experience.

Repairing obsolete test cases (CO_r). Often, obsolete test cases are still potentially useful for the current system. For example, when a class interface is changed by one parameter type, existing test cases related to that class can not be used directly, but a simple change may repair them. Similarly, a test case for which inputs remain the same but for which expected output has changed can require edits of oracle information. This cost varies with the number of test cases needing repair, and the complexity of the repairs, test cases, oracle procedures, and system.

Supporting analysis (CA). CA represents the cost of the analysis needed to support a regression testing technique. For the techniques being considered here, CA can include costs of instrumenting code, analyzing changes between old and new versions, and collecting test execution traces, and thus can vary widely with characteristics of techniques, programs, tests, and executions.

Significantly, CA can also vary with the extent to which data from previous testing sessions is reused or leveraged in the current testing session. For example, suppose engineers previously instrumented and collected test execution traces for release r_1 of program P in order to apply a regression testing technique to a subsequent release r_2 of P. When r_2 is regression tested, to prepare for the next release, r_3, engineers must instrument and collect test execution traces for r_2. As a software system evolves, however, a large percentage of its code may be shared between consecutive versions. Thus, engineers can re-instrument a version incrementally by identifying code changes between consecutive versions, and using previous instrumentation in unchanged code. Similarly, engineers can collect test execution traces for only the subset of test cases that are affected by instrumentation changes, reusing prior traces for others.

If the costs of instrumentation and trace collection are sufficiently high and the changes between versions are sufficiently small, then we would expect lower costs to be associated with this approach.

Regression testing technique execution (CR). CR represents the cost of applying a regression testing technique or tool (for regression test selection or test case prioritization), itself, after supporting analyses have been completed. This cost also varies with characteristics of techniques, programs, test suites, and changes [13].

Test execution (CE). CE represents the cost of executing tests. This cost varies with test execution processes (e.g., manual, automatic, or semi-automatic), as well as with characteristics of the system under test and the particular test cases utilized. Many organizations attempt to run test cases automatically, but many others continue to use manual or semi-automated testing approaches; for example, in human/machine interface testing, test cases may primarily involve human interaction [6].

Test result validation (CV_d and CV_i). CV_d and CV_i represent the cost of checking test results to determine whether or not test cases reveal failures. These two variables represent two components of the validation task: (1) CV_d is the cost of using automated differencing tools on test outputs to detect output differences with respect to prior testing sessions, and (2) CV_i is the (human time) cost of inspecting test outputs flagged as different to determine whether the difference in fact represents a failure. These costs vary with the number of test cases and the complexity of test output, as well as with the automated technique used to check outputs for differences. A regression testing technique that reduces the number of test cases to be executed also reduces CV_d and CV_i.

Missing faults (CF). CF represents the cost of missing faults. Regression test selection techniques can miss faults due to omission of existing test cases that could, if executed, have revealed them. In this work, we focus on the costs of missing faults that the regression test suite could, if executed in full, have detected. (In addition, all regression testing techniques can miss faults that are not detectable by any of the test cases executed; however, these costs are incurred similarly by all techniques so we do not consider them here.)

CF varies with regression testing technique; clearly, non-safe techniques incur this cost to a greater extent than safe techniques. As discussed in Section 3.1, CF also varies with the testing organization's processes (e.g., with reduced testing time caused by early test termination). Finally, CF varies with business and financial characteristics such as market conditions, product sensitivity to the market, and the severity of missed faults.

Delayed fault detection feedback (CD). CD captures the cost of delayed fault detection feedback. When faults are detected late in a regression testing cycle, efforts to correct them can delay product release. Faults detected early in a cycle can potentially be addressed, prior to completion of the cycle. As a simple example, suppose a fault requiring five days to correct is discovered on the last day of a ten day regression testing cycle. In this case, product delivery is delayed by the four days required to correct the fault, and also by the time required to (again) regression test the corrected program (another ten days under the retest-all approach). If this fault is detected prior to the fifth day of the testing cycle, it does not add any additional delay to product delivery time, beyond the time required to retest the corrected program.

Other costs not considered. In addition to the costs we have described, there are other testing costs, such as initial test case development, initial automation costs, test tool development, test suite maintenance, management overhead, database cost, and the cost of developing new test cases. In this work we restrict our attention to the costs just listed, but our cost-benefit model could easily be extended to incorporate these other costs.

3.3 A Cost-Benefit Model

We use the preceding costs to formulate a cost-benefit model that allows us to investigate the research questions we focus on in this paper. We consider all of the costs just outlined, and for analysis costs we consider two analyses on which the specific regression test selection and test case prioritization techniques we study depend: the cost of inserting instrumentation into the system, and the cost of collecting test traces.

The model that we present is constructed based on the regression testing process model discussed in Section 3.1, but the method we have used to construct the model can be used to construct models for other processes.

Before we describe our cost-benefit model, we define several terms and coefficients that are used in the model, most of which instantiate the general constituent costs outlined in Section 3.2. Assume that we are considering regression testing technique *R*, *n* releases of software system *S* denoted $S_1, S_2, \ldots, S_n$, and *n* versions of test suite *T* (one per release of *S*) denoted $T_1, T_2, \ldots, T_n$.

