Impact Analysis in the Presence of Dependence Clusters Using Static Execute After in WebKit

Lajos Schrettner†, Judit Jász‡, Tamás Gergely†, Árpád Beszédes† and Tibor Gyimóthy†

†Department of Software Engineering
University of Szeged, Hungary
Email:{schrettner, gertom, beszedes, gyimothy}@inf.u-szeged.hu

‡Research Group on Artificial Intelligence
Hungarian Academy of Sciences
Szeged, Hungary
Email: jasy@inf.u-szeged.hu

***Abstract*—Impact analysis based on code dependence can be an integral part of software quality assurance by providing opportunities to identify those parts of the software system that are affected by a change. Because changes usually have far reaching effects in programs, effective and efficient impact analysis is vital, which has different applications including change propagation and regression testing. Static Execute After (SEA) is a relation on program elements (procedures) that is efficiently computable and accurate enough to be a candidate for use in impact analysis in practice. To assess the applicability of SEA in terms of capturing real defects, we present results on integrating it into the build system of WebKit, a large, open source software system, and on related experiments. We show that a large number of real defects can be captured by impact sets computed by SEA, albeit many of them are large. We demonstrate that this is not an issue in applying it to regression test prioritization, but generally it can be an obstacle in the path to efficient use of impact analysis. We believe that the main reason for large impact sets is the formation of dependence clusters in code. As apparently dependence clusters cannot be easily avoided in the majority of cases, we focus on determining the effects these clusters have on impact analysis.**

***Index Terms*—Change impact analysis, Source code analysis, Static Execute After, Regression testing, Dependence clusters.**

I. Introduction

During software development and maintenance it is vitally important to ensure that changes made to a system do not degrade its quality. Even a smallest change can affect a significant portion of the system and, obviously, the changed parts are affected directly, but we should look further and determine which other parts are affected indirectly. Impact analysis methods deal with this situation [8], but unfortunately, indirectly affected parts are often neglected. Impact analysis can aid implementing the changes in various ways. Most importantly, it can give hints about where in the system the changes need to be propagated [9] and, after the changes have been implemented, it can aid regression testing to capture defects introduced nevertheless [19]. Comprehensive retesting may be very expensive in this case, so it is beneficial to be able to identify a subset of the test suite that tests those parts of the system which may be affected by a particular change.

A widely accepted impact analysis method rests on static code dependence analysis, most notably (forward) slicing based on System Dependence Graphs [12]. Slicing is accurate, but at the same time it is computationally rather expensive. Static Execute After (SEA) is an alternative to slicing on the procedure level as it is more efficient at the expense of being a bit less accurate [13], [14]. Our long term goal is to build on this potential to let software developers use impact analysis in their daily routine.

Recently we have experimented [15] with integrating SEA into the build environment of WebKit [23], a real size, complex, and lively evolving software system. In this paper we report on an extended set of experiments using SEA for impact analysis in WebKit. We use real defect data based on regression test execution result histories, and relate them to changes that introduced and later eliminated defects. We do this by checking whether the impact sets of the failure introducing changes contain the modifications at the fixing revisions (we call this the *prediction capability* of impact analysis). By using a larger data set than previously and an improved analysis method, we have found that SEA performs well in terms of prediction, but the impact sets are often very large. This can be a problem when using the impact sets for manual change propagation, but for other applications such as automatic regression test selection and prioritization they can still be useful, however. Hence we experimented with augmenting our former results on selective regression testing with a new algorithm that takes impact sets into account in addition to the changes.

After investigating the large impact sets we observed that the main reason for such sets seems to be the formation of large dependence clusters in code [6]. The root causes of this phenomenon are not well understood yet; it seems to be an inherent property of program code dependence graphs. As apparently dependence clusters cannot be easily avoided in the majority of cases, we focused our further research on determining the effects these clusters have on impact analysis. We found that changes can often be associated to clusters, so

this supports the findings of other researchers that refactoring clusters should be a concern.

The contributions of this paper are the following:

- We tested the SEA algorithm on a large industrial system and found that it is suitable to be integrated into the build environment and exhibits good prediction capabilities.
- We defined SEA-based dependence clusters and established that they are the primary causes of large impact sets which hinder the applicability of change-based impact analysis in some applications.
- We checked whether the theoretical importance of the formation of large dependence clusters, which seem to present an inherent limitation in impact analysis, can be justified in practice.
- We showed how SEA-based impact analysis can be used to enhance test case selection and prioritization in a regression test environment.

The paper is organized as follows. In Section II, we overview related research. Section III provides background information about the concepts used and formulates the research questions we seek answers for. Section IV describes the data collecting methods that were used and the measurements that were carried out. Section V presents results and discusses threats to their validity. Finally, we conclude in Section VI.

II. Related work

A. Impact analysis

Impact analysis [8] deals with the problem of identifying those parts (the impact set) of a software system that might be affected by a change in the system, in other words finding the possible dependences between the change and the other parts of the system. There are different fundamental approaches for determining impact sets. In this work we are concerned with code dependences [16].

Program slicing based on dependence graphs [12] is one of the most important practical methods for this task. Here, first a program representation is built that captures all the different dependences between program elements like statements, and then a reachability algorithm finds the dependences starting from a particular point (the slicing criterion), