# Using Sensitivity Analysis to Create Simplified Economic Models for Regression Testing

Hyunsook Do
North Dakota State University
hyunsook.do@ndsu.edu

Gregg Rothermel
University of Nebraska - Lincoln
grother@cse.unl.edu

## ABSTRACT

Software engineering methodologies are subject to complex cost-benefit tradeoffs. Economic models can help practitioners and researchers assess methodologies relative to these tradeoffs. Effective economic models, however, can be established only through an iterative process of refinement involving analytical and empirical methods. Sensitivity analysis provides one such method. By identifying the factors that are most important to models, sensitivity analysis can help simplify those models; it can also identify factors that must be measured with care, leading to guidelines for better test strategy definition and application. In prior work we presented the first comprehensive economic model for the regression testing process, that captures both cost and benefit factors relevant to that process while supporting evaluation of these processes across entire system lifetimes. In this work we use sensitivity analysis to examine our model analytically and assess the factors that are most important to the model. Based on the results of that analysis, we propose two new models of increasing simplicity. We assess these models empirically on data obtained by using regression testing techniques on several non-trivial software systems. Our results show that one of the simplified models assesses the relationships between techniques in the same way as the full model.

## Categories and Subject Descriptors

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

## General Terms

Experimentation, Measurement, Verification, Economics

## Keywords

Regression testing, test case prioritization, regression test selection, economic models, sensitivity analysis, empirical studies

## 1. INTRODUCTION

Many software engineering methodologies face complex cost and benefit tradeoffs that are influenced by a wide variety of factors. For example, software development processes are influenced by system size, personnel experience, and requirements volatility [2], and these contribute to determining the overall costs and benefits of particular processes.

To assess methodologies and predict their costs and benefits, effective *economic models* are needed. Several such models have been created in relation to software engineering. For example, Boehm et al. [2] provide COCOMO-II for use in estimating software development costs, and Freimut et al. [10] provide a model for use in assessing the cost-effectiveness of inspection processes. While these models estimate costs incurred during the software development phase, other models consider issues related to post-development phases. For example, Ostrand et al. [34] provide a model for use in estimating the fault proneness of systems, and Wagner [36] provides a model for use in assessing the costs and benefits of defect detection techniques.

Economic models such as these cannot be created and assessed in a single step; rather, they must be refined and improved iteratively over time. Thus, for example, the COCOMO model has gone through several calibration and evaluation steps, and to assess its accuracy researchers used a database of 63 completed software projects over a period of 15 years [2]. Models must also be assessed, iteratively, across different systems in order to generalize them; thus, for example, Ostrand et al. [35] have applied and then adjusted their fault proneness model across several systems.

One fundamental question to ask when refining an economic model involves whether it can be simplified, because model simplification renders the process of collecting or estimating model data less expensive. A second question to ask involves a model's sources of uncertainty and imprecision, because in most non-trivial models some data (e.g., fault costs) must be estimated, and some input factors are associated with uncertainty (e.g., programmer salary and revenue with economic growth). Answers to this second question can help practitioners and researchers identify factors that require more careful measurement tools and estimation efforts, leading to guidelines for better test strategy definition and application.

There are many approaches available, both analytical and empirical, for refining and improving economic models. For example, principal component analysis [12] assesses the importance of particular factors in models, supporting model simplification. Sensitivity analysis [30] uses empirical data and statistical methods to investigate the dependence of model outputs on model inputs, and it facilitates the apportionment of uncertainty in outputs to sources of uncertainty in inputs, supporting both model simplification and input factor prioritization. Empirical studies formulate and test hypotheses that allow us to determine causal connections between factors and cost-benefits, and to assess the degree of agreement between model estimations and cost-benefits in practice.

One area of software engineering in which comprehensive economic models have been relatively scarce (see Section 2) involves


*ISSTA'08,* July 20–24, 2008, Seattle, Washington, USA.

software testing, and in particular, software regression testing. This scarcity is unfortunate, because comprehensive economic models of testing and regression testing processes would be useful to both practitioners and researchers, providing ways to predict the relative cost-benefits of particular techniques, and ways to assess techniques more accurately in empirical studies.

In earlier work we presented EVOMO, the first comprehensive economic model for assessing the costs and benefits of regression testing processes [4], that captures both cost and benefit factors relevant to those processes, while facilitating their evaluation across entire system lifecycles rather than on single releases. In this paper we use sensitivity analysis to refine EVOMO. We use the results of this analysis to identify the input factors that are most important to the model; this analysis allows us to generate two new models of increasing simplicity. We assess our simplified models empirically by applying them to data obtained from several software systems. Our results show that one of our simplified models can assess the relationships between regression testing techniques in the same way as the full model.

Ultimately, this paper makes two primary contributions. First, our simplified models are less expensive to utilize than our initial model, as they require attention to fewer metrics. Second, our analysis yields a way to prioritize the factors included in our models, helping researchers understand which factors merit study first, and helping practitioners understand which factors merit most careful measurement. Beyond these contributions, as an illustration of the process of using sensitivity analysis, this work may be useful to other researchers creating economic models for other analysis and testing processes and techniques.

The rest of this paper is organized as follows. Section 2 provides background information on regression testing, economic models, and sensitivity analysis. Section 3 describes our economic model. Section 4 describes the application of sensitivity analysis to our model, and presents our simplified models. Section 5 presents our empirical evaluation of the models. Section 6 discusses the implications of the analysis and empirical work and concludes.

# 2. BACKGROUND AND RELATED WORK

## 2.1 Regression Testing Practice and Research

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 [21], 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. [33] 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.

For this reason, researchers have studied various methods for reducing the cost and improving the effectiveness of regression testing. *Regression test selection techniques* (e.g., [22, 29], for a survey and additional references see [28]) attempt to reduce testing costs by selecting, from $T$, a subset of test cases that should be executed on $P'$. *Test case prioritization techniques* (e.g., [8, 33]) reorder test cases, scheduling test cases with the highest priority according to some criterion earlier in the testing process. Most selection and prioritization techniques operate in concert with some sort of program analysis activities; most prevalent are techniques that utilize code coverage information obtained through instrumentation of code and collection of test traces.

Numerous studies of test selection and prioritization have been performed (e.g., [8, 14, 27]), and these have revealed various factors responsible for the performance of specific techniques. In general, regression test selection techniques can reduce costs by reducing testing time, but they can lead to increased costs if they let faults slip through undetected into the post-testing period. Test case prioritization techniques can reduce costs by revealing faults early, allowing fault correction activities that occur in that period to be performed in parallel with testing rather than afterward, but they can increase costs if the analyses they depend on are inordinately expensive. Determining in advance what techniques are likely to be most cost-effective in a particular situation, however, is not simple, and even finding metrics capable of properly assessing the relative benefits of techniques can be difficult [7, 9].

## 2.2 Economic Models

There have been only a few, simple models for regression testing proposed to date. Leung and White [17] present a model that considers some of the cost factors (testing time, technique execution time) that affect the cost of regression testing, but their model does not consider benefits. Malishevsky et al. [18] extend this work with cost models for regression test selection and test case prioritization that incorporate benefits related to the omission of faults and to rate of fault detection. Harrold et al. [11] present a coverage-based predictive model of regression test selection effectiveness, but this predictive model focuses only on reduction in numbers of test cases. Do et al. [6] extend Leung and White and Malishevsky's models, for test case prioritization, to incorporate additional cost factors (analysis and technique execution time).

There has been some work on economic models for testing (as distinct from regression testing). Muller et al. [19] present an economic model for the return on investment of test-driven development (TDD) compared to conventional development, provide a cost-benefit analysis, and identify a break-even point at which TDD becomes beneficial over conventional development. As already noted, Wagner [36] proposes an analytical model of the economics of defect detection techniques that incorporates various cost factors and revenues, considering uncertainty and using sensitivity analysis to identify the factors most relevant for model simplification.