- *i* is an index denoting a particular release S_i of *S*.
- *u* is a unit of time (e.g., hours or days).
- REV is an organization's revenue in dollars per time unit *u*, relative to *S*.
- $ED(i)$ is the expected time-to-delivery in units *u* for release S_i when testing begins (in Figure 3.1, interval *t2-t3*).
- PS is a measure of the cost (average hourly salary) associated with employing a programmer per unit of time *u*.
- $CS(i)$ is the setup cost for testing release S_i.
- $CO_i(i)$ is the cost of identifying obsolete tests for release S_i.
- $CO_r(i)$ is the cost of repairing obsolete tests for release S_i.
- $CA_{in}(i)$ is the time needed to instrument all units in *i*.[2]
- $CA_{tr}(i)$ is the time required to collect traces for test cases in T_{i-1} for use in analyses needed to regression test release S_i.
- $CR(i)$ is the time required to execute *R* itself on release S_i.
- $CE(i)$ is the time required to execute test cases on release S_i (either all of the test cases in T_i or some subset of T_i).
- $CV_d(i)$ is the cost of applying automated differencing tools to the outputs of test cases run on release S_i (all test cases in T_i or some subset of T_i).
- $CV_i(i)$ is the (human) cost of checking the results of test cases determined to have produced different outputs when run on release S_i (all test cases in T_i or some subset of T_i).
- $CF(i)$ is the cost associated with a missed fault after the delivery of release S_i.
- $CD(i)$ is the cost associated with delayed fault detection feedback on release S_i.
- $a_{in}(i)$ is a coefficient used to capture reductions in costs of instrumentation required for release *i* following changes, in terms of the ratio of the number of units instrumented in *i* to total number of units in *i*:

$$a_{in}(i) = \frac{numberOfUnitsInstrumented}{totalNumberOfUnits} \quad (1)$$

  When all units are instrumented, this ratio equals 1.
- $a_{tr}(i)$ is a coefficient used to capture reductions in costs of the trace collection required for *i* following changes, in terms

[2]Systems can be incrementally instrumented at various levels, such as per file, per class, or per method. We use "unit" generically to account for this; in our studies we consider instrumentation at the level of class files.

of the ratio of the reduced number of traces collected when focusing on changes in *i* to the total number of traces that would need to have been collected otherwise.

$$a_{tr}(i) = \frac{numberOfTracesCollected}{totalNumberOfTraces} \quad (2)$$

When all traces are collected, this ratio equals 1.

- $b(i)$ is a coefficient used to capture reductions in costs of executing and validating test cases for *i*, when only a subset of *T* is rerun:

$$b(i) = \frac{NumberOfTestsRerun}{TotalNumberOfTestsInT} \quad (3)$$

When all test cases are run, this ratio equals 1.

- $c(i)$ is the number of faults that could be detected by *T* on release *i* but that are missed due to execution of subsets of *T*.

To formulate a cost-benefit model incorporating the foregoing costs, we must ensure that all costs are measured in identical units. To do this, we initially record all costs for which the mnemonics take the form CX using a time metric in some unit *u*. We then convert these costs into monetary values so that we can combine them in calculations involving revenues. To perform this conversion, we categorize the costs into two groups: costs related to human efforts (CS, CO_i, CO_r, CV_i and CF), and costs related to machine time (CA_{in}, CA_{tr}, CR, CV_d, and CE).

We then project the cost-benefits of regression testing by considering techniques in light of their business value to organizations, in terms of how much organizations pay for applying the techniques and how much revenue they gain or lose by doing so. This involves two equations: one that captures costs in terms of salaries of the engineers who perform regression testing tasks (using PS to translate time spent by one or more engineers into monetary values), and one that captures revenue gains or losses related to changes in product release time (using REV to translate times into monetary values).

Further, in keeping with our desire to account for *lifetime factors* by tracking costs and benefits across entire system lifetimes, our equations calculate costs and benefits *across entire sequences of system releases*, rather than simply on individual system releases.

The two equations that comprise our model are as follows:

$$Cost = PS * \sum_{i=2}^{n}(CS(i) + CO_i(i) + CO_r(i) + b(i) * CV_d(i) + c(i) * CF(i)) \quad (4)$$

$$Benefit = REV * \sum_{i=2}^{n}(ED(i) - (CS(i) + CO_i(i) + CO_r(i) + a_{in}(i-1) * CA_{in}(i-1) + a_{tr}(i-1) * CA_{tr}(i-1) + CR(i) + b(i) * (CE(i) + CV_i(i) + CV_d(i)) + CD(i))) \quad (5)$$

Relating these formulas to our prior discussions of processes and cost-benefits, if an organization does not test their product at all before delivery, then they gain potential revenue by reducing all of the cost terms other than CF in Equation 4 to zero, and all the cost terms of form CX in Equation 5 to zero. If CF is zero, the resulting revenue increase is proportional to the saved expected delivery time ED. When a regression testing technique reduces (increases) testing time, either through selection or prioritization, the right hand side of Equation 5 is positive (negative), indicating an increase (decrease) in revenue. These revenue changes are coupled, however, with changes in costs captured in Equation 4 in determining whether techniques are cost-beneficial overall.

Note that of the costs that we consider in this work, several (CS, CO_i, CO_r, CA) can potentially be partially offloaded from the critical testing phase to the maintenance phase; that is, the phase denoted *t1-t2* in Figure 1. For example, test engineers can make test hardware ready or perform preliminary analyses on modules on which maintenance is complete. In this case, costs may decrease: they continue to have associated salary and hardware aspects, but may be less likely to contribute directly to delays in release dates. Four other costs (CR, CE, CV_i, CV_d) are incurred primarily during the regression testing phase. CD occurs during the regression testing and fault correction phase, but may also extend into the post-release phase. CF is incurred during the post-release phase.

In constructing the foregoing model we make several simplifying assumptions. We assume that *S* has just one (evolving) test suite, that tests have equal run times, that instrumentation costs per unit and trace are uniform, and that fault costs are all the same. We assume that test case execution, analysis, and regression testing technique costs involve only machine time, with no human cost component, and we consider test setup and obsolete test detection to have only human effort cost, (an assumption appropriate to our experiment objects). In this work, where we consider the relative efficacy of regression testing techniques that re-use *T*, we consider only fault losses incurred due to execution of subsets of *T*. We make these assumptions for convenience, as they are suitable to the scenarios we consider in our empirical study. All of these assumptions can be relaxed, however, given appropriate changes made in the model and sufficiently accurate measurement instruments.

