# Trade-off Analysis for Selective versus Brute-Force Regression Testing in FSMWeb

Anneliese Andrews
Department of Computer Science
University of Denver
Denver, CO 80208
andrews@cs.du.edu

Hyunsook Do
Department of Computer Science
North Dakota State University
Fargo, ND 58102
hyunsook.do@ndsu.edu

***Abstract*—This paper provides a tradeoff framework to determine whether selective regression testing or brute force regression testing is preferable. The framework is based on classifying tests as reusable, retestable, and obsolete. It uses the FSMWeb model as the behavioral model upon which test generation is based. To illustrate how our tradeoff framework works for evaluating and choosing regression testing approaches for FSMWeb models, we use an example that has different modification scenarios in the FSMWeb model and analyze cost-benefit tradeoffs for the techniques that vary with the amount of changes and the impact of model changes. Further, we discuss practical issues to be considered when we choose regression testing techniques so that the employed technique meets the organization's business values.**

***Index Terms*—regression testing, web applications, trade-off analysis**

## I. Introduction

Over the past decades, web applications have been widely used because web applications offer easy access for users and diverse services. Typically, web applications go through more frequent updates compared to traditional desktop applications due to various reasons, such as continual feature upgrade demands from users or security problems. This necessitates more frequent regression testing to ensure the quality of the modified applications. However, regression testing is costly, and in particular for large web applications, applying regression testing to the entire application with all existing test cases can be prohibitively expensive [1]. Further, organizations could lose their business opportunities or customers if they fail to supply new releases in a timely manner due to a lengthy regression testing process. (Note that a recent study [2] reports that small/medium-sized organizations lost, on average, $70,000 per downtime hour.)

To resolve this issue, we can employ different regression testing approaches, such as selective regression testing, or prioritizing test cases that are more important to run early. However, it is difficult to know in advance which techniques would be more cost-effective than others. To date, many empirical studies on regression testing [3], [4], [5] have evaluated various regression testing techniques in terms of their effectiveness and efficiency, but often, the studies used simple measures of testing effort, such as the number of test cases or the time required for testing. Some studies have evaluated the cost-benefit tradeoffs for techniques using explicit cost models [6], [7], [8]. Although these models provide more proper ways to evaluate the techniques, they allow evaluation only after applying the techniques to the applications rather than choosing the proper techniques beforehand.

In this paper, we propose a tradeoff analysis framework to determine a proper regression testing technique by considering various cost factors that are associated with software artifact analysis and technique application. The framework is based on classifying tests as reusable, retestable, and obsolete. It uses the FSMWeb model as the behavioral model upon which test generation is based. The FSMWeb model is used to model and generate tests for web applications. We apply the proposed tradeoff framework to an example that contains different modification scenarios in the FSMWeb model and analyze the cost-benefit tradeoffs for the two regression testing techniques (brute force and selective techniques), which vary with the amount of change and the impact of model changes. Further, we discuss the practical issues for choosing a regression testing technique considering various factors that can be affected by an organization's business values.

The next section describes the FSMWeb method. Sections III, IV, and V present details about the tradeoff framework, including practical considerations of cost-benefit tradeoffs for selecting regression testing techniques. Section VI describes the related work and Section VII presents conclusions and future work.

## II. FSMWeb

### A. FSMWeb Approach

Functional testing for web application follows the approach in [9]:

1) Build a hierarchical model *HFSM*:
   a) Partition the web application into clusters (*C*s).
   b) Define Logical Web Pages (*LWP*s) and Input-Action constraints for each.
   c) Build FSMs for clusters as a multi-level hierarchy.
   d) Build an Aggregate FSM (*AFSM*) to represent the top level of the application.
2) Generate tests from the *HFSM*.
   a) Generate paths through each FSM that meet the coverage criteria.

b) Aggregate paths to form abstract tests.
c) Choose inputs to create executable tests.

The *cluster* is used to refer to collections of software modules/web pages that implement a logical, user level function. The first step partitions the web application into clusters. At the highest level of abstraction, clusters represent functions that can be identified by users. At a lower level, clusters represent cohesive software modules/web pages that work together to implement a portion of a user level function.

Many web pages contain HTML forms, each of which can be connected to a different back-end software module. To facilitate testing for these modules, web pages are modeled as multiple *Logical Web Pages* (LWPs). A LWP is either a physical web page or the portion of a web page that accepts data from the user through an HTML form and then sends the data to a specific software module. LWPs are abstracted from the presentation defined by the HTML and are described in terms of their sets of *inputs* and *actions*. All inputs in a LWP are considered atomic: data entered into a text field is considered to be only one user input symbol, regardless of how many characters are entered into the field.

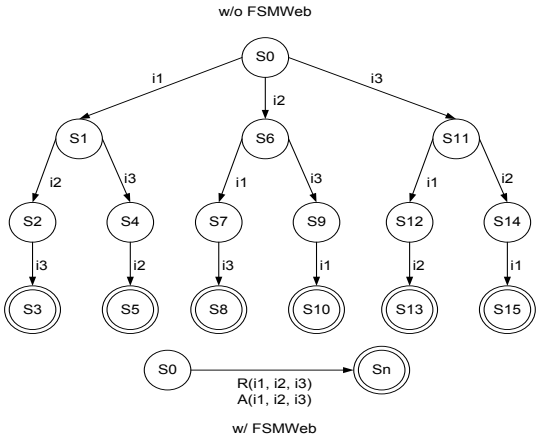


Fig. 1. Three Optional Inputs, Any Order

The lowest level cluster FSMs are generated with only LWPs and navigation between them. Input-action constraints annotate each edge [9]. Higher level FSMs represent FSMs from a lower level cluster by a single node and may contain LWP nodes as well. Ultimately, a top level Aggregate Finite State Machine (AFSM) is formed and represents a finite state model of the application at the highest level of abstraction.

Test sequences are generated during phase 2 of the FSMWeb method. A test sequence is a sequence of transitions through the application FSM and through each lower level FSM. FSMWeb's test generation method first generates paths through each FSM based on some graph coverage criterion such as *edge coverage*. These paths are then aggregated based on an aggregation criterion for each FSM's paths, such as *all combinations* or *each path at least once* [9]. This process results in a set of aggregate paths. We call them *abstract tests*. The final step of test generation is selecting inputs to replace the input constraints for the transitions of the aggregate paths.

Hence, an $HFSM = \{FSM_i\}_{i=0}^{n}$ with a top level $FSM_0 = AFSM$. Each FSM has nodes that represent LWPs or clusters. Edges are internal or external to an FSM. External nodes span cluster boundaries. Figure 1 shows how an FSM model that represents a selection with only three choices is reduced to two nodes and one transition in FSMWeb.

## III. Approach

### A. Classification of Tests after Model Changes

In [10], rules classifying of obsolete, retestable, and reusable test cases are defined. Together with the existing regression testing framework [11], [12] Andrews et al. [10] form the basis for selective regression testing for FSMWeb. We summarize the approach briefly because it is the basis upon which we built the tradeoff analysis:

$T_0$ is the original test suite; $T_{rt}$ is the retestable test set; $T_{ob}$ is the obsolete test set; $T_{ru}$ is the reusable test set.

In FSMWeb, test cases are generated in stages. We have to determine the level of abstraction at which *HFSM* changes are mapped to regression test case classification. Section II introduced an *abstract test case*, $t_a$, as a test path through the *HFSM* that represents an aggregated path.

There is a one to many relationship between abstract test case $t_a$ and test case $t$. (The input conditions may be satisfied with more than one set of values.) Hence, a path change (either due to changes in a node or an edge) may affect one or more test cases. If a change in the *HFSM* FSMWeb model affects an abstract test case, $t_a$, it also affects the set of executable test cases, $T_a$, associated with it.

Changes to nodes and edges are classified as to their impact on existing test cases. This determines whether tests are obsolete, retestable, or reusable:

1) Classify types of changes:
   a) node change: delete or modify (i.e., change the node type from LWP to cluster or vice versa). Node addition is covered by edge changes.
   b) edge change: modify in/out edge (i.e., modify the edge's input-action constraint)[1], delete in/out edge, add in/out edge with or without a new node
   c) classify model changes as path affecting (PC) or non-path affecting (NC).
2) Classify abstract test cases as obsolete, reusable, or retestable. Map this classification to the associated tests.

Table I shows the notations and conventions used in [10]. We also assume that any changes to the *HFSM* transformed it into a valid $HFSM'$ (e.g., no unreachable nodes or dangling edges). For details, see [9]. The test classification rules are as follows.

**Obsolete Test Cases**

They are caused by node or edge deletions and modifications. Let $N_o = \{n | n \in N; n \text{ is deleted or modified}\}$; then, the set

[1]Modifying the source or target node of an edge, that is, redirecting the edge, is covered by edge deletion and subsequent edge addition.

TABLE I
NOTATIONS AND CONVENTIONS

| | |
|---|---|
| $e$ | an edge in one of the FSMs in an *HFSM*. $E = \{e\}$ is the set of all such edges. |
| $n$ | a node in one of the FSMs in an *HFSM*. $N = \{n\}$ is the set of all such nodes. |
| $e'$ | an edge in one of the FSMs of $HFSM'$ (created through modifications to *HFSM*). $E'$ are all such edges. |
| $n'$ | a node in one of the FSMs of $HFSM'$ (created through modifications to *HFSM*). $N'$ are all such nodes. |
| $n_{source}$, $n_{sink}$ | source and sink nodes of the AFSM; i. e. nodes in the AFSM that have either no incoming or no outgoing edges. |
| $\hat{e}$ | an edge in one of the FSMs in *HFSM* that is changed. |
| $\hat{n}$ | a node in one of the FSMs in *HFSM* that is changed. |
| $t_a$ *tours* $e = (n_i, n_j)$ | there is a subsequence $n_i; I_{ij}; n_j$ in $t_a$ where $(n_i, n_j) \in E$ is an edge from node $n_i$ to node $n_j$ and $I_{ij}$ is the input constraint annotation on the edge. |

of obsolete, abstract test cases due to node changes is given by $O_N = \{t_a | \exists n \in N_o : t_a \text{ visits } n\}$.

Let $E_o = \{e | e \in E;\ e \text{ is deleted or modified}\}$; then, the set of obsolete abstract test cases due to edge changes is given by $O_E = \{t_a | t_a \text{ tours } e \in E_o\}$. Hence, the set of obsolete, abstract test cases is given by $O_{AT} = O_N \cup O_E$.

Further, let the set of abstract test cases be $AT = \{t_{a_1}, \ldots, t_{a_k}\}$. Let $T_{a_i}$ be the set of executable tests associated with $t_{a_i}$ $(i = 1, \ldots, k)$. Consequently, the obsolete test cases are given by $O_T = \{t | \exists t_a \in O_{AT} : t \in T_a\}$.

**Retestable Test Cases**

[10] defines retestable tests as those that are still valid and test portions of the application that may tour portions of the FSMWeb model that are close to the changes. As an example, assuming edge coverage, they consider how any node $n$ that is one edge away from a modified or deleted node must be retested, except for the source and sink nodes of the AFSM:

$N_{r_{node}} = \{n | \exists e : (n, \hat{n}) \text{ or } (\hat{n}, n); \hat{n} \in N_o; n \neq n_{source}; n \neq n_{sink}\}$.

When edges are changed, the set of retestable edges depends on the type of change. When edges are deleted or modified, it is assumed that the starting and ending nodes of the changed edges are potentially affected, hence non-obsolete tests that visit these nodes are retestable (except for the AFSM's source and sink nodes):

$N_{r_{edm}} = \{n | \hat{e} \in E_o; \hat{e} = (n_i, n) \text{ or } \hat{e} = (n, n_i); n \neq n_{source}; n \neq n_{sink}\}$.

Similarly, when edges (and possibly nodes) are added, the existing nodes at which these new edges start or end are considered potentially affected by the change, hence non-obsolete tests that visit these nodes are retestable (except for the AFSM's source and sink nodes):

$N_{r_{ea}} = \{n | \exists n' \in N' : e' = (n, n') \text{ or } e' = (n', n); n \neq n_{source}; n \neq n_{sink}\}$.

In other words, $n$ appears in both *HFSM* and *HFSM'* related to edge changes. The set of retestable nodes is then given by $N_r = N_{r_{node}} \cup N_{r_{edm}} \cup N_{r_{ea}}$, and the set of retestable abstract test cases is $R_{AT} = \{t_a | t_a \text{ visits } n \in N_r\} \setminus O_{AT}$.

The corresponding executable retestable tests are determined as a function of $R_{AT}$ as before. $R_T = \{t | \exists t_a \in R_{AT} : t \in T_a\}$

**Reusable Tests**

Reusable tests are neither obsolete nor retestable.

$U_{AT} = AT \setminus (O_{AT} \cup R_{AT})$, $U_T = T \setminus (O_T \cup R_T)$

### *B. Selective versus Brute Force Regression Testing*

Intuitively speaking, selective regression testing is preferable if model changes are isolated, if few tests are obsolete, and/or if additions have few overlaps with existing parts of the model. Conversely, when model changes impact many of the abstract test paths, rendering them obsolete, or when additions to the model connect with many parts of the existing model, brute force or systematic regeneration of tests may be better. Likewise, using a brute force approach when we have many obsolete tests, may be a waste of time and it would be better to regenerate the test suite from scratch. In order to quantify the tradeoff properly, both testing processes must be defined so that costs can be associated with the steps of each process. Note that brute force regression testing determines whether a test is obsolete based on validating results of its execution while selective regression testing determine a tests obsoleteness based on classification rules for changes to the FSMWeb model (i.e., without execution).

**Brute Force Regression Testing**

1) Execute the original test suite, $T_0$, with cost $c_e(T_0)$.
2) Validate $T_0$ ($c_v(T_0)$): This process involves determining whether a test passed or failed and whether failure is due to obsolescence or a defect ($c_v(T_0) = c_v(T_p) + c_v(T_f)$). Also note that $T_p$ is comprised of retestable and reusable tests (although they have not been classified as such) and $T_f$ is comprised of failing tests and obsolete tests (again, brute force regression testing does not classify them ahead of execution).
3) Generate new tests to replace $T_{ob}$, and obtain coverage of uncovered parts of the model ($T''$).
4) Execute and validate $T''$. Let the cost of generating, executing, and validating $T''$ be $c_n(T'')$.

Hence, the cost of brute force testing is
$B = c_e(T_0) + c_v(T_0) + c_n(T'')$

**Selective Regression Testing**

1) Classify $T_{ob}$, $T_{rt}$, and $T_{ru}$ at a cost, $c_{class}(T_0)$.
2) Execute $T_{rt}$ at cost $c_e(T_{rt})$.
3) Validate $T_{rt}$ at cost $c_v(T_{rt})$.
4) Generate new tests, $T''$, to replace $T_{ob}$ and missing coverage.
5) Execute and validate $T''$. Let the cost of generating, executing, and validating $T''$ be $c_n(T'')$.

Hence, the cost of selective regression testing is
$S = c_{class}(T_0) + c_e(T_{rt}) + c_v(T_{rt}) + c_n(T'')$

Selective regression testing is preferable when $S < B$. Substituting the formulas for both types of regression testing and simplifying by eliminating the identical terms on both sides of the inequality produces.

$c_{class}(T_0) + c_e(T_{rt}) + c_v(T_{rt}) < c_e(T_0) + c_v(T_0)$

Simplifying further shows that selective regression testing is better when

$c_{class}(T_0) < c_e(T_0 \setminus T_{rt}) + c_v(T_0 \setminus T_{rt})$

Note that we were able to simplify the formula by using the following rules of thumb when generating new tests:

- The cost of generating new tests for missing coverage due to obsolete tests is comparable for brute force and selective regression Testing.
- The cost of generating new tests due to uncovered test requirements (because none exist in the prior version) is comparable as well.

### C. Tradeoff Computations for Decision Making

Having a tradeoff inequality such as the one we derived above is only the first step to make good decisions. The costs actually have to be measured, and parameters in the cost functions have to be estimated. We can streamline this by making simplified assumptions. (These assumptions will be evaluated and adjusted through empirical studies.) These will have to be checked when using the formulas. If they do not hold, the estimate may be off. We can simplify the tradeoff formula further by making assumptions about the cost of executing and validating a test. Let $t \in T$ be a test $t$ in test suite $T$.

**Assumption 1 (option 1):** The cost of executing $t$, $c_e(t)$, and the cost of validating $t$, $c_v(t)$, are proportional to the length of test case $t$.

$c_e(t) = a * len(t) = a * \sum_{e \in t} e$ and

$c_v(t) = b * len(t) = b * \sum_{e \in t} e$

where $e$ is an edge in the model, and $a$ and $b$ are positive calibration parameters.

Alternatively, length could be measured by the number of nodes in test $t$. Each abstract test path contains edge predicates that specify the input parameter(s) that must be selected for the transition to occur. Option 1 assumes that differences for input number and size average out, hence an average cost for executing and validating edges can be used. If this assumption does not hold, it becomes necessary to consider the inputs instead of edges.

**Assumption 1 (option 2):** The cost of executing and validating test $t$ is proportional to the number of inputs on each edge annotation. Let $I_e$ be the set of input parameters for edge $e$ and $card(I_e)$ be the number of such inputs. Then,

$c_e(t) = d * \sum_{e \in t} card(I_e)$ and $c_v(t) = f * \sum_{e \in t} card(I_e)$

As before, $e$ is an edge in $t$ while $d$ and $f$ are positive calibration parameters. Similarly, we can make a simplifying assumption about the classification cost.

**Assumption 2:** The cost of classification, $c_{class}$, is proportional to the size of the existing test suite, $T_0$. Hence,

$c_{class} = x * card(T_0)$ for some positive $x$.

This formula assumes that differences in classification cost due to the length of a test suite average out. As before, when this assumption does not work well, we can use the length of the test case instead:

$c_(class) = y * \sum_{t \in T_0} len(t)$ for some positive $y$.

Some implementations can classify tests in a fixed number of operations if they create a bit mask during test creation that identifies which nodes/edges are visited in a path, rendering the classification cost independent from the test case length.

### D. Determining Calibration Parameters

Because the calibration parameters are not known a priori, we suggest to use a simple threshold of the change amount in the model, to accept that the first decision may not be optimal, and then to record the actual cost and determine the first estimators for the calibration parameters, improving them each round.

Whether calibration parameters for one project can be reused on another should be determined by project characteristics. If the projects are similar, one may start reusing parameters, improving them along the way. Otherwise, it may be better to start fresh. If the costs for selective and brute force approaches are close, making an incorrect decision involves minor cost differences, hence the cost penalty is small. What needs to be avoided are incorrect decisions that carry a sizable cost difference.

Most of our assumptions and tradeoff formulas are either independent from FSMWeb specifics or can easily be generalized to other graph models such as EFSMs or UML behavioral models.[2] It stands to reason that this tradeoff approach could be adapted to other types of models. However, there is one FSMWeb characteristic that offers unique opportunities, its hierarchical nature and the opportunity for partial test path regeneration in the face of changes. This issue is discussed in the next subsection.

### E. Partial Regeneration

Because of the hierarchical nature of the FSMWeb models and the associated three stage test generation, it is possible to use partial regeneration to replace obsolete test cases and, thus, potentially save costs. Also, note that FSMWeb generates a flattened version of concrete tests and avoids having to flatten the model from which the tests are generated. The flattened version is necessary to make them executable.

Recall that $HFSM = \{FSM\}_{i=0}^{n}$. If changes are isolated to a few $FSM_i$, i.e., $\forall FSM_i' : FSM_i' \neq FSM_i$ ($FSM_i$ changed):

- classify abstract tests.
- regenerate paths through $FSM_i'$ to cover uncovered edges.
- reaggregate abstract tests that contain the cluster node that stands for $FSM_i$ with the new paths.

When there are changes in many $FSM_i$ and/or at high levels of the hierarchy, this will approach full regeneration. Thresholds that determine whether partial or full regeneration is warranted could be defined. We need one threshold, $B_1$, for the proportion of $FSM_i$ that changed and another for the degree of change, $B_{2i}$, in $FSM_i$.

[2]The classification rules for types of model changes have to be restated in terms of these other models.

Then, as long as $\frac{No.ofFSM_i changed}{n+1} = PC_{HFSM} < B_1$, we will use partial regeneration. For partial regeneration to be better, its savings must be greater than the classification cost.

To assess the degree of change within an $FSM_i$, we can set the threshold for change based on the proportion of changed nodes or edges. In an $HFSM$ with edges outnumbering nodes, a fair amount (navigation bars and links in web applications tend to lead to this type of graph), using degree of edge change, is probably more fruitful, and we use it here. Further, note that only two types of node changes exist: An LWP becomes a cluster node or a cluster node becomes an LWP. Any global impact would have to show itself in terms of modifications of inputs along the way and be then reflected in edge changes.

Then, as long as $B_{2i} > \frac{No.ofedges(deleted,added,modified)}{card(E'_i)} = PC_{FSM_i}$ (where $card(E'_i)$ is the number of edges in $FSM'_i$), we will use selective regression testing.

Setting these thresholds is initially a matter of how conservative the estimate should be. A low threshold will lead to more regeneration, a higher one to more partial reconstruction.

## IV. Example

In this section, we will illustrate our tradeoff framework using a simple example, Figure 2, which has five FSMs and three levels of hierarchy. Table II shows paths through each FSM that achieve edge coverage.

These test paths now have to be aggregated to form abstract tests through the HFSM. As aggregation coverage criterion we use the all-paths substitution. We illustrate this on $t_{01} = n_1c_2n_2$. Substituting $t_{21}$ and $t_{22}$ for $c_2$ results in two paths: $p_1 = n_1n_6n_7n_2$ and $p_2 = n_1n_6c_4c_3n_7n_2$.

The first path consists of all LWP nodes and does not have to be refined. The second path uses cluster nodes $c_4$ and $c_3$, each of which resolved by substituting two paths each, resulting in four test paths. Hence, $t_{01}$ results in five test paths when fully aggregated.

TABLE II
Testpaths through Individual FSMs

| FSM | Testpaths |
|---|---|
| $AFSM$ | $t_{01} = n_1c_2n_2$, $t_{02} = n_1c_1n_2$ |
| $FSM_1$ | $t_{11} = n_3n_4c_2n_5n_4n_3$, $t_{12} = n_3n_4c_2n_5c_3n_3$ |
| $FSM_2$ | $t_{21} = n_6n_7$, $t_{22} = n_6c_4c_3n_7$ |
| $FSM_3$ | $t_{31} = n_8n_{13}n_{14}$, $t_{32} = n_8n_9n_{10}n_{12}n_9n_{10}n_{11}n_{14}$ |
| $FSM_4$ | $t_{41} = n_{15}n_{16}n_{17}n_{18}$, $t_{42} = n_{15}n_{16}n_{19}n_{17}n_{18}$ |

Aggregation of $t_{02}$ is more involved, because it visits cluster node $c_1$ requiring aggregation of two test paths through $FSM_1$ which both use $c_2$ (2 paths), one of which uses $c_4$ and $c_3$ (the latter twice) with 2 paths each. This results in 20 paths when FSMs are fully aggregated. Table III shows these paths, including derivation rules and the test path lengths. The first five test paths are obtained by applying $t_{01}$, and the rest of them are are obtained by applying $t_{02}$.

Next, consider each change. Change 1 is a change at the lowest level; $t_{41}$ is obsolete; $t_{42}$ is still valid. To achieve coverage, we can add $t'_{41} = n_{15}n_{18}$. All paths that contain substitutions for $t_{41}$ need to be reaggregated. This affects 8 paths, 2 based on $t_{01}$ and 6 based on $t_{02}$. Change 2 is also a change at the lowest level, because all nodes in $FSM_3$ are LWP nodes. The test path $t_{32}$ in $FSM_3$ is obsolete. Two new test paths need to be generated. They replace $t_{32}$ in the 11 obsolete tests, resulting in 22 new tests. Change 3 renders $t_{22}$ through $FSM_2$ obsolete. Two new test paths are needed: $t'_{21} = n_6c_4n_7$ and $t'_{22} = n_6c_3n_7$.

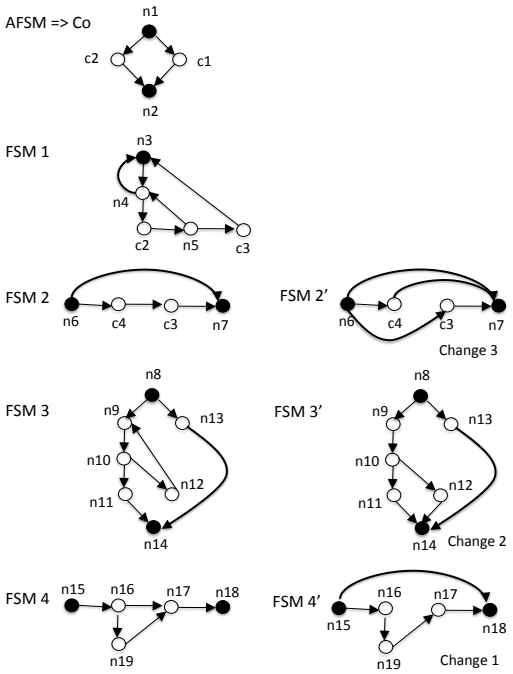


Fig. 2. Example with Three Changes

All paths where cluster node $c_2$ was resolved with the obsolete path must be reaggregated, resulting in all new test paths, because both $t_{01}$ and $t_{02}$ include cluster node $c_2$ at some point in the aggregation. In addition, since the number of paths through $FSM'_2$ increased from 2 to 3, the resulting number of aggregated test paths multiplies.

### *Practical Thoughts on Combining Partial Regeneration with Selective Regression Testing*

Test generation in FSMWeb naturally lends itself to partial regeneration, because it is based on three stages:

1) Generate paths through FSMs.
2) Aggregate paths into abstract tests.
3) Select inputs along paths.

In the example, when considering each change individually, only branches in a single FSM are uncovered after changes, hence it would be a waste of time to regenerate paths through FSMs that have not changed in step 1. Rather, it is a matter of how much reaggregation needs to occur. If the derivation history of each $t \in T_0$ was kept, this can mean that, in all $t \in T_{ob}$ that contain an obsolete path through some $FSM_i$, a valid new path through $FSM_i$ is substituted. If more than one new path is generated, the number of new tests multiplies. Similarly, because the derivation history would indicate the point of first breakage for an obsolete test (i.e., where the

obsolete path started), it is also known where to restart value generation.

In a small pilot study, we had 4 Ph.D. students and 1 M.S. student manually generate tests for the example. All students kept the derivation history so that they could check their work, compare results with their peers, and more easily identify discrepancies. Hence, it appears reasonable to assume that derivation history is available.

**Change 1:** There are eight tests that use $t_{41}$ with the deleted edge $(n_{16}n_{17})$. Substituting the new path, $t'_{42} = n_{15}n_{18}$ for all occurrences of $t_{42}$ together with the eight retestable paths provides the required coverage. Note that, because all nodes in $FSM_4$ are LWPs, no lower level substitutions need to take place. Value generation from the point of substitution will complete the test suite. Of course, for regression testing purposes, the reusable tests need not be run.

**Change 2:** There are 11 tests which use obsolete path $t_{32}$ (edge $(n_{12}, n_9)$ is missing). To cover the new edge $(n_{12}, n_{14})$, an additional path is needed, so there are two new paths:

$t'_{32} = n_8n_9n_{10}n_{11}n_{14}$ and $t'_{33} = n_8n_9n_{10}n_{12}n_{14}$

Similar to change 1, we substitute each path for the obsolete $t_{32}$, resulting in 26 new abstract tests. There are a total of 11 such paths. Two of them have double occurrences, resulting in four new paths each due to our aggregation coverage criterion. As in change 1, all nodes in $FSM_3$ are LWPs, so no lower level substitutions need to take place. Value generation from the first point of substitution completes the test suite. (Note that, because of possible dependencies of inputs selected between two substitutions in a test path, it is not possible to keep these old values.) There are no retestable tests, so the regression test suite consists of 26 tests.

**Change 3:** This change requires modifications through several substitution levels. This change makes $t_{22}$ obsolete. Because a large portion of the tests is derived from $t_0 = n_1c_2n_2$, 16 of the 20 tests are obsolete, suggesting that a great deal more reaggregation is necessary. Because $c_2$ occurs at the highest level of aggregation, this really means that we have to reaggregate from $t_{01}$ on down. At the same time, only $FSM_2$ needs path changes, namely two new paths: $t'_{22} = n_6c_3n_7$ and $t'_{23} = n_6c_4n_7$.

Because the new paths only refer to one lower level cluster each, there are only 4 new abstract tests to substitute for the obsolete ones, two for $c_3$ and 2 for $c_4$. Value generation has to be done almost from the beginning. There are no retestable tests, so the regression test suite consists of 32 tests.

Changes 2 and 3 affect more paths at more levels in the hierarchy than change 1, because cluster nodes associated with the changes occur in more of the $FSMs$ or at higher levels.

So far, we have assumed single changes. What if all three changes occur at the same time? There are 17 obsolete test cases and 3 retestable ones. In this case, reaggregation needs to start with $t_{01}$ and $t_{02}$ at the top level, and value generation also must start almost from the beginning. However, we can still reuse paths through $AFSM$ and $FSM_1$ as well as the ones in the modified FSMs that have not become obsolete. Depending on the character of the changes, these can be found in retestable or reusable tests. In total, the modified FSMs require 30 test cases: six for $t_{01}$ and 24 for $t_{02}$.

In a small pilot study as part of an advanced graduate class in software testing, five graduate students derived test paths, and performed selective regression test generation using partial regeneration for the example. The task was done manually with a pen and paper. The same coverage and aggregation criteria were used as in the previous example. The original test path generation time ranged between 1.5 and 3 hours. Change 1 took 10-20 minutes; change 2 took 20-30 minutes; and change 3 took 20-42 minutes.

This data roughly reflects the progressively higher impact of the changes on the paths. Even though the third change was more involved and required much more regeneration, the time spent was still much lower than the original abstract test generation times. We speculated that learning effects might account for some of this discrepancy; the students had become familiar with the model and its tests. We plan on investigating this phenomenon further.

**Cost Tradeoffs Between Brute Force and Selective Approaches**

Now, we analyze the cost tradeoffs between the brute force and selective approaches for each change in the example.

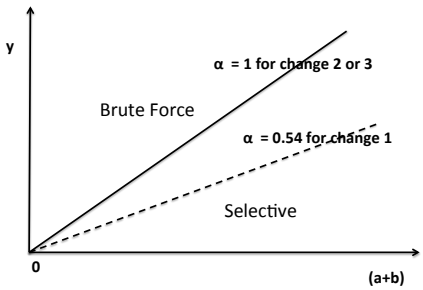


Fig. 3. Cost Tradeoffs Between Brute Force and Selective Approaches

**Change 1:** For change 1, obsolete tests are {2, 4, 7, 9, 13, 15, 17, 19}, and retestable tests are {3, 5, 8, 10, 14, 16, 18, 20}. Thus, $T_0 \setminus T_{rt} = \{1, 2, 4, 6, 7, 9, 11, 12, 13, 15, 17, 19\}$, and the lengths of $T_0 \setminus T_{rt}$ and $T_0$ are 196 and 360, respectively. Thus, using $c_{class}(T_0) < c_e(T_0 \setminus T_{rt}) + c_v(T_0 \setminus T_{rt})$, we have $y * 360 < (a + b) * 196$, and finally, $y < (a + b)\frac{196}{360}$, where $y$, $a$, and $b$ are the effort parameters for test classification, test execution, and test validation, respectively. For this case, in Figure 3, the slope, $\alpha$, is 0.54. If the relationship between parameters falls under the slope, selective regression testing is beneficial; otherwise, brute force regression testing is beneficial.

**Change 2:** For change 2, obsolete tests are {4, 5, 9, 10, 12, 15, 16, 17, 18, 19, 20}, and retestable tests are $\{\phi\}$. Thus, $T_0 \setminus T_{rt} = T_0$, and the length of both $T_0 \setminus T_{rt}$ and $T_0$ is 360. Using the same equation, we have $y * 360 < (a + b) * 360$, and then finally, $y < (a + b)$. In Figure 3, the slope, $\alpha$, is 1.

**Change 3:** Obsolete tests are {2, 3, 4, 5, 7, 8, 9, 10, 13, 14, 15, 16, 17, 18, 19, 20} and retestable tests are $\{\phi\}$. Because $T_0 \setminus T_{rt} = T_0$, the slope, $\alpha$, is also 1 for change 3.

TABLE III
TEST PATHS FOR AFSM

| $t_{01} : n_1c_2n_2$ | | | |
|---|---|---|---|
| Test | Test Paths | Derivation Rules | Length |
| 1 | $n_1\mathbf{t_{21}}n_2$ | $c_2 \rightarrow t_{21}$ | 4 |
| 2 | $n_1n_6\mathbf{t_{41}t_{31}}n_7n_2$ | $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, and $c_3 \rightarrow t_{31}$ | 11 = 4 + 4 + 3 |
| 3 | $n_1n_6\mathbf{t_{42}t_{31}}n_7n_2$ | $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, and $c_3 \rightarrow t_{31}$ | 12 = 4 + 5 + 3 |
| 4 | $n_1n_6\mathbf{t_{41}t_{32}}n_7n_2$ | $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, and $c_3 \rightarrow t_{32}$ | 16 = 4 + 4 + 8 |
| 5 | $n_1n_6\mathbf{t_{42}t_{32}}n_7n_2$ | $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, and $c_3 \rightarrow t_{32}$ | 17 = 4 + 5 + 8 |
| $t_{02} = n_1c_1n_2$ | | | |
| Test | Test Paths | Derivation Rules | Length |
| 6 | $n_1n_3n_4\mathbf{t_{21}}n_5n_4n_3n_2$ | $c_1 \rightarrow t_{11}$, $t_{11} \rightarrow n_3n_4c_2n_5n_4n_3$, and $c_2 \rightarrow t_{21}$ | 9 = 7 + 2 |
| 7 | $n_1n_3n_4n_6\mathbf{t_{41}t_{31}}n_7n_5n_4n_3n_2$ | $c_1 \rightarrow t_{11}$, $t_{11} \rightarrow n_3n_4c_2n_5n_4n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, and $c_3 \rightarrow t_{31}$ | 16 = 9 + 4 + 3 |
| 8 | $n_1n_3n_4n_6\mathbf{t_{42}t_{31}}n_7n_5n_4n_3n_2$ | $c_1 \rightarrow t_{11}$, $t_{11} \rightarrow n_3n_4c_2n_5n_4n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, and $c_3 \rightarrow t_{31}$ | 17 = 9 + 5 + 3 |
| 9 | $n_1n_3n_4n_6\mathbf{t_{41}t_{32}}n_7n_5n_4n_3n_2$ | $c_1 \rightarrow t_{11}$, $t_{11} \rightarrow n_3n_4c_2n_5n_4n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, and $c_3 \rightarrow t_{32}$ | 21 = 9 + 4 + 8 |
| 10 | $n_1n_3n_4n_6\mathbf{t_{42}t_{32}}n_7n_5n_4n_3n_2$ | $c_1 \rightarrow t_{11}$, $t_{11} \rightarrow n_3n_4c_2n_5n_4n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, and $c_3 \rightarrow t_{32}$ | 22 = 9 + 5 + 8 |
| 11 | $n_1n_3n_4\mathbf{t_{21}}n_5\mathbf{t_{31}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{21}$, and $c_3 \rightarrow t_{31}$ | 11 = 6 + 2 + 3 |
| 12 | $n_1n_3n_4\mathbf{t_{21}}n_5\mathbf{t_{32}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{21}$, and $c_3 \rightarrow t_{32}$ | 16 = 6 + 2 + 8 |
| 13 | $n_1n_3n_4n_6\mathbf{t_{41}t_{31}}n_7n_5\mathbf{t_{31}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, $c_3 \rightarrow t_{31}$ (first $c_3$), and $c_3 \rightarrow t_{31}$ (second $c_3$) | 18 = 8 + 4 + 3 + 3 |
| 14 | $n_1n_3n_4n_6\mathbf{t_{42}t_{31}}n_7n_5\mathbf{t_{31}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, $c_3 \rightarrow t_{31}$ ((first $c_3$), and $c_3 \rightarrow t_{31}$ (second $c_3$) | 19 = 8 + 5 + 3 + 3 |
| 15 | $n_1n_3n_4n_6\mathbf{t_{41}t_{32}}n_7n_5\mathbf{t_{31}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, $c_3 \rightarrow t_{32}$ (first $c_3$), and $c_3 \rightarrow t_{31}$ (second $c_3$) | 23 = 8 + 4 + 8 + 3 |
| 16 | $n_1n_3n_4n_6\mathbf{t_{42}t_{32}}n_7n_5\mathbf{t_{31}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, $c_3 \rightarrow t_{32}$ (first $c_3$), and $c_3 \rightarrow t_{31}$ (second $c_3$) | 24 = 8 + 5 + 8 + 3 |
| 17 | $n_1n_3n_4n_6\mathbf{t_{41}t_{31}}n_7n_5\mathbf{t_{32}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, $c_3 \rightarrow t_{31}$ (first $c_3$), and $c_3 \rightarrow t_{32}$ (second $c_3$) | 23 = 8 + 4 + 3 + 8 |
| 18 | $n_1n_3n_4n_6\mathbf{t_{42}t_{31}}n_7n_5\mathbf{t_{32}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, $c_3 \rightarrow t_{31}$ (first $c_3$), and $c_3 \rightarrow t_{32}$ (second $c_3$) | 24 = 8 + 5 + 3 + 8 |
| 19 | $n_1n_3n_4n_6\mathbf{t_{41}t_{32}}n_7n_5\mathbf{t_{32}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{41}$, $c_3 \rightarrow t_{32}$ (first $c_3$), and $c_3 \rightarrow t_{32}$ (second $c_3$) | 28 = 8 + 4 + 8 + 8 |
| 20 | $n_1n_3n_4n_6\mathbf{t_{42}t_{32}}n_7n_5\mathbf{t_{32}}n_3n_2$ | $c_1 \rightarrow t_{12}$, $t_{12} \rightarrow n_3n_4c_2n_5c_3n_3$, $c_2 \rightarrow t_{22}$, $t_{22} \rightarrow n_6c_4c_3n_7$, $c_4 \rightarrow t_{42}$, $c_3 \rightarrow t_{32}$ (first $c_3$), and $c_3 \rightarrow t_{32}$ (second $c_3$) | 29 = 8 + 5 + 8 + 8 |