While economic models in the software testing and regression testing areas are not well established, in other software engineering areas models have been considered much more extensively. These include the models of Ostrand et al. [34] and Freimut et al. [10], already mentioned; Kusumoto et al. [16] also propose a model for assessing the cost-effectiveness of software inspection processes. There have also been many models created to address software process and development costs and benefits; Boehm et al. [2]'s COCOMO-II model, mentioned earlier, is probably the most well-known. Recent research in value-based software engineering has also sought to model various engineering activities; Return On Investment (ROI) models, which calculate the benefit return of a given investment [24], provide one such approach, supporting systematic, software business value analysis [1, 19].

## 2.3 Sensitivity Analysis

While sensitivity analysis has been applied in many areas in which complex modeling is required, including atmospheric chemistry, transport emission, and fish population dynamics [30], it has not yet seen widespread use in relation to software engineering. Nonetheless there has been some research that has utilized sensitivity analysis in software engineering contexts.

[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 $P$.

Wagner [36, 37] applies sensitivity analysis to his model of defect detection techniques and the COCOMO model to investigate which input factors are most important. Freimut et al. [10] apply sensitivity analysis to their models of software inspection processes. Musilek et al. [20] perform sensitivity analysis on COCOMO II to assess the effect of imprecise evaluations, since estimation models typically rely heavily on data obtained by subjective expert evaluations affected by imprecision and uncertainty. Rodrigues et al. [26] use sensitivity analysis to identify components and usage profiles that have the greatest impact on the reliability of software systems. Wakeland et al. [38] apply sensitivity analysis in evaluating changes to a software development process to capture a number of important aspects of the software development environment.

To date, of this work, only Wagner's has studied models related to testing, and none of this work has considered sensitivity analysis in relation to economic models of regression testing processes.

## 3. THE EVOMO ECONOMIC MODEL

In this work we utilize an economic model of regression testing that we introduced initially in [4]. Our economic model, EVOMO (EVOlution-aware economic MOdel for regression testing) captures costs and benefits of regression testing methodologies relative to particular regression testing processes, considering methodologies in light of their business value to organizations, in terms of the cost of applying the methodologies and how much revenue they help organizations obtain. The regression testing process that the model corresponds to is the common *batch* process in which system modifications are performed over a period of time (which might range from days to several weeks or months) and then regression testing is used (in succeeding days, weeks or months) to assess those modifications [21].

EVOMO involves two equations: one that captures costs related to the salaries of the engineers who perform regression testing (to translate time spent into monetary values), and one that captures revenue gains or losses related to changes in system release time (to translate time-to-release into monetary values). Significantly, the model accounts for costs and benefits across entire system lifetimes, rather than on snapshots (i.e. single releases) of those systems, through equations that calculate costs and benefits *across entire sequences of system releases*. The model also accounts for the use of incremental analysis techniques (e.g., reliance on previously collected data where possible rather than on complete recomputation of data), an improvement also facilitated by the consideration of system lifetimes.

The two equations that comprise EVOMO are as follows; terms and coefficients are described briefly in Table 1.

$$Cost = PS * \sum_{i=2}^{n}(CS(i) + CO_i(i) + CO_r(i) + b(i) * CV_i(i) + c(i) * CF(i)) \quad (1)$$

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

To illustrate the use of this formula, if an organization does not test their product at all before delivery, then they gain revenue by reducing all of the cost terms other than $CF$ in Equation 1 to zero, and all the cost terms of form $CX$ in Equation 2 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 2 is positive (negative), indicating an increase (decrease) in revenue. These revenue changes are coupled, however, with changes in costs captured in Equation 1 in determining whether techniques are cost-beneficial overall.

**Table 1: Terms and Coefficients**

| Term | Description |
|---|---|
| $S$ | software system |
| $i$ | index denoting a release $S_i$ of $S$ |
| $n$ | the number of releases of the software system |
| $u$ | unit of time (e.g., hours or days) |
| $CS(i)$ | time to setup for testing activities $S_i$ |
| $CO_i(i)$ | time to identify obsolete tests for $S_i$ |
| $CO_r(i)$ | time to repair obsolete tests for $S_i$ |
| $CA_{in}(i)$ | time to instrument all units in $i$ |
| $CA_{tr}(i)$ | time to collect traces for test cases in $S_{i-1}$ |
| $CR(i)$ | time to execute a technique itself on $S_i$ |
| $CE(i)$ | time to execute test cases on $S_i$ |
| $CV_d(i)$ | time to apply tools to check outputs of test cases run on $S_i$ |
| $CV_i(i)$ | (human) time for inspecting the results of test cases |
| $CF(i)$ | cost associated with missed faults after the delivery of $S_i$ |
| $CD(i)$ | cost associated with delayed fault detection feedback on $S_i$ |
| $REV$ | revenue in dollars per unit $u$ |
| $PS$ | average hourly programmer's salary in dollars per unit $u$ |
| $ED(i)$ | expected time-to-delivery in units $u$ for $S_i$ when testing begins |
| $a_{in}(i)$ | coefficient to capture reductions in costs of instrumentation for $S_i$ due to the use of incremental analysis techniques |
| $a_{tr}(i)$ | coefficient to capture reductions in costs of trace collection for $S_i$ due to the use of incremental analysis techniques |
| $b(i)$ | coefficient to capture reductions in costs of executing and validating test cases for $S_i$ due to the use of incremental analysis techniques |
| $c(i)$ | number of faults that are not detected by test suite on $S_i$ |

When comparing regression testing techniques $T1$ and $T2$ for relative cost-benefits, we use Equations 1 and 2 as follows:

$$(Benefit_{T1} - Cost_{T1}) - (Benefit_{T2} - Cost_{T2}) \quad (3)$$

with positive values indicating that $T1$ has greater value than $T2$, and negative values indicating that $T1$ has lesser value than $T2$.

To utilize EVOMO, the constituent costs captured by the model must be measured or estimated in a manner appropriate to the particular regression testing processes being utilized and the particular systems being tested. While various approaches could be used to measure constituent costs, in this work we measure costs as follows, a manner suitable to the object systems that we use in our empirical work with the model (described further in Section 4).

**Cost of test setup** ($CS$). This includes the costs of setting up the system for testing, compiling the version to be tested, and configuring test drivers and scripts.

**Cost of identifying obsolete test cases** ($CO_i$). This includes the costs of manual inspection of a version and its test cases, and determination, given modifications made to the system, of the test cases that must be modified for the next version.

**Cost of repairing obsolete test cases** ($CO_r$). This includes the costs of examining specifications, existing test cases, and test drivers, and those of observing the execution of suspect tests and drivers.

**Cost of supporting analysis** ($CA$). This includes the costs of instrumenting programs ($CA_{in}$) and collecting test traces ($CA_{tr}$).

**Cost of technique execution** ($CR$). This is the time required to execute a regression testing technique itself.

**Cost of test execution** ($CE$). This is the time required to execute test cases.

**Cost of test result validation (automatic via differencing)** ($CV_d$). This is the time required to run a differencing tool on test outputs as test cases are executed.

**Cost of test result validation (human via inspection)** ($CV_i$). This is the time needed by engineers to inspect test output comparisons.

**Number and cost of missing faults** ($c$ and $CF$). For any regression testing technique that could miss faults, we measure the number $c$ of faults missed. Determining the cost of missed faults is much more difficult. Given the factors that can contribute to, and the long-term nature of, this cost, we cannot obtain this measure directly. Instead, we rely on data provided in [32] to obtain estimates of the cost of faults, and given the size of our experiment objects we choose 1.5 person hours, which is attributed by [32] as the cost of "ordinary" faults, as the cost of localizing and correcting one fault.

**Cost of delayed fault detection feedback** ($CD$). For each regression testing technique, we measure the rate of fault detection exhibited by the test suite associated with that technique, applied to a program version. Then, following the approach of [18], we translate this rate into the cumulative cost (in time) of waiting for each fault to be exposed while executing test cases under a particular order, defined as *delays*.

**Revenue** ($REV$). We cannot measure revenue directly, so we estimate it by utilizing revenue values cited in a survey of software products ranging from \$116,000 to \$596,000 per employee.[2] Because our object programs are relatively small, we use the smallest revenue value cited, and an employee headcount of ten.

**Programmer salary** ($PS$). We cannot measure this metric directly so we estimate it. We rely on a figure of \$100 per person-hour, obtained by adjusting an amount cited in [13] by an appropriate cost of living factor.

**Expected time-to-delivery** ($ED$). Actual values for $ED$ cannot easily be obtained for the systems that we study. In our empirical work we rely on comparisons of techniques to a control suite using Equation 3; this approach cancels out the $ED$ values because these are the same for all cases considered.

**Other coefficients**. $a_{in}(i)$, $a_{tr}(i)$, and $b(i)$ capture, proportionally, reductions in cost due to the use of incremental analysis techniques; their values are captured directly from our empirical data.

In addition to the costs just described, there are other costs associated with regression testing that are not currently accounted for by EVOMO, such as the costs of initial test case development, initial automation, test tool development, test suite maintenance, management overhead, database usage, and developing new test cases. There are also ways in which the costs captured by EVOMO have been simplified; for example, expected-time-to-delivery does not account for factors such as the additional costs of failing to meet specific (e.g., contractual or sales-related) deadlines. EVOMO could be extended to incorporate other costs such as these.

# 4. SENSITIVITY ANALYSIS

There are various ways in which sensitivity analysis can be used; we focus on screening input factors to identify those that are most influential, in order to improve our model. To do this, we follow standard sensitivity analysis procedures as defined in [31]. The following subsections describe our application of the approach, our results, and the simplified models that result.

## 4.1 Sensitivity Analysis Approach

We used the *Simlab* sensitivity analysis package, which is designed for Monte Carlo methods,[3] to perform our analysis [31].

[2]http://www.culpepper.com

[3]Monte Carlo methods are one of the widely used sampling based sensitivity analyses.

**Table 2: Object Programs and Associated Data**

| Objects | Versions | Classes | Size (KLOCs) | Test Cases | Mutants |
|---|---|---|---|---|---|
| *ant* | 9 | 627 | 80.4 | 877 | 412 |
| *jmeter* | 6 | 389 | 43.4 | 78 | 386 |
| *xml-security* | 4 | 143 | 16.3 | 83 | 246 |
| *nanoxml* | 6 | 26 | 7.6 | 216 | 204 |
| *galileo* | 16 | 87 | 15.2 | 912 | 2494 |

Monte Carlo methods require us to specify an input distribution function for each input factor (each input variable in our economic model). There are various ways to obtain such input distribution functions, including the use of literature surveys, empirical data, or expert opinion. In this paper we utilize empirical data collected through the following procedure.

### *Data collection procedure*

We collected empirical data using sequences of versions of the Java systems described in Table 2: *ant*, *xml-security*, *jmeter*, *galileo*, and *nanoxml*. *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.

The table lists, for each of these objects, information on 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 first three objects have JUnit test suites, and the last two have TSL (Test Specification Language) suites [23]. These programs, versions, and test suites are all available as part of the SIR infrastructure, a public repository supporting experimentation [3].

We consider three test case prioritization approaches as shown in Table 3; an original order of test cases (*To*), corresponding to the use of the retest-all technique with no prioritization analysis performed and serving as a control, and two prioritization heuristics: "total coverage" and "additional coverage". *To* utilizes the order in which test cases are executed in the original testing scripts provided with the object programs, and thus, serves as one potential representative of "current practice". The former heuristic orders test cases in terms of the total numbers of basic blocks (maximal single-entry, single-exit sequences of statements) they cover in the programs tested, as measured through test traces. The latter heuristic adds a "feedback" mechanism, prioritizing test cases in terms of the numbers of additional (as-yet-uncovered) 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 prioritized.

**Table 3: Test Case Prioritization Techniques.**

| Label | Technique | Description |
|---|---|---|
| *To* | original | original ordering |
| *Tct* | total coverage | prioritize on coverage of blocks |
| *Tca* | additional coverage | prioritize on coverage of blocks not yet covered |

To capture costs and benefits related to regression test selection approaches we used two selective retesting approaches, which we applied to each of the three test prioritization approaches utilized. The first approach uses complete test suites, and thus is equivalent

to the retest-all technique. The second approach halts test execution halfway through the testing process (when half of the test cases in the ordered test suite have been executed). This approach allows us to model, in a controlled manner, a test selection process in which prioritization is used to rank test cases, such that when any intervening time constraints halt testing, the test cases that are most important (up to that point in time) will have been executed.

Because our economic model evaluates regression testing techniques relative to their fault-revealing capabilities, we required object programs containing regression faults, so we utilized mutation faults that came with our object programs, having been constructed for use in a prior study [5]. The numbers of mutation faults available for each of our object programs is shown in column six of Table 2. To obtain test coverage information required by techniques used in the experiment, we used the Sofya [15] analysis tool.

EVOMO evaluates the application of regression testing techniques across entire sequences of versions of programs, so we collected data relative to our programs' sequences of versions. To do this, we considered each object program in turn, and for each version of that program and each selected mutant group for that version, applied each prioritization technique and collected the appropriate values for cost variables. Then, for each object program, we obtained 30 *sequences of mutant groups* by randomly selecting a mutant group for each version. We then collected cost variables considering all orderings of test cases (original order and orders produced by prioritization heuristics) under both selective retesting approaches (all test cases or 50% of the test cases executed.) We used the collected cost variables to provide the data for input distributions for sensitivity analysis.[4]

### *Analysis Procedure*

Having collected empirical data, we examined the data to select distribution functions for input factors. We chose a triangular distribution, a choice that is recommended when empirical data does not show clear distribution patterns or when other evidence is not available [31]. Using our empirical data we obtained the input values necessary for generating sample data under this distribution: the minimum, maximum, and most likely values for each input factor. Table 4 presents this data.[5] All data except for coefficients ($a_{in}$, $a_{tr}$, $b$, and $c$), *REV* and *PS* are provided in seconds. For other terms' metrics see Table 1.

**Table 4: Data Utilized in Sensitivity Analysis**

| Input Factor | Min | Max | Most likely value |
|---|---|---|---|
| $CS$ | 720 | 3600 | 1730 |
| $CO_i$ | 2460 | 12300 | 5900 |
| $CO_r$ | 0 | 20200 | 4040 |
| $CA_{in}$ | 0 | 5440 | 2020 |
| $CA_{tr}$ | 0 | 11900 | 3530 |
| $CR$ | 0 | 11300 | 1980 |
| $CE$ | 114 | 12000 | 3350 |
| $CV$ | 329 | 6700 | 2070 |
| $CF$ | 5400 | 82300 | 43800 |
| $CD$ | 5 | 34600 | 4660 |
| $REV$ | 0.088 | 0.82 | 0.454 |
| $PS$ | 0.022 | 0.033 | 0.027 |
| $a_{in}$ | 0 | 1 | 0.41 |
| $a_{tr}$ | 0 | 1 | 0.78 |
| $b$ | 0 | 1 | 0.44 |
| $c$ | 0 | 28 | 5 |

[4]Complete data sets can be obtained from the first author.

[5]$CV_d$ and $CV_i$ have been merged into a single variable *CV*, because in our data $CV_d$ contributed less than 10 seconds on average.

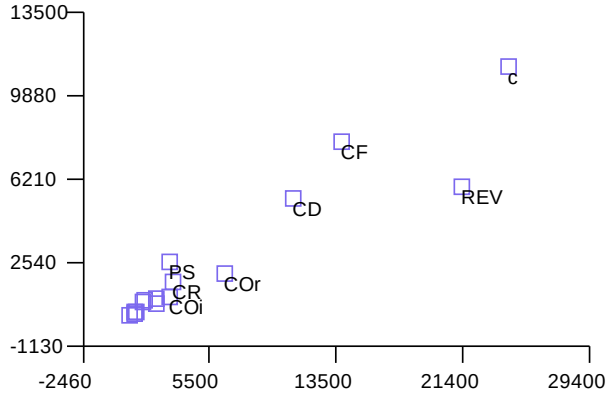


**Figure 1: Sensitivity analysis results (vertical axis: $\sigma$; horizontal axis: *u*)**

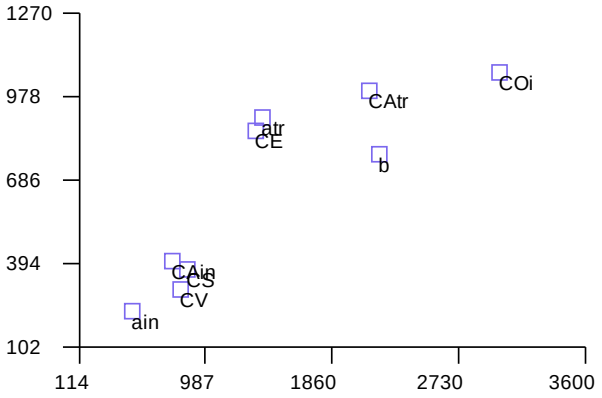


**Figure 2: Detailed view of the lower-left corner of Figure 1 (vertical axis: $\sigma$, horizontal axis: *u*)**

The next step in sensitivity analysis after defining input distributions is to choose a sample generation method – these are typically chosen based on the analysis goal [31]. Since our primary goal is the identification of important factors and model simplification, we selected the Morris method, a technique that ranks input factors in order of importance (details given below). We applied this method to our data using the *Simlab* package.

## 4.2 Sensitivity Analysis Results

Figures 1 and 2 show the result of our application of sensitivity analysis. The first figure plots all factors, and the second provides a more detailed view of the lower-left corner of the first. The Morris method calculates two sensitivity measures for each factor: a measure *u* that estimates the overall effect of the factor on the output, and a measure $\sigma$ that estimates standard deviation of the factor. Typically, *u* is used when the analysis goal is to rank factors in order of importance (it does not quantify the differences in importance). A smaller value of *u* implies that a factor is less influential in affecting output than a larger value. In the figures, the vertical axes indicate $\sigma$ values, and the horizontal axes indicate *u* values.

From Figure 1, we observe that input variables c (number of missed faults), *CF* (cost associated with missed faults), *REV* (revenue), and *CD* (cost associated with delayed fault detection feedback) have the most significant influence on the output, and are clearly separated from others. In particular, the number of missed faults has the strongest effect, which suggests that techniques that can reduce post release defects can make the most significant contributions to the reduction of overall regression testing costs.

Turning our attention to other factors and Figure 2, we can identify three separate groups of points in the figure: (1) $a_{in}$, *CV*, *CS*, and $CA_{in}$; (2) *CE*, $a_{tr}$, $CA_{tr}$, *b*; (3) $CO_i$. The first group has the smallest influence on the output, the second group has the second smallest influence, and so on. Factor $CO_i$ could be placed into the second group, but we separate it out because Figure 1 shows that $CO_i$ is closer to *PS* and *CR*. Reflecting on this grouping, the low influence of *CS* and *CV* on output is not surprising, because these costs tend to be relatively small compared to others.

## 4.3 Simplified Models

Given the foregoing classification of factors we proceed with model simplification by fixing unimportant factors at a given value over their range of uncertainty [31].

When fixing factors at given values, we need to select appropriate values. One common approach for doing this involves utilizing estimates of value ranges obtained from prior projects or expert knowledge. This approach may be the only one available when working with a new system, but it has the disadvantage of possible imprecision. When dealing with evolving systems, an alternative approach involves using values derived from earlier system releases. This incurs measurement costs initially but the costs of obtaining these values are then amortized over subsequent versions, on which the fixed factors need no longer be measured. We thus chose this alternative approach.

Based on our analysis results we consider three models, including our original EVOMO model and two refinements, as follows:

- *Model 1* (the original EVOMO model) considers all factors, as shown in Equations 1 and 2 in Section 3.
- *Model 2* fixes the four least significant factors, *CS*, *CV*, $CA_{in}$, and $a_{in}$. In the equations, we represent the value of these fixed factors collectively as constants $K_1$ and $K_2$.

$$Cost = PS*(\sum_{i=2}^{n}(CO_i(i)+CO_r(i)+c(i)*CF(i))+K_1) \quad (4)$$

$$Benefit = REV*(\sum_{i=2}^{n}(ED(i)-(CO_i(i)+CO_r(i) + a_{tr}(i-1)*CA_{tr}(i-1)+CR(i) + b(i)*CE(i)+CD(i)))-K_2) \quad (5)$$

- *Model 3* fixes the four additional factors, $b$, *CE*, $a_{tr}$, and $CA_{tr}$, in addition to those fixed for *Model 2*. We represent these together as constant $K_3$.

$$Cost = PS*(\sum_{i=2}^{n}(CO_i(i)+CO_r(i)+c(i)*CF(i))+K_1) \quad (6)$$

$$Benefit = REV*(\sum_{i=2}^{n}(ED(i)-(CO_i(i)+CO_r(i) + +CR(i)+CD(i)))-K_2-K_3) \quad (7)$$

# 5. MODEL EVALUATION

In the previous section, we obtained two simplified economic models by conducting sensitivity analysis. There are various questions that should be examined empirically about these models. Ultimately we would like to assess the accuracy of the model computations, and whether the models generalize over other systems.

Before addressing these questions, however, it is important to first assess whether the simplifications identified by the sensitivity analysis themselves are acceptable in terms of the impact they may have on model computations. Sensitivity analysis allows us to prioritize factors, but this does not tell us the degree of impact that the removal of particular factors may have on model output, and it does not tell us how far to proceed with such removal. Until we have examined the effects of the simplifications, we cannot assert that the simplified models we have created are appropriate.

Because our simplified models omit certain cost factors, we would not expect them to produce results that are numerically equivalent to those produced by the original model. However, if the simplified models preserve the same relationships between regression testing techniques as are captured by the full model, then they continue to support accurate conclusions about the relative cost-benefits of techniques. In this case, the simplified models are beneficial to researchers, who can focus on more important factors when they assess and create regression testing techniques, and to practitioners, who can avoid the costs of measuring omitted factors and still use the models to select techniques.

Thus, we investigate the research question: *Do our simplified models preserve the relative cost-benefit relationships between regression testing techniques that the initial model captures?*

To answer this question we performed a controlled experiment. The following subsections present our objects of analysis, independent and dependent variables, experiment setup and design, threats to validity, and data and analysis. Section 6 provides further discussion of the implication of the results.

## 5.1 Objects of Analysis

We use the five Java systems described in Section 4 (Table 2), together with their versions, tests, and faults, as objects of analysis.

## 5.2 Variables and Measures

### *Independent Variables*

Our experiment manipulated two independent variables: prioritization approach and selective retesting approach. As in Section 4, the three prioritization approaches are those listed in Table 3, and the two selective retesting approaches are those in which tests are run to completion (the retest-all technique), and in which time constraints halt testing prematurely (in our case, when testing is halfway complete). The latter choice provides a controlled approach to selective test re-execution that can be combined in a controlled manner with each of the priorization techniques considered.

### *Dependent Variable and Measures*

Our dependent variable is the cost-benefit value calculated by applying the models presented in Sections 3 and 4. Cost and benefit components are measured in dollars. The cost components include the constituent measures collected as described in Section 3. To compare techniques, however, we use a further calculation involving Equation 2. For this calculation, we use the original test order as a baseline, assuming that the cost-benefit value of using that order is zero for the given case.[6] Then, we determine the difference in cost-benefit values between the original test order and the techniques by treating the technique as $T1$ in Equation 2, and the original order as $T2$ in that equation. We apply the foregoing approach separately for the cases in which testing is run to completion and halted halfway. This approach removes the need to have values for $ED$ because the use of Equation 2 cancels that variable out.

[6]We use a mean value of the cost-benefit of the original test order for each version since each has 30 sequence values.

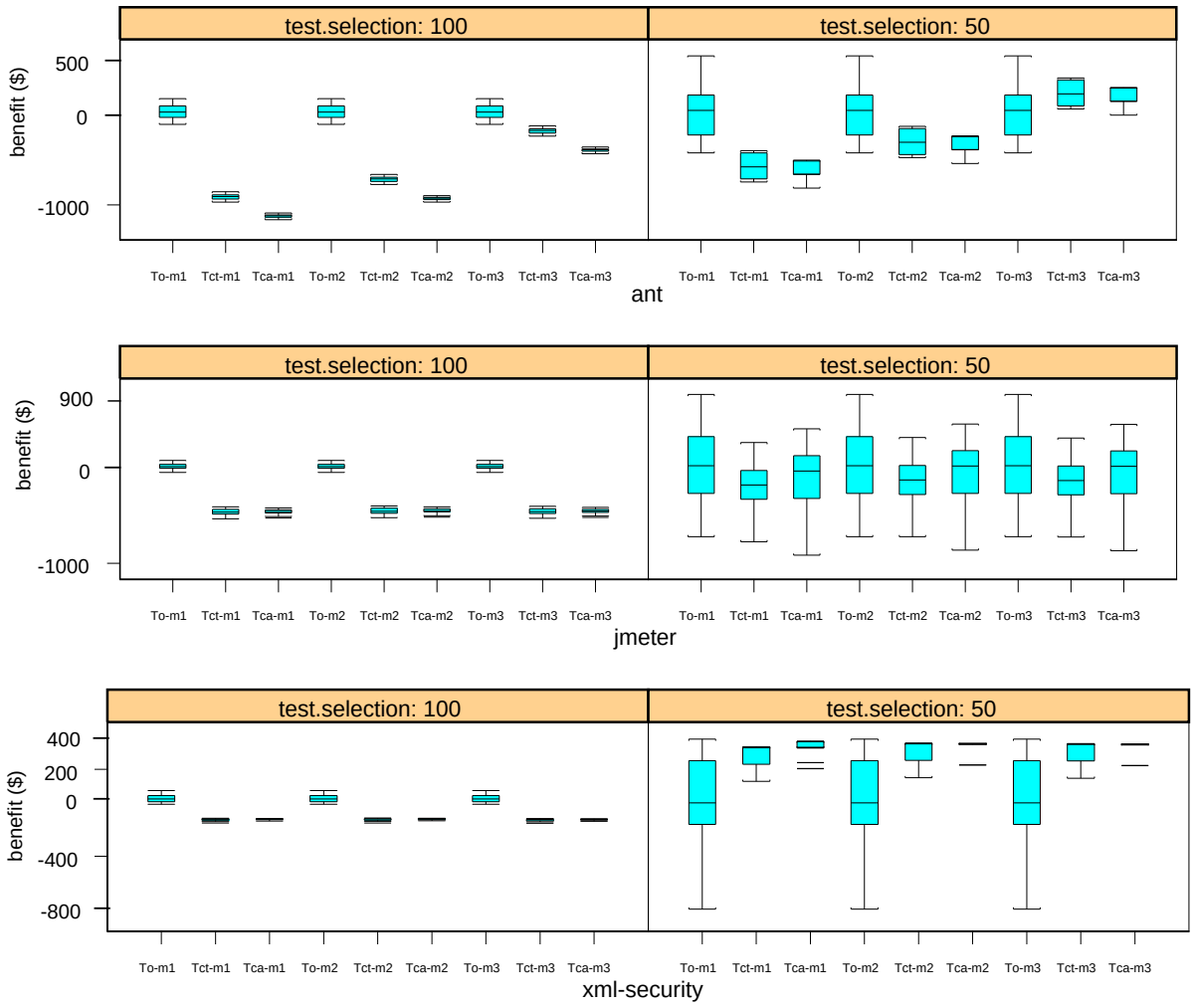


**Figure 3: Empirical data from each model for *ant*, *jmeter*, and *xml-security*.**

## 5.3 Experiment Setup

In Section 4, we described how we collected and estimated cost variables relative to our object programs. In this experiment, we utilize the same procedure. We then used the collected cost variables to calculate relative cost-benefit values for each of the three prioritization approaches applied to each of the five programs, for each of the two selective retesting approaches, and we repeated this for each of the three economic models. Each of these calculations required a calculation of the relative cost-benefit value for the given technique for each of the 30 sequences of mutant groups created for each given program. These relative cost-benefit values serve as the data for our subsequent analysis.[7]

## 5.4 Threats to Validity

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

**External Validity.** The Java programs that we study are relatively small (7KLoc - 80KLoc), and their test suites' execution times are relatively short. Complex industrial programs with different characteristics may be subject to different cost and benefit factors and tradeoffs. The testing process and the cost of faults we used are not representative of all processes and the fault severities seen in practice. On the other hand, our experiment objects are real, non-trivial systems, with the original test suites created for them by developers. Ultimately, further understanding of the extent to which our results on these objects generalize can be achieved only through additional studies with wider populations of programs and test suites, different testing processes, and wider ranges of faults.

**Internal Validity.** The inferences we draw could have been affected by several factors. One 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. A second factor involves the actual values we used to calculate costs, some of which involve estimations (e.g., fault costs, revenue, and programmer's salary). 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. A third factor involves the choice of factors for model simplifications. Since the Morris method provides qualitative sensitivity measures rather than measures that quantify how much more important a given factor is

[7]Complete data sets can be obtained from the first author.