3.4 Evaluating and Comparing Techniques

The foregoing cost models can be used in cost-benefit analyses in various ways. Let *A* and *B* be regression testing techniques with costs $Cost_A$ and $Cost_B$, and benefits $Benefit_A$ and $Benefit_B$. We can determine whether *A* is beneficial by calculating:

$$Benefit_A - Cost_A \quad (6)$$

Further, we can determine the difference in value between A and B by calculating:

$$(Benefit_A - Cost_A) - (Benefit_B - Cost_B) \quad (7)$$

with positive values indicating that A has greater value than B, and negative values indicating that A has lesser value than B.

4. EMPIRICAL STUDY

The foregoing model captures a richer set of factors than have been considered in prior research on regression testing techniques, and allows us to address various questions about those techniques. Our study is designed to address three such questions:

RQ1: What effect does the imposition of time constraints have on the relative cost-benefits of regression testing techniques?

RQ2: What effect does availability of incremental resources have on the relative cost-benefits of regression testing techniques?

RQ3: What are the relative cost-benefits of regression test selection and test case prioritization techniques?

None of these research questions have been addressed previously in empirical studies of regression testing; In fact, no cost-benefit models previously defined capture the necessary factors. In the case of RQ1 and RQ2, no prior models consider the relationship between fault omission or rate of fault detection and technique execution costs. In the case of RQ3, no prior models have been capable of expressing the cost-effectiveness of these two classes of

techniques in comparable units. All three of these questions are important, however, for practitioners who wish to determine what technique might be most cost-effective in their organizations.

4.1 Objects of Analysis

We used five Java programs as objects of analysis: *ant*, *xml-security*, *jmeter*, *galileo*, and *nanoxml*. The first three objects have JUnit test suites, and the last two have TSL (Test Specification Language) suites [26]. *Ant* is a Java-based build tool; it is similar to make, but it is extended using Java classes instead of with shell-based commands. *Jmeter* is a Java desktop application used to load-test functional behavior and measure performance. *Xml-security* implements security standards for XML. *Galileo* is a Java bytecode analyzer. *Nanoxml* is a small XML parser for Java. Several sequential versions of each of these systems, modified to varying degrees, were selected for use in this study. These programs, versions, and test suites are all available as part of an infrastructure supporting experimentation [8].

Table 1 lists, for each of our objects, its associated "Versions" (the number of versions of the object program), "Classes" (the number of class files in the most recent version of that program), "Size (KLOCs)" (the total lines of code in the most recent version of that program), and "Test Cases" (the number of test cases available for the most recent version of that program). The rightmost column is described in Section 4.3.

4.2 Variables and Measures

4.2.1 Independent Variable

Our study manipulated one independent variable, regression testing technique. We consider the three different regression testing methodologies described earlier in Section 2: retest-all, regression test selection, and test case prioritization. For each of these methodologies, we consider one or more specific techniques, as follows.

Retest-all (control). The retest-all technique (reusing an entire existing test suite) together with original test case order serves as our control technique, representing the typical common practice of running all non-obsolete test cases on a new version of a system, in whatever order they are presented in.

Regression test selection. For regression test selection we consider a safe technique, that selects test cases which exercise code that has been changed to produce a modified program version [30]. The technique relies on control flow graphs and program coverage information at the basic block level to select all test cases that execute changed code.

Test case prioritization. For test case prioritization we consider two coverage-based techniques [11]. These techniques rely on block coverage information per test case. The first technique, total block coverage prioritization, simply counts the total number of blocks each test case covers and sorts test cases in terms of those counts. This technique has relatively low analysis costs. The second technique, additional block coverage prioritization, orders test cases in terms of the numbers of additional blocks they cover by greedily selecting the test case that covers the most as-yet-uncovered blocks until all blocks are covered, then repeating this process until all test cases have been used. This second technique incorporates a notion of feedback not present in the total block coverage prioritization, which causes it to have larger analysis costs than that technique.

Techniques facing time constraints. We also consider each of the techniques just described in a manner that reflects the effects of time constraints, in which regression testing activities are terminated early. To do this, for each of the foregoing techniques, we shorten the test execution process by 50%, simulating the effects of having the testing process halted half way through.

Table 1: Experiment Objects and Associated Data

Objects	Versions	Classes	Size (KLOCs)	Test Cases	Mutants
ant	9	627	80.4	877	2907
xml-security	4	143	16.3	83	127
jmeter	6	389	43.4	78	295
galileo	16	87	15.2	1533	1923
nanoxml	6	26	7.6	216	132

Techniques using incremental resources. To investigate the effects of incremental resource availability, we consider versions of each of our prioritization techniques and our regression test selection technique that re-use analysis data pertaining to instrumentation from previous testing sessions. In contrast to the non-incremental techniques just discussed, which re-instrument all code and re-execute all tests under instrumentation, these incremental techniques re-instrument only classes that have changed, and re-execute only test cases known to have passed through changed classes previously.

4.2.2 Dependent Variables and Measures

Our dependent variables are the cost and benefit factors presented in Section 3.3, and calculated by Equations 4 and 5. These values are measured in dollars, and their calculation depends on several constituent cost measures, which we collect as follows.

Cost of test setup (CS). For our objects, the cost of test setup involves only human resources, not hardware resources. The relevant activities include setting up a working directory for testing, compiling the program version to be tested, configuring test drivers and test scripts, and (in some cases) performing minor edits to test scripts. We measured the costs of these activities directly as an average of the time taken by two graduate students (Ph.D. students from our research group) to perform them.

Cost of identifying obsolete test cases (CO_i). For our objects, identification of obsolete test cases as versions were developed would have required manual inspection of a version and its test cases, and determination, given modifications made to the system, of the test cases that need to be modified for the next version (due to changes in inputs or expected output). Our objects were already provided with test suites, so to measure this cost we asked a graduate student to perform these activities, working with the given suites.