## V. COST-BENEFIT TRADEOFFS IN PRACTICE

As we observed from the previous section, the amount of changes and their impact on other model parts can affect the choice of regression testing techniques. In this paper, we only considered safe regression testing techniques, but depending on the organization's situations, they might not be the best option to use. For instance, some organizations might prefer a technique that does not guarantee to retest all changed components but tests the important components to release the product earlier. Thus, to consider the practicality of regression testing, we need to project the comparative costs between techniques in light of their business value to organizations by considering several factors, such as the type of applications that organizations build, the product's market share, revenue associated with the product, the product's release time sensitivity, and revision/testing frequency for the product.

These factors are interrelated to each other, and a certain factor can greatly impact others. For instance, if a software game company is targeting Christmas shoppers who would purchase the new game product, then the product's release time greatly affects revenue and the product's market share. Thus, delaying the product release would be a great loss for the company. In particular, the type of applications and the organization's goal would be the major influential factors for others. In general, we can consider two types of regression testing approaches: safe regression testing and economy-sensitive regression testing.

**Safe Regression Testing**

If the organization's goal is to build dependable software, then we need to consider safe regression test selection techniques. (e.g., Organizations that build web applications handle financial issues or deal with sensitive data.) In this case, the choice is very obvious. The organizations can choose a brute force approach or a safe, selective testing approach, and calculating the tradeoffs between techniques is straightforward. We just need to compare costs for the two techniques as shown in earlier sections.

However, this simple cost difference should be projected by considering the economic impact of the savings. For example, if the relationship between the product release time and revenue gains is insignificant or ignorable, then a rough threshold estimation would be acceptable when the organization chooses a regression testing technique. If the product release time affects revenue gains greatly, then small cost savings from one technique over the other could bring significant revenue gains. In that case, a more accurate threshold should be estimated.

**Economy-Sensitive Regression Testing**

If the organization's goal is to make a profit, then testing the product throughly and, as a consequence, delaying the product release might not be a good strategy. Actually, the majority of profit-pursuing organizations have time to market pressures due to the constraint budgetary problem and a competitive software market. To resolve this situation, companies might release the buggy product early and patch it later. Many companies are using this strategy to achieve an early market share because delay is costly [13].

As we can speculate, this approach has tradeoffs. It leads to a fast delivery, thus it is competitive in the market. However, some faults might escape, thus the product could have low reliability. The field failure can be very expensive depending on types of faults. Further, a bad reputation for quality could lead to losing customers in the long term.

Thus, when we apply non-safe regression testing techniques that are more sensible for the majority of software companies,

we need to consider their cost-benefit tradeoffs so that organizations can choose techniques that are more beneficial for their circumstances. To test a new software version, we can consider the following tasks:

1) Create and run new tests that are associated with new or modified components.
2) Select existing tests that are associated with more important components or risky (critical) components.
3) Prioritize existing tests based on certain criteria (e.g., code coverage, risk factor, and criticality), and run important tests early until time runs out.

To offer the best strategy for the company, these three tasks can be used together, considering the cost of fixing defects (detected after release) and potential revenue gains from early product release.

Consider the example we illustrated in Section IV. Suppose that the component for $FSM_4$ has been developed by a third party; the others are in-house components; and all three FSMs are modified as shown in Figure 2 due to feature updates. Assume that the modified FSMs are ready to retest, but the testing team does not have enough time to retest all of them because the product release deadline has already been delayed and because the company wants to ship the product as soon as possible so that it can minimize the profit loss. Because safe regression testing is not a good choice for this situation, the testing team could decide to run more important or risky test cases before releasing the product, and then run the rest of the tests later. Assuming that third party components are riskier and take more time when defects occur due to the longer communication required with an outside party, the testing team can run tests that use paths through $FSM_4'$ first.

## VI. Related Work

The simplest regression testing technique is to rerun all existing test cases, but it can be very expensive in practice [14]. Thus, various regression testing techniques have been proposed and empirically evaluated to improve the cost-effectiveness of regression testing, such as regression test selection techniques (e.g., [4], [12]), test suite minimization (or reduction) techniques (e.g., [5], [15]), and test case prioritization technique (e.g., [3], [16]). The majority of this research focused on regression testing for desktop applications such as C or Java, but more recent research targeted web applications because the use of web applications has grown rapidly over the past decade (e.g., [17], [10]. Two recent surveys [18], [19] provide an overview of the regression testing techniques and approaches and the current state of research on them.

Most early work on regression testing utilized simple measures, such as the number of test cases reduced, time savings from regression test selection, or the rate of fault detection to evaluate the effectiveness of regression testing techniques. With such simple metrics, however, we cannot properly assess the costs and benefits of regression testing techniques in practical settings.

To address this problem, a few models of regression testing have been proposed. Leung and White [7] present a cost model that considers some factors (analysis cost, testing time, and technique execution time) that affect the cost for regression testing a software system. While they provide the cost relationship between the retest-all and selective strategies, no empirical work has been presented. Malishevsky et al. [20] extend this work with cost models for regression test selection and test case prioritization that incorporate benefits related to fault omission and the rate of fault detection, and the authors assess the cost-benefits of regression testing techniques through empirical results. Do and Rothermel [21], [6] present an economic model (EVOMO) and evaluate the costs and benefits of regression testing processes that capture both cost and benefit factors relevant to those processes and facilitate their evaluation across system lifecycles. However, these models allow evaluation only after applying the techniques to the applications. Unlike these, in this work, we propose a tradeoff framework that can choose the proper techniques before applying the techniques.

## VII. Conclusions and Future Work

In this paper, we proposed a cost-benefit tradeoff framework for FSMWeb models that helps choose between selective and brute force regression testing. With an example, we illustrated how the tradeoff framework can be applied when analyzing the cost-benefit tradeoffs of the brute force and selective regression testing techniques when models change, and we established the relationship among effort parameters for cost estimation. We also discussed the practical issues to consider when choosing regression testing techniques that can support the organization's business goals. In this work, we applied our framework to the FSMWeb model. We plan to adapt the framework to other graph-based behavioral models, such as UML behavioral models.

The proposed tradeoff framework provides a quick and simple way to choose a cost-effective regression testing technique by simplifying the assumptions for cost parameters, but these parameters should be adjusted through calibration and empirical evaluation efforts. Toward this direction, we plan to conduct case studies using industrial web applications through collaboration with graduate students at the University of Denver who work for software companies and to calibrate cost parameters based on empirical results. Further, we plan to evaluate our framework considering its efficiency, effectiveness, and user experiences through such case studies.

## Acknowledgment

This work was supported, in part, by NSF IUCRC grant #0934413 to the University of Denver and NSF CAREER Award CCF-1149389 to North Dakota State University.

## References

[1] A. von Mayrhauser and N. Zhang, "Automated Regression Testing Using DBT and Sleuth," *Software Maintenance: Research and Practice*, vol. 11, no. 4, pp. 93–116, 1999.

[2] R. Boggs, J. Bozman, and R. Perry, "Reducing downtime and business loss: Adressing business risk with effective technology," IDC, Tech. Rep., Aug. 2009.

[3] S. Elbaum, A. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," *IEEE Transactions on Software Engingeering*, vol. 28, no. 2, pp. 159–182, 2002.

[4] M. Harrold, D. Rosenblum, G. Rothermel, and E. Weyuker, "Empirical studies of a prediction model for regression test selection," *IEEE Transactions on Software Engingeering*, vol. 27, no. 3, pp. 248–263, 2001.

[5] J. Jones and M. Harrold, "Test suite reduction and prioritization for modified condition/decision coverage," *IEEE TSE*, vol. 29, no. 3, pp. 193–209, 2003.

[6] H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel, "The effects of time constraints on test case prioritization: A series of controlled experiments," *IEEE Transactions on Software Engineering*, vol. 26, no. 5, pp. 593–617, Sep. 2010.

[7] H. Leung and L. White, "A cost model to compare regression test strategies," in *Proceedings of the International Conference on Software Maintenance*, October 1991, pp. 201–208.

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

[9] A. A. Andrews, J. Offutt, and R. T. Alexander, "Testing Web Applications by Modeling with FSMs," *Journal of Software and Systems Modeling*, vol. 4, no. 3, pp. 326–345, 2005.

[10] A. Andrews, S. Azghandi, and O. Pilskalns, "Regression testing of web applications using FSMWeb," in *Proceedings of the International Conference on Software Engineering and Applications*, 2010.

[11] H. Leung and L. White, "Insights into Regression Testing ," in *Proceedings of the International Conference on Software Maintenance*, October 1989, pp. 60–69.

[12] G. Rothermel and M. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engingeering*, vol. 22, no. 8, pp. 529–551, 1996.

[13] A. Arora, J. P. Caulkins, R. Caulkins, and R. Telang, "Sell first, fix later: Impact of patching on software quality," *Management Science*, vol. 52, no. 3, pp. 465–471, Mar. 2006.

[14] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 97–106, 2002.

[15] J. Offutt, J. Pan, and J. M. Voas, "Procedures for reducing the size of coverage-based test sets," in *Proc. Int'l. Conf. Testing Comp. Softw.*, Jun. 1995, pp. 111–123.

[16] W. Wong, J. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," in *Proceedings of the International Symposium on Software Reliability Engineering*, November 1997, pp. 264–274.

[17] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, and A. Souter, "An empirical comparison of test suite reduction techniques for user-session-based testing of web applications," in *Proceedings of the International Conference on Software Maintenance*, October 2005, pp. 587–596.

[18] E. Engstrom, P. Runeson, and M. Skoglund, "A systematic review on regression test selection techniques," *Information and Software Technology*, vol. 52, no. 1, pp. 14 – 30, 2010.

[19] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritisation: A survey," *JSTVR*, pp. 67–120, Mar. 2010.

[20] A. Malishevsky, G. Rothermel, and S. Elbaum, "Modeling the cost-benefits tradeoffs for regression testing techniques," in *Proceedings of the International Conference on Software Maintenance*, October 2002, pp. 204–213.

[21] H. Do and G. Rothermel, "Using sensitivity analysis to create simplified economic models for regression testing," in *Proceedings of the International Symposium on Software Testing and Analysis*, July 2008, pp. 51–61.