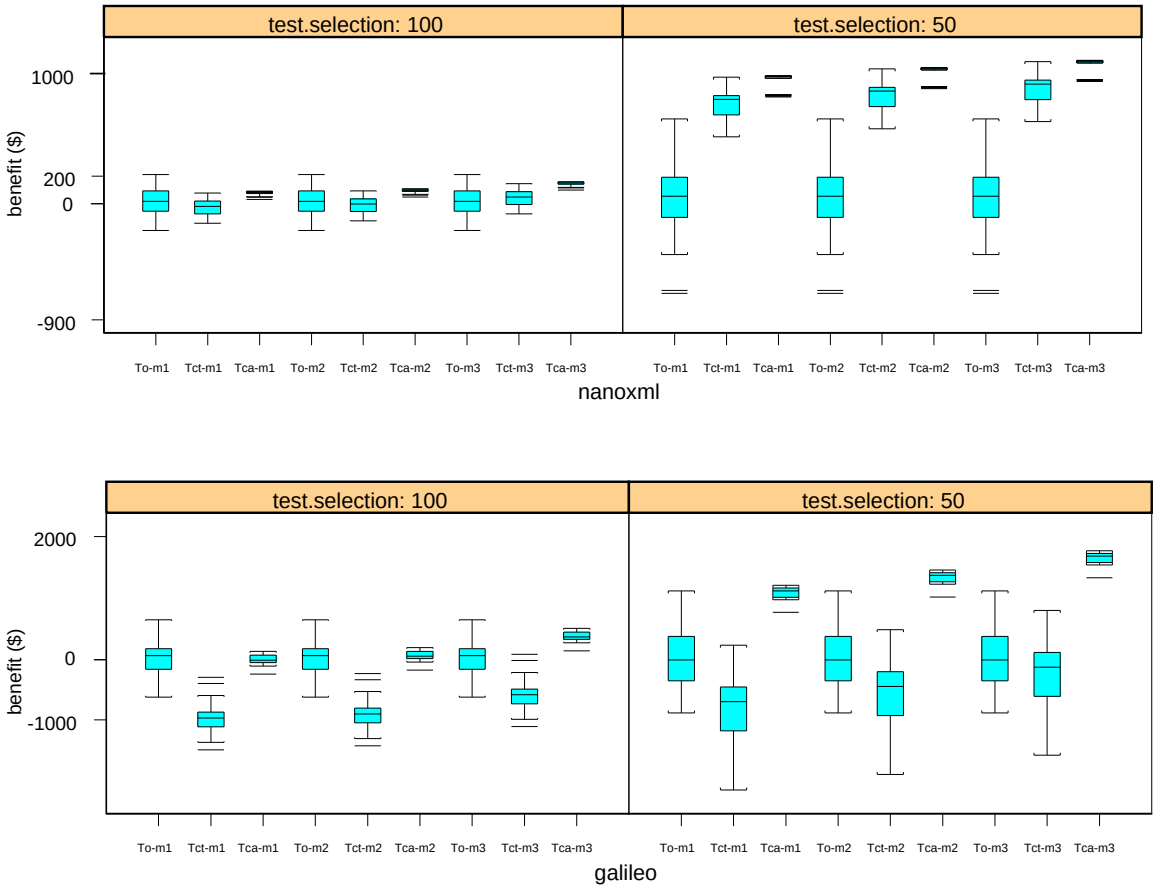


**Figure 4: Empirical data from each model for *nanoxml* and *galileo*.**

than others, our choice of factors to elide is somewhat subjective. In summary, we exercised care in selecting reasonable estimations relevant to our object programs, but larger-scale studies will be needed to follow up on these results.

**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. Other testing costs, such as initial test case development, initial automation costs, and test suite maintenance, might be worth measuring for different testing situations and organizations.

## 5.5 Data and Analysis

In this section, we describe and analyze our results. We discuss implications of these results in Section 6.

To provide an overview of the collected data, we present boxplots in Figures 3 and 4, which show relative cost-benefit results for the three programs (*ant*, *jmeter*, and *xml-security*) that have JUnit test suites, and the two programs (*nanoxml* and *galileo*) that have TSL test suites, respectively. In the figures, each row presents results for one object program. Each row contains two subfigures, presenting results for the two selective retesting approaches (complete test runs and runs halted halfway, labeled "test selection: 100%" and "test selection: 50%", respectively).

In each subfigure, the horizontal axis lists technique-model pairs, and the vertical axis lists relative cost-benefits in dollars. Higher values correspond to greater relative cost-benefit values. The first three boxplots present results obtained using *Model 1*, the next three boxplots present results using *Model 2*, and the last three boxplots present results using *Model 3* (as indicated by suffixes on technique-model labels appearing beneath the subfigure). Within each model's results, the three boxplots represent, for the given test suite orderings, the distribution of relative cost-benefits in dollars across the 30 sequences of mutant groups used on the versions of that object program. For example, the first boxplot (To-m1) in each subfigure presents data for the original test ordering under *Model 1*, and the next two boxplots (Tct-m1 and Tca-m1) present data for the two prioritization heuristics under *Model 1*.

We begin with descriptive analysis of the data in the boxplots, considering cost-benefit differences between techniques.

We first examine the boxplots for each object program in the subfigures on the left, corresponding to the case in which all test cases are executed. In Figure 3, for the three object programs that have JUnit test suites, we see that neither of the prioritization heuristics (*Tct* and *Tca*) appears to be cost-beneficial compared to the original ordering (*To*), under any of the models. For the two object programs in Figure 4, which have TSL test suites, the heuristics us-

ing feedback (*Tca*) appear to be slightly more cost-beneficial than the original ordering for all models. Techniques not using feedback (*Tct*) are more cost-beneficial than original orderings for Model 3 on *nanoxml*, but not in other cases.

The boxplots in the subfigures on the right, corresponding to the case in which test selection halts halfway through, display different results. In all cases but one (*ant*), at least one of the prioritization heuristics appears to produce test orders that are more cost-beneficial than the original ordering. In particular, in the cases of *xml-security* and *nanoxml*, both heuristics appear to be more cost-beneficial than the original order.

In other words, there are several cases, under each of the models, in which prioritization heuristics appear to improve the cost-effectiveness of regression testing, but the improvements are greater in the cases in which testing has been halted halfway through, where selective retesting has occurred.

Moving on to our research question, we next consider the relationships between techniques as our model is simplified. There are two ways to consider these relationships. The first approach considers whether the *relative ranking* of techniques (which performs best, which second best, which third best) remains the same across models. However, relative rankings of techniques can remain the same even though technique benefits change, so the second approach considers whether *assessed benefit* (whether an heuristic outperforms the original order for a given type of test selection) varies across models.

Considering the results descriptively, under both models it appears that on all programs other than *ant*, the relative ranking of techniques remains the same. In the case of *ant*, relative rankings appear the same in all cases except for Model 3 under selective retesting; this model, unlike Models 1 and 2, ranks both heuristics above the original order. Viewing the results in terms of assessed benefit, the prior observation indicates that *ant* is problematic, and also, on *nanoxml* and *galileo* there are cases in which, under simplified models, heuristics seem to be assessed as beneficial where they were not under Model 1.

## *Statistical Analysis*

The foregoing observations prompt us to assess the differences between models statistically. Since results vary substantially across object programs, we analyzed the data for each object separately. For our statistical analyses, we used the Kruskal-Wallis nonparametric one-way analysis of variance followed by Bonferroni's test for multiple comparisons [25]. We used the Kruskal-Wallis test because our data did not meet the assumptions necessary for using ANOVA: our data sets do not have equal variance, and some data sets have severe outliers. For multiple comparisons, we used the Bonferroni method for its conservatism and generality. We used the Splus statistics package[8] to perform the analyses.

Table 5 presents the results of the Kruskal-Wallis tests (df = 2), for a significance level of 0.05, and Table 6 presents the results of the Bonferroni tests using a Bonferroni correction [25]. In Table 6, techniques are listed within each program and each model by their mean relative cost-benefit values, from higher (better) to lower (worse). We use group letters (columns with headers "Grp") to partition the techniques such that those that are not significantly different share the same group letter. For example, in the top half of the table, the results for *jmeter* on Model 1 show that (1) *To* and *Tca* are statistically significantly different (they have different group letters; *To* has A and *Tca* has B), but (2) *Tca* and *Tct* are not statistically significantly different (they share group letter B).