Cost of repairing obsolete test cases (CO_r). For our objects the cost of repairing obsolete test cases includes the costs of examining specifications, existing test cases, and test drivers, as well as observing the execution of suspect tests and drivers. Although all of our objects had obsolete test cases, and the cost of identifying them was measured as described above, on only one object, *nanoxml*, were repaired tests present. To measure the cost of repairing tests on this object, we asked two graduate students (Ph.D. students from our research group) to perform these activities. We averaged the times taken by these students.

Cost of supporting analysis – non-incremental (CA). The analysis costs for the non-incremental regression testing techniques include the costs of instrumenting programs (CA_{in}) and collecting test execution traces (CA_{tr}). We calculated these values directly for each version of each object program, by measuring the time required to run the Sofya system [20] for instrumentation of Java bytecode, and the time required to execute the test cases for that version on that instrumented version.

Cost of supporting analysis – incremental (CA). Incremental analysis costs consist of the time required to re-instrument only modified classes for a given version (given a version previously fully instrumented), and the time required to re-execute, on that version, only those test cases known to have reached modified classes in the prior version. Our code instrumenter does not support incremental instrumentation, so we partially estimated these values by utilizing the directly measured non-incremental analysis costs collected as just described, and (as shown in Equation 5), multiplying this number by (in the case of re-instrumentation) the ratio of the number of classes requiring re-instrumentation to the total number of classes and (in the case of re-execution) the ratio of the number of traces requiring recollection to the total number of traces.

Cost of regression testing technique execution (CR). We directly measured the time required to apply each regression testing technique studied, by running it against each version of each object program using appropriate analysis information.

Cost of test execution (CE). For cases in which all test cases were executed, we directly measured execution time of test suites automatically, by running them against each version of each object program using appropriate analysis information. For cases in which a subset of a test suite was executed, we estimated execution time by multiplying the cost of executing the entire test suite by the ratio of the number of test cases being rerun to the total number of test cases, as shown in Equation 5.[3]

Cost of test result validation (automatic via differencing) (CV_d). For cases in which all test cases were executed, we directly measured this validation time automatically, by measuring the cost of running a differencing tool on test outputs as test cases were executed, for each version of each object program. For cases in which a subset of a test suites was executed, for reasons similar to those discussed immediately above, we estimated this time by multiplying the cost of validating the entire test suite by the ratio of the number of test cases being rerun to the total number of test cases.

Cost of test result validation (human via inspection) (CV_i). To measure the cost of validating test results, we averaged the time taken by two graduate students (Ph.D. students from our research group) to compare program outputs across versions, for each pair of versions. For cases in which a subset of a test suites was executed, we estimated this time (for reasons discussed above) by multiplying the cost thus measured by the ratio of the number of test cases being rerun to the total number of test cases.

Cost of missing faults (CF). For each regression testing technique that could omit faults, we measured the number of faults omitted during a testing session on each version of each object program. Determining the cost of missing faults, however, is much more difficult. Given the many factors that can contribute to these costs, and the long-term nature of these costs, we could not obtain this measure directly. Instead, we rely on data provided in [31] to obtain estimates of the costs of faults. Because fault difficulties range widely, we decided to analyze results relative to two classes of fault importance: one corresponding to costs attributed in [31] to "severe" faults, and one corresponding to costs attributed to "ordinary" faults. These costs, respectively, are 22 and 1.5 hours.

[3]We used estimation in this case for two reasons: (1) the cost of executing every test suite subset considered in this study was large; and (2) because the test cases for each of our particular object versions are quite similar to one another in terms of execution times, and test suite execution time ultimately accounts for a small fraction of overall costs, this estimation could not affect overall results.

Cost of delayed fault detection feedback (CD). For each prioritization technique applied to each object version and test suite, we measured the rate of fault detection using the APFD (Average Percentage Faults Detected) metric (a metric introduced for this purpose in [12]) for that version and test suite. Then, following the approach of [23], we translated APFD scores into the cumulative costs (in time) of waiting for each fault to be exposed while executing test cases under a particular order, defined as *delays*.

Revenue (REV). A second metric that we cannot measure directly relative to our object programs involves revenue, and to utilize our cost models we required an estimate of this value. To obtain such an estimate, we utilized revenue values cited in survey data from software products [7], ranging from $116,000 to $596,000 per employee. Because our object programs are relatively small compared to many commercial software systems, we utilize the smallest revenue and a headcount of ten in this study.

Programmer salary (PS). A third metric that we cannot measure directly on our object programs involves the salaries of programmers. To obtain an estimate, we rely on a figure of $100 per person-hour, obtained by adjusting an amount cited in [17] by an appropriate cost of living factor.

Expected time-to-delivery (ED). We do not calculate ED, because the comparisons we need to perform to address our research questions do not require its calculation. To explain: we use Equation 7 to compare techniques, and this equation subtracts the benefit value for a second technique from the benefit value for the first. In so doing, because ED is necessarily identical for two techniques compared on the same version, the value of ED is canceled out.

4.3 Experiment Setup and Analysis Strategy

To perform test case prioritization and regression test selection we required two types of data: coverage information and fault data. We obtained coverage information by running test cases on our object programs instrumented using Sofya [20]. The resulting information lists which test cases exercised which blocks in the program; a previous version's coverage information is then used to prioritize a current version's set of test cases, and to support the selection of a subset of test cases for the current version.

To measure rate of fault detection for test case prioritization techniques, and fault omission for non-safe regression test selection, we required object programs containing faults. The object programs we obtained had not been supplied with any such faults or fault data. Thus we used mutation faults generated using a Java bytecode mutant generator [9].[4] Because our focus is regression testing, however, we use only generated mutants that fall within modified areas of code. The number of mutants created for each of our object programs is shown in column five of Table 1.

In actual testing scenarios, programs do not typically contain as many faults as the number of mutants we generated. Also, we wish to investigate the use of regression testing techniques (relative to the lifetime factor) across the entire sequences of versions of our object programs. To do this, for each version of each program we randomly selected several *mutant groups* from the mutant pool for that version; each mutant group's size varied randomly between one and five.[5] Then, for each program, we obtained four *sequences*

[4]Although studies involving real faults can be preferable for external validity, real faults are seldom available in numbers sufficient to support controlled experimentation; thus, researchers often rely on faults created by mutation tools. Recent studies [1, 9] have shown, moreover, that mutation faults can be representative of real faults.

[5]These numbers were chosen to maintain consistency with setup procedures followed in an earlier experiment [9].

of mutant groups by randomly selecting a mutant group for each version of that program.

Given these materials, to collect the data necessary to investigate our research questions, we considered each object program in turn, and for each version of that program, applied each regression testing technique, and collected the appropriate values for necessary cost variables (as indicated in Section 4.2.2). In this process, all times were measured on a PC running SuSE Linux 9.1 with 1G RAM and with a 3 GHZ processor.

Given these cost variables we calculated, for each object program and each technique, the benefit and cost of that technique applied to the sequence of versions (with their associated test suites) of that program for each of its four sequences of mutant groups. We then averaged these numbers. These benefit and cost numbers serve as the data for our subsequent analysis.[6]

4.4 Threats to Validity

In this section we describe the construct, internal, and external threats to the validity of our study, and the approaches we used to limit the effects of these threats.

Construct Validity. The dependent measures that we have considered for costs and benefits are not the only possible measures related to regression testing cost-effectiveness. As described in Section 3.2, other testing costs might be worth measuring for different testing situations and organizations.

Internal Validity. The inferences we have made about the cost-benefits of regression testing techniques could have been affected by two factors. The first factor involves potential faults in the tools that we used to collect data. To control for this threat, we validated our tools on several simple Java programs. The second factor involves the actual values we used to calculate costs, some of which involve estimations. For example, we used code change ratios to estimate incremental instrumentation costs, and an average test case execution time over the instrumented program to estimate incremental trace collection costs. We also measured the costs of test setup, finding obsolete tests, repairing obsolete tests, and validating outputs by measuring the time taken by one or two graduate students. The use of such estimates could confound results. The values we used for revenue and costs of correcting and missing faults are estimated based on surveys found in the literature, but such values can be situation-dependent; for example, Perry and Stieg [27] present a different set of fault costs. However, we did choose a relatively small revenue figure so as not to inflate results, given that our object programs are relatively small. In summary, we exercised care in selecting reasonable estimations relevant to our object programs, but larger-scale industrial case studies will be needed to follow up on these results.

External Validity. The Java programs that we study are relatively small (7K - 80K), and their test suites' execution times are relatively short. Complex industrial programs with different characteristics may be subject to different cost-benefit tradeoffs, including also different amounts of revenue that could yield different cost-benefit tradeoffs. The testing process we used is not representative of all processes used in practice, and our results should be interpreted in light of this. The tools we use in this study are prototypes, and thus may not reflect tools used in a typical industrial environment. Control for these threats can be achieved only through additional studies with wider populations of programs, other testing processes, and enhanced performance-efficient tools.

[6]Complete data sets can be obtained by contacting the first author.

Incremental Analysis Resources

Time Constraints	No	Yes
No	**BOX 1** org original order rta retest−all rts safe reg. test. sel. tot total cov. prio. add addt'l cov. prio.	**BOX 2** org original order rta retest−all rts.i safe reg. test. sel. tot.i total cov. prio. add.i addt'l cov. prio.
Yes	**BOX 3** org.50 original order rta.50 retest−all rts.50 reg. test. sel. tot.50 total cov. prio. add.50 addt'l cov. prio.	**BOX 4** org.50 original order — — tot.50.i total cov. prio. add.50.i addt'l cov. prio.

Figure 2: Cost factor scenarios.

4.5 Data and Analysis

In our analysis of results, in keeping with our research questions, we organize the data considering two different context factors: time constraints (captured in our process model, and through various factors in our cost-benefit model, through the early termination of testing activities), and availability of incremental analysis resources (captured in our cost-benefit model in terms of incorporation of differing forms of analysis costs in relation to instrumentation and trace collection). The combinations of these context factors yield four classes of technique applications, as illustrated in Figure 2. Each of these classes (each of the four boxes in the figure) denotes a different scenario that an organization could face in testing, depending on resource availability and time constraints. We describe each scenario further as follows.

Upper left (Box 1): no time constraints are applied and no incremental analysis resources are available. In this situation, five of the regression testing techniques we consider apply: three test case prioritization techniques (original test order ("org"), total block coverage ("tot"), and additional block coverage ("add")), and two regression test selection techniques (retest-all ("rta") and regression test selection ("rts")).

Upper right (Box 2): no time constraints are applied and incremental analysis resources are available. In this situation we consider the same five techniques considered in Box 1, but three of the heuristics ("tot", "add", and "rts") use incremental analysis resources. To identify techniques succinctly we add the tag "i" to each technique's mnemonic: "tot.i", "add.i", and "rts.i.".

Lower left (Box 3): time constraints are applied and no incremental analysis resources are available. In this situation we also consider five techniques, representing the case in which testing activities following the application of techniques in Box 1 are terminated early. We eliminate the second half of the test suites for four of the techniques ("org.50", "tot.50", "add.50", and "rta.50").[7] For regression test selection, we chose a different approach, randomly selecting half of the test cases in the test suite, because we wished all test suites for a given version in Box 3 to have the same size.

Lower right (Box 4): time constraints are applied and incremental analysis resources are available. In this situation we consider only three techniques, "org.50", "tot.50.i", and "add.50.i", because incremental analysis does not apply to a test suite obtained (for regression test selection) by random reduction.

[7]In principle test suites are sets, but in practice test cases are ordered, and thus the notion of using the first half of a suite applies.

To address each of our research questions, we need to compare pairs of techniques for cost-benefit tradeoffs, and then compare the relationships that occur between techniques under one set of factors to the relationships that occur under another set of factors. For example, we ask whether the relationship between "org" and "rts" in Box 1 is the same as the relationship between "org" and "rts" in Box 3, in order to assess whether the effects of early test termination affect the relative cost-benefits of these two techniques.