[8]http://www.insightful.com/products/splus

**Table 5: Kruskal-Wallis Test Results, per Program**

| test selection: 100% | | | | | | |
|---|---|---|---|---|---|---|
| Program | $Model1$ | | $Model2$ | | $Model3$ | |
| | chi. | p-val. | chi. | p-val. | chi. | p-val. |
| *ant* | 79 | $< 0.0001$ | 79 | $< 0.0001$ | 79 | $< 0.0001$ |
| *jmeter* | 59 | $< 0.0001$ | 59 | $< 0.0001$ | 59 | $< 0.0001$ |
| *xml-security* | 59 | $< 0.0001$ | 59 | $< 0.0001$ | 59 | $< 0.0001$ |
| *nanoxml* | 33 | $< 0.0001$ | 37 | $< 0.0001$ | 45 | $< 0.0001$ |
| *galileo* | 58 | $< 0.0001$ | 57 | $< 0.0001$ | 65 | $< 0.0001$ |
| test selection: 50% | | | | | | |
| Program | $Model1$ | | $Model2$ | | $Model3$ | |
| | chi. | p-val. | chi. | p-val. | chi. | p-val. |
| *ant* | 58 | $< 0.001$ | 30 | $< 0.001$ | 10 | 0.005 |
| *jmeter* | 12 | 0.002 | 4 | 0.13 | 4.5 | 0.1 |
| *xml-security* | 25 | $< 0.001$ | 31 | $< 0.001$ | 25 | $< 0.001$ |
| *nanoxml* | 74 | $< 0.001$ | 74 | $< 0.001$ | 74 | $< 0.001$ |
| *galileo* | 204 | $< 0.001$ | 204 | $< 0.001$ | 204 | $< 0.001$ |

**Table 6: Bonferroni Test Results, per Program**

| test selectin: 100% | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Pr. | $Model1$ | | | $Model2$ | | | $Model3$ | | |
| | Tech. | Mean | Grp | Tech. | Mean | Grp | Tech. | Mean | Grp |
| *ant* | To | 0 | A | To | 0 | A | To | 0 | A |
| | Tct | -912 | B | Tct | -725 | B | Tct | -206 | B |
| | Tca | -1117 | C | Tca | -930 | C | Tca | -412 | C |
| *jm.* | To | 0 | A | To | 0 | A | To | 0 | A |
| | Tca | -467 | B | Tca | -457 | B | Tca | -461 | B |
| | Tct | -470 | B | Tct | -460 | B | Tct | -465 | B |
| *xml.* | To | 0 | A | To | 0 | A | To | 0 | A |
| | Tca | -145 | B | Tca | -143 | B | Tca | -147 | B |
| | Tct | -146 | B | Tct | -145 | B | Tct | -148 | B |
| *na.* | Tca | 70 | A | Tca | 87 | A | Tca | 142 | A |
| | To | 0 | B | To | 0 | B | Tct | 33 | B |
| | Tct | -38 | B | Tct | -20 | B | To | 0 | B |
| *ga.* | To | 0 | A | Tca | 50 | A | Tca | 365 | A |
| | Tca | -14 | A | To | 0 | A | To | 0 | B |
| | Tct | -965 | B | Tct | -901 | B | Tct | -586 | C |
| test selection: 50% | | | | | | | | | |
| Pr. | $Model1$ | | | $Model2$ | | | $Model3$ | | |
| | Tech. | Mean | Grp | Tech. | Mean | Grp | Tech. | Mean | Grp |
| *ant* | To | 0 | A | To | 0 | A | Tct | 193 | A |
| | Tct | -585 | B | Tct | -324 | B | Tca | 193 | A |
| | Tca | -624 | B | Tca | -363 | B | To | 0 | B |
| *jm.* | To | 0 | A | To | 0 | A | To | 0 | A |
| | Tca | -98 | A | Tca | -47 | A | Tca | -141 | A |
| | Tct | -188 | A B | Tct | -136 | A | Tct | -51 | A |
| *xml.* | Tca | 345 | A | Tca | 364 | A | Tca | 361 | A |
| | Tct | 289 | A | Tct | 314 | A | Tct | 311 | A |
| | To | 0 | B | To | 0 | B | To | 0 | B |
| *na.* | Tca | 929 | A | Tca | 992 | A | Tca | 1046 | A |
| | Tct | 755 | B | Tct | 818 | B | Tct | 873 | B |
| | To | 0 | C | To | 0 | C | To | 0 | C |
| *ga.* | Tca | 1078 | A | Tca | 1330 | A | Tca | 1645 | A |
| | To | 0 | B | To | 0 | B | To | 0 | B |
| | Tct | -758 | C | Tct | -506 | C | Tct | -191 | B |

For the comparison between techniques, results of the Kruskal-Wallis tests indicate that in all but two cases (*jmeter* when test runs are halted halfway) there are significant differences between techniques. However, this does not necessarily mean that the relationships between techniques are preserved as the model changes. To determine whether simplified models preserve the relationships that the full model captures we examine the Bonferroni results. As we can observe from Table 6, the results from multiple pair-wise comparisons (Bonferroni test) show slightly different trends between techniques among object programs, so we discuss each in turn.

In the case of *ant*, as we observed from the boxplots, relative ranking and assessed benefit remain the same across all models when all test cases are executed. When testing is halted halfway, Models 1 and 2 remain the same in both respects as well. In this case, however, Models 2 and 3 differ both in relative ranking (comparing heuristics to original) and with respect to assessed benefit (heuristics become beneficial under Model 3).

In the case of *jmeter*, similar to the results for *ant*, relative ranking and assessed benefit remain the same across all models when all test cases are executed. When testing is halted halfway, though, under Model 1, relative rankings between *To* and *Tct* change (in terms of statistical significance).

In the case of *xml-security*, both relative ranking and assessed benefit remain the same across all models.

In the case of *nanoxml*, both relative ranking and assessed benefit remain the same for Models 1 and 2. Models 2 and 3 differ in relative ranking and assessed benefit, however, in the case where all tests are executed (on *To* and *Tct*). The difference, however, is not statistically significant.

In the case of *galileo*, both relative ranking and assessed benefit vary for Models 1 and 2 when all test cases are executed (on *To* and *Tca*), although the differences are not statistically different. Also, relative ranking varies for Models 2 and 3 in both selection cases (on *To* and *Tct*) in terms of statistical differences.

# 6. DISCUSSION AND CONCLUSIONS

Through sensitivity analysis, we obtained two simplified models from our original full model, and empirically evaluated whether these simplified models possessed the same ability as the original to assess cost-benefit relationships between regression testing techniques. To further explore the implications of our results we consider two issues: (1) an analysis of the model simplification results; (2) a discussion of the practical implications of the results.

Where issue (1) is concerned, our empirical evaluation suggests that if the results of our studies generalize, our first simplified model (Model 2) can typically be expected to assess the relationship between techniques in the same way as the full model. We observed only two exceptions to this, as detailed in the preceding section: (1) on *jmeter* when test runs are halted halfway, the relationship between *To* and *Tct* changes in terms of the statistical significance of the difference between them, but the simplified model continues to rank them in the same order; (2) on *galileo* with complete test runs, the rankings of *To* and *Tct* are swapped between the two models, but the statistical relationship between these two techniques is maintained. Only in case (2) does Model 2 find a technique beneficial that was not found so by Model 1 (the mean benefit of *Tca* increases from -14 to 50); however this difference is small and its effect could be offset by providing margin-of-error estimates along with reported comparisons.

In the case of Model 3, in contrast, we observed quite a few cases (detailed in the previous section) in which the model fails to preserve the relationships between techniques as captured by Model 1 (or Model 2). Having fixed the four additional factors $b$, *CE*, $a_{tr}$, and $CA_{tr}$ at particular values, the model evaluated techniques differently in several cases. One plausible reason for this involves the particular fixed values selected for these four factors. We obtained these values from data gathered on the initial version of each program, and then we used these values across the rest of the versions of the system. This is potentially a practical approach for obtaining such values, but if the values of these variables on the initial version of the program are quite different from their values on later versions of the system (e.g., significantly larger as the system evolves), then the results obtained using the initial values in Model 3 can lead to greater differences in its computations. The two programs that show the greatest inconsistency in model comparison results relative to Model 3 are *ant* and *galileo*, and further examination of cost factor values on the versions of these systems as the systems evolve suggests that they fit the trend just described.