We first perform technique comparisons within each box. Tables 2 and 3 summarize the result of this comparison, reporting relative cost-benefit relationships measured for each pair of techniques within each box, per program, using Equation 7. Table 2 contains one subtable corresponding to Box 1, one subtable corresponding to Box 2 and two subtables for Box 3 – one for the case in which non-severe faults are utilized in the cost-benefit equations, and another for the case in which severe faults are utilized. Table 3 contains data for both types of faults with respect to Box 4. (The use of pairs of tables for Boxes 3 and 4 corresponds to our wish to analyze results relative to two classes of faults differing in severity. Note, however, that differences between fault severities have effects only for cases in which time constraints limit test execution, because when constraints are not applied and full test suites are executed, there are no omitted faults and thus no fault costs. Thus these results are reported only for Boxes 3 and 4.)

All of the data in Tables 2 and 3 is represented in dollar values, obtained by converting time measurements using the formulas and values described in Sections 3.3 and 4.2.2, respectively.

Within each subtable in the tables, columns are labeled with pairs of regression testing techniques compared, and rows are labeled with object programs considered. If an entry in the table in Column $B(T1, T2)$ and row *foo* contains a positive amount, then $T1$ yields benefit by that amount, in dollars, over $T2$, for program *foo*. If an entry in Column $B(T1, T2)$ and row *foo* contains a negative amount, then $T2$ yields benefit by that amount, in dollars, over $T1$ for *foo*. For example, the cell in Column $B(tot, org)$, row *ant*, in the topmost subtable in Table 2, lists the result of applying Equation 7 treating "tot" as technique *A* and "org" as technique *B*; the amount listed, -916, is the dollar-cost advantage (or rather, disadvantage) of applying *A* rather than *B* to *ant*.

We now use the data in Tables 2 and 3 to address each of our research questions, in turn.

4.5.1 RQ1: Effects of time constraints

Our first research question considers whether the imposition of time constraints affects the relative cost-benefits of regression testing techniques. To answer this question, we compare technique pairs in Boxes 1 and 2 in Figure 2 to corresponding technique pairs in Boxes 3 and 4, respectively. We restrict our attention to comparisons between heuristics and control techniques, deferring comparisons between regression test selection and test case prioritization techniques to our discussion of RQ3.

Columns 2 through 4 in Table 2, in the subtable for Box 1, indicate that heuristic regression testing techniques are not beneficial compared to corresponding control techniques for any of the object programs considered. All comparisons yield negative numbers, indicating that the original and retest-all techniques outperformed the heuristics in all cases. Data in the same columns in the subtable for Box 2 also shows similar trends in the cases of columns 2 and 4 ("tot" versus "org" and "rts" versus "rta"), but not in the case of Column 3 ("add.i" versus "org").

Comparing this data to that for corresponding technique comparisons in Boxes 3 and 4 for non-severe faults reveals different trends: in all but three cases in Box 3 and one in Box 4, heuristics outperform control techniques. Furthermore, even the few cases in which heuristics are not beneficial over control techniques are altered when we consider the case in which faults are severe.

Table 2: Relative Benefits Between Technique Pairs (dollars)

No incremental analysis resource & no time constraints (Box 1)					
Object	B(tot, org)	B(add, org)	B(rts, rta)	B(rts, tot)	B(rts, add)
ant	-916	-1083	-1779	-616	-448
jmeter	-298	-297	-552	-146	-147
xml	-100	-104	-223	-63	-59
nanoxml	-70	-41	-188	179	150
galileo	-815	-322	-415	2576	2083
Incremental analysis resource & no time constraints (Box 2)					
Object	B(tot.i, org)	B(add.i, org)	B(rts.i, rta)	B(rts.i, tot.i)	B(rts.i, add.i)
ant	-599	148	-1483	-616	-448
jmeter	-145	155	-399	-146	-147
xml	-25	70	-148	-63	-59
nanoxml	-32	66	-150	179	150
galileo	-583	729	-182	2576	2079
No incremental analysis resource & time constraints (Box 3: non-severe faults)					
Object	B(tot.50, org.50)	B(add.50, org.50)	B(rts.50, rta.50)	B(rts.50, tot.50)	B(rts.50, add.50)
ant	160	184	656	527	-1069
jmeter	-96	-36	145	258	-158
xml	572	461	437	-132	-184
nanoxml	694	559	947	269	156
galileo	-668	889	1822	2731	-1722
No incremental analysis resource & time constraints (Box 3: severe faults)					
Object	B(tot.50, org.50)	B(add.50, org.50)	B(rts.50, rta.50)	B(rts.50, tot.50)	B(rts.50, add.50)
ant	17800	21455	9994	-7773	-13002
jmeter	3016	4114	2221	-778	-2234
xml	10949	9281	6663	-4282	-2778
nanoxml	14183	11973	14436	269	2231
galileo	9707	42393	27762	18295	-17286

Table 3: Relative Benefits Between Technique Pairs (dollars)

Incremental analysis resource & time constraints (Box 4)				
Object	B(tot.50.i, org.50) non-severe	B(tot.50.i, org.50) severe	B(add.50.i, org.50) non-severe	B(add.50.i, org.50) severe
ant	477	18116	500	21771
jmeter	57	3170	117	4267
xml	647	11023	536	9356
nanoxml	732	14221	597	12011
galileo	-436	9931	1122	42626

4.5.2 RQ2: Effects of incremental resource use

Our second research question considers whether the availability of incremental resources affects the relative cost-benefits of regression testing techniques. To answer this question, we compare technique pairs in Boxes 1 and 3 in Figure 2 to corresponding technique pairs in Boxes 2 and 4, respectively. Again, we focus on comparisons between heuristics and control techniques.

As already noted in Section 4.5.1, all three comparisons among heuristics and control techniques in Box 1 show no benefits accruing to heuristics. When we consider the use of incremental analysis resources (Box 2), however, comparisons do reveal a few differences. First, in all cases, the use of incremental analysis yields advantages over the use of non-incremental analysis: all numbers in the table are higher than their corresponding numbers in Box 1. In the comparisons of "tot" to "org" and "rts" to "rta", however, control techniques continue to outperform heuristics overall. A second difference, however, is more apparent: in the comparison of "add" to "orig", the use of incremental analysis does render the heuristic beneficial with respect to the control technique.

Comparisons between Box 3 and Box 4 do not reveal many differences, but here too, overall the benefits associated with heuristics increase, and in two cases ("tot.50" versus "orig.50" and "add.50" versus "orig.50" for *jmeter*) the use of incremental analysis resources allows heuristics to outperform control techniques.

4.5.3 RQ3: Test selection versus prioritization

Our third research question considers whether the relative benefits of regression test selection and test case prioritization techniques differ. Columns 5 and 6 in the subtables for Boxes 1 and 2 show the comparison results between these techniques when no time constraints are applied, and when safe regression test selection is involved. (Note that Columns 5 and 6 in Box 1 contain values identical to those in Box 2; this is because the techniques used in the two boxes differ only in terms of their use of incremental analysis resources, and in the case of these particular techniques, where time constraints are not applied, the costs of the activities performed do not differ across the boxes.)

The results show that the regression test selection technique ("rts") is more cost-effective than test case prioritization techniques ("tot" and "add") for the two object programs (*nanoxml* and *galileo*) that have specification-based test suites. For the other three programs, which use JUnit test suites, test case prioritization techniques are more beneficial than regression test selection. This result is important because it suggests that in practice, a preferred technique might vary with test suite type. Further study of this effect is needed, however, to determine whether test suite type, technique, or their interaction are responsible for this effect.

Turning to the subtable for Box 3, when we compare results between test case prioritization and regression test selection in the case in which time constraints apply, we see different relationships between techniques. For non-severe faults, the selection technique is better than the "tot" technique in all but one case, but the "add" technique is better than selection in all but one case. For severe faults, the comparison between random and "add" reveals trends similar to that of the non-severe fault case, but the comparison between selection and "tot" reveals two cases (*ant* and *jmeter*) in which selection ceases to be better than the "tot" technique.

5. DISCUSSION

To further explore the results of our study we consider two topics: (1) the ramifications for practice of the results we obtained; and (2) a comparison of our results with those obtained in earlier work using different cost models.

Where the first topic is concerned, our results support the conclusion that accounting for different context factors in assessing regression testing techniques makes a difference when assessing the relative benefits of those techniques. In particular, our analysis shows that the time constraints factor had a large impact on relative benefits. In practice, cases in which time constraints intervene to affect product release are frequent in the software industry; Hendricks et al. [16] report that a typical reason for product delays is the need for additional testing and debugging. At other times, organizations cut back on testing activities in order to ensure timely release of their product.

Further study of our data suggests, in fact, that the primary cause of the impact of time constraints was the tradeoff between the costs of applying additional tests and not missing faults, and the costs of reduced (non-safe) testing in which faults are missed. On-time but incomplete-test delivery can lead to revenue increases, but if the product contains defects after delivery, the organization can suffer from post-delivery revenue losses (due to additional defect removal costs and the loss of customers due to distrust of the product). Meanwhile, complete-test but late delivery can lead to smaller numbers of post-release defects, but if the delivery date is delayed long, the company can lose opportunities to earn revenue from the product. These inferences are not unexpected, but what our empirical results suggest is that cost models such as ours can be used to ascertain the regression testing technique that can best be used in a particular scenario, based on expected revenues and values of other factors related to testing costs.

Regarding the use of incremental resources, our results show that this factor, too, can affect evaluations of regression testing techniques, but such impact was apparent in only some cases. One cause of this was the relationship observed, for our objects, between instrumentation and trace collection costs. In general, we expect that if we reduce the number of class files that need to be instrumented to collect information for a testing session, we could also reduce the number of trace files to be collected by a proportional amount, because we need to collect only traces that are affected by instrumentation changes. However, this expectation was often not met on our objects. For example, in the case of *nanoxml*, incremental instrumentation required only 30% of total instrumentation time, but incremental trace collection required 98% of total trace collection cost. This result occurred because some of the newly instrumented files are accessed by most test cases. The lesson learned from this example, where our study and the use of cost models are concerned, is that it can be important to decouple factors in those models, to avoid conflating different effects.

Where our second topic of discussion is concerned, in this work we evaluated regression testing techniques using a cost model that (1) allows comparisons of previously incomparable classes of techniques (prioritization and selection) and (2) includes a richer set of factors than has been employed in prior evaluations. Our comparison of prioritization and selection (RQ3) illustrates the effects of considering such factors on the relative cost-effectiveness of these classes of techniques: not only time constraints, but also fault severity and test suite type potentially affect tradeoffs between them.

To gain further insights into how evaluations of regression testing techniques differ as cost models vary, we compared our results with those from a previous empirical study of prioritization techniques in which three of the same JUnit object programs were used (*ant*, *jmeter*, and *xml-security*) [10]. The previous study evaluated a prioritization technique (additional block coverage) using a cost model that considered only two cost components (test case execution time and prioritization technique cost) and one benefit (fault detection rate). The study showed that the additional block coverage technique was beneficial compared to original test case orderings for two of the three programs (*jmeter*, and *xml-security*). This result is quite different from our results in this study, which do not show benefits for any heuristics over control techniques in the case in which time constraints do not apply. One lesson suggested by this observation is that evaluations of techniques based on different models can result in quite different evaluations of the cost-benefits of techniques, and so, efforts to capture richer sets of factors in models, as we have done in this work, are worthwhile.

6. CONCLUSIONS

Empirical assessments of regression testing techniques depend on cost-benefit models. In this paper we have presented such a model, that captures a richer set of the factors (including context and lifetime factors) that affect technique cost-effectiveness than prior models. Our model facilitates the investigation and comparison of techniques along dimensions that have not previously been possible, and our empirical results indicate that this expanded view has practical implications for users and researchers of techniques.