Given the foregoing, we return to one of our primary initial questions: how far we can go in simplifying our model. The extent of the differences we observed between Models 1 and 2 is relatively small, suggesting that Model 2 is likely to be a sufficiently reliable substitute. Simplifying to Model 3, however, appears risky because the model seems overly prone to loosing accuracy in terms of the relationships it captures. Nevertheless, it is a bit early to draw an absolute conclusion about Model 3; for one thing, a more accurate approach to fixing factors might improve the model's accuracy. Furthermore, the model might retain sufficient accuracy in certain situations. For example, if analysis costs have only a small effect on overall regression testing costs, as they do in the cases of *xml-security* and *nanoxml*, the simplifications made to Model 3 do not affect its evaluation of relationships between techniques. Further understanding of these issues requires additional empirical evaluation utilizing a wider range of programs, including complex industrial programs.

Regarding issue (2), our results have several practical implications. One type of implication relates to our motivation for model simplification. Simplified models provide many benefits to researchers and practitioners; they reduce the effort required for data collection and they simplify the empirical evaluation process. Our simplification of our original model should yield both of these benefits.

A second type of implication relates to another use of sensitivity analysis involving prioritization of input factors. The results of our sensitivity analysis show that four input factors (the number of missed faults, the cost associated with missed faults, revenue, and the cost associated with delayed fault detection feedback) have the most significant influence on overall regression testing costs. This result is useful in several ways. First, this result identifies, for both practitioners and researchers, which factors merit the most careful measurement when utilizing or empirically studying regression testing techniques. Second, this result can lead to guidelines for organizations in making decisions regarding resource allocations. For example, organizations can make decisions about where to spend additional effort in regression testing by examining the identified important factors. Finally, and perhaps of greatest usefulness to researchers interested in regression testing methodologies, this result identifies several issues worth considering when attempting to create or improve such methodologies. For example, incremental analysis techniques do not seem to be the best choice for initial attention; rather, a focus on faults escaping testing and the costs of residual faults (faults that escape testing and then persist through multiple releases) would have a greater chance of leading to techniques that might have high impact in terms of improving regression testing cost-effectiveness.

Our results suggest several avenues for future work. First, although the economic 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. Second, in the study reported in this paper, we evaluated regression testing techniques and refined EVOMO using small to medium size systems and relatively small revenue estimates. While the systems and tests utilized are actual, practical systems drawn from the field, more complex industrial systems will yield higher revenues than those considered here, and might involve higher costs associated with post-release defects. We expect that in such cases, the important factors we have identified through sensitivity analysis will have an even greater impact on the relative cost-benefits of regression testing techniques. We will be seeking ways to leverage this through techniques that focus on these factors. Finally, the most promising prospect suggested by our results is the potential for model simplification without losing the essential behavior of the model. We expect to apply this approach to assess and improve our models iteratively across different systems, as those models evolve.

## Acknowledgements

This work was supported in part by NSF under Award CNS-0454203 to the University of Nebraska - Lincoln.

## 7. REFERENCES

[1] B. Boehm. Value-based software engineering. *Softw. Eng. Notes*, 28(2), Mar. 2003.

[2] B. Boehm, C. Abts, A. W. Brown, S. Chulani, E. H. B. K. Clark, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Prentice-Hall, 2000.

[3] 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.

[4] H. Do and G. Rothermel. An empirical study of regression testing techniques incorporating context and lifetime factors and improved cost-benefit models. In *Int'l. Symp. Found. Softw. Eng.*, pages 141–151, Nov. 2006.

[5] H. Do and G. Rothermel. On the use of mutation faults in empirical assessments of test case prioritization techniques. *IEEE Trans. Softw. Eng.*, 32(9):733–752, 2006.

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

[7] S. Elbaum, D. Gable, and G. Rothermel. Understanding and measuring the sources of variation in the prioritization of regression test suites. In *Int'l. Softw. Metrics Symp.*, pages 169–179, Apr. 2001.

[8] 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.

[9] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky. Selecting a cost-effective test case prioritization technique. *Softw. Qual. J.*, 12(3):185–210, Sept. 2004.

[10] B. Freimut, L. C. Briand, and F. Vollei. Determining inspection cost-effectiveness by combining project data and expert opinion. *IEEE Trans. Softw. Eng.*, 31(12):1074–1092, Dec. 2005.

[11] 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.

[12] R. Johnson and D. Wichern. *Applied Multivariate Statistical Analysis*. Prentice Hall, 1992.

[13] C. Jones. *Applied Software Measurement: Assuring Productivity and Quality*. McGraw-Hill, 1997.

[14] 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.

[15] 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.

[16] S. Kusumoto, K. Matsumoto, T. Kikuno, and K. Torii. A new metric for cost-effectiveness of software reviews. *IEICE Trans. Info. Sys.*, E75-D(5):674–680, 1992.

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

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

[19] M. Muller and F. Padberg. About the return on investment of test-driven development. In *Int'l. W. Econ.-Driven Softw. Eng. Res.*, May 2003.

[20] P. Musilek, W. Pedrycz, N. Sun, and G. Succi. On the sensitivity of COCOMO II software cost estimation model. In *Int'l. Symp. Softw. Metrics*, 2002.

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

[22] A. Orso, N. Shi, and M. J. Harrold. Scaling regression testing to large software systems. In *Int'l. Symp. Found. Softw. Eng.*, pages 241–251, Nov. 2004.

[23] T. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Comm. ACM*, 31(6):676–688, June 1988.

[24] J. J. Phillips. *Return on Investment in Training and Performance Improvement Programs*. Gulf Publishing Company, Houston, TX, 1997.

[25] F. L. Ramsey and D. W. Schafer. *The Statistical Sleuth*. Duxbury Press, 1st edition, 1997.

[26] G. Rodrigues, D. Rosenblum, and S. Uchitel. Sensitivity analysis for a scenario-based reliability prediction model. In *W. Arch. Dep. Sys.*, 2005.

[27] G. Rothermel, S. Elbaum, P. Kallakuri, X. Qiu, and A. G. Malishevsky. On test suite composition and cost-effective regression testing. *ACM Trans. Softw. Eng. Meth.*, 13(3):277–331, July 2004.

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

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

[30] A. Saltelli. Sensitivity analysis for important assessment. *Risk Analysis*, 22(3):579–590, 2002.

[31] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice*. Wiley, 2004.

[32] F. Shull, V. Basili, B. Boehm, A. W. Brown, P. Costa, M. Lindvall, D. Port, I. Rus, R. Tesoriero, and M. Zelkowitz. What we have learned about fighting defects. In *Int'l. Softw. Metrics Symp.*, pages 249–258, June 2002.

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

[34] T. Ostrand, E. Weyuker, and R. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Softw. Eng.*, 31(4):340–355, Apr. 2005.

[35] T. Ostrand, E. Weyuker, and R. Bell. Automating algorithms for the identification of fault-prone files. In *Int'l. Symp. Softw. Test. Rel.*, pages 219–227, July 2007.

[36] S. Wagner. A model and sensitivity analysis of the quality economics of defect-detection techniques. In *Int'l. Symp. Softw. Test. Anal.*, pages 73–84, July 2006.

[37] S. Wagner. An approach to global sensitivity analysis: FAST on COCOMO. In *Int'l. Symp. Emp. Softw. Eng. Measurement*, pages 440–442, Sept. 2007.

[38] W. Wakeland, R. Martin, and D. Raffo. Using design of experiments, sensitivity analysis, and hybrid simulation to evaluate changes to a software development process: A case study. In *Int'l. W. Softw. Process Sim. Model.*, 2003.