Although the cost-benefit model that we present captures specific testing-related factors relative to just one (common) regression testing process, it can be adapted to include other factors and apply to other processes, and our future work will consider such adaptations. Further, our study results are somewhat explorative, in the sense that they do not provide data sufficient to support statistical analysis. Such results are important in the early stages of research to show whether value potentially exists in models; however, having shown this, this work motivates future studies employing larger data sets and statistical analysis.

In the study reported in this paper, we evaluated regression testing techniques using systems of size (7K - 80K) and relatively small revenue estimates. Program size, however, does not appear to be a factor in our results, for the programs that we consider; thus, we conjecture that similar trends could be expected for larger, industrial systems. Such larger systems, however, will also be associated with higher revenues than those considered here, and we expect that in such cases, the context factors we have considered will have an even greater impact on the relative cost-benefits of regression testing techniques. We hope that through continuing research in this area, we can bring the benefits of better cost models and expanded empirical understanding to organizations that create such systems.

Acknowledgements

This work was supported in part by NSF under Awards CCR-0080898 and CCR-0347518 to the University of Nebraska - Lincoln.

7. REFERENCES

[1] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Int'l. Conf. Softw. Eng.*, pages 402–411, May 2005.

[2] B. Beizer. *Black-Box Testing*. John Wiley and Sons, New York, NY, 1995.

[3] J. Bible, G. Rothermel, and D. Rosenblum. Coarse- and fine-grained safe regression test selection. *ACM Trans. Softw. Eng. Meth.*, 10(2):149–183, Apr. 2001.

[4] R. Binder. *Testing Object-Oriented Systems*. Addison Wesley, Reading, MA, 2000.

[5] Y. Chen, D. Rosenblum, and K. Vo. TestTube: A system for selective regression testing. In *Int'l. Conf. Softw. Eng.*, pages 211–220, May 1994.

[6] R. D. Craig and S. P. Jaskiel. *Systematic Software Testing*. Artech House Publishers, Boston, MA, first edition, 2002.

[7] http://www.culpepper.

[8] H. Do, S. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Int'l. J. Emp. Softw. Eng.*, 10(4):405–435, 2005.

[9] H. Do and G. Rothermel. A controlled experiment assessing test case prioritization techniques via mutation faults. In *Conf. Softw. Maint.*, pages 113–124, Sept. 2005.

[10] H. Do, G. Rothermel, and A. Kinneer. Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis. *Int'l. J. Emp. Softw. Eng.*, 11(1):33–70, 2006.

[11] S. Elbaum, A. Malishevsky, and G. Rothermel. Prioritizing test cases for regression testing. In *Int'l. Symp. Softw. Test. Anal.*, pages 102–112, Aug. 2000.

[12] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE Trans. Softw. Eng.*, 28(2):159–182, Feb. 2002.

[13] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Softw. Quality J.*, 12(3), 2004.

[14] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test selection techniques. *ACM Trans. Softw. Eng. Meth.*, 10(2):184–208, Apr. 2001.

[15] M. J. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker. Empirical studies of a prediction model for regression test selection. *IEEE Trans. Softw. Eng.*, 27(3):248–263, Mar. 2001.

[16] K. B. Hendricks and R. V. Singhal. Delays in new product introductions and the market value of the firm: The consequences of being late to the market. *Mgmt. Science*, 43(4):422–436, Apr. 1997.

[17] C. Jones. *Applied Software Measurement: Assuring productivity and quality*. McGraw-Hill, 1997.

[18] J. Kim and A. Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *Int'l. Conf. Softw. Eng.*, May 2002.

[19] J. Kim, A. Porter, and G. Rothermel. An empirical study of regression test application frequency. In *Int'l. Conf. Softw. Eng.*, pages 126–135, June 2000.

[20] A. Kinneer, M. Dwyer, and G. Rothermel. Sofya: A flexible framework for development of dynamic program analysis for Java software. Technical Report TR-UNL-CSE-2006-0006, University of Nebraska–Lincoln, Apr. 2006.

[21] H. Leung and L. White. Insights into regression testing. In *Conf. Softw. Maint.*, pages 60–69, Oct. 1989.

[22] H. Leung and L. White. A cost model to compare regression test strategies. In *Conf. Softw. Maint.*, Oct. 1991.

[23] A. Malishevsky, G. Rothermel, and S. Elbaum. Modeling the cost-benefits tradeoffs for regression testing techniques. In *Conf. Softw. Maint.*, pages 204–213, Nov. 2002.

[24] K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma. Regression testing in an industrial environment. *Comm. ACM*, 41(5):81–86, May 1988.

[25] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. In *Found. Softw. Eng.*, Nov. 2004.

[26] T. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Comm. ACM*, 31(6), June 1988.

[27] D. E. Perry and C. S. Stieg. Software faults in evolving a large, real-time system: A case study. In *Eur. S.E. Conf.*, 1993.

[28] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley. Chianti: A tool for change impact analysis of Java programs. In *O.O. Prog., Sys., Lang., Apps.*, pages 432–448, Oct. 2004.

[29] G. Rothermel and M. J. Harrold. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.*, 22(8):529–551, Aug. 1996.

[30] G. Rothermel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Meth.*, 6(2):173–210, Apr. 1997.

[31] Shull, F. et al. What we have learned about fighting defects. In *Int'l. Softw. Metrics Symp.*, 2002.

[32] A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. In *Int'l. Symp. Softw. Test. Anal.*, pages 97–106, July 2002.

[33] F. I. Vokolos and P. G. Frankl. Empirical evaluation of the textual differencing regression testing technique. In *Int'l. Conf. Softw. Maint.*, pages 44–53, Nov. 1998.

[34] W. Wong, J. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Int'l. Symp. Softw. Rel. Engr.*, pages 230–238, Nov. 1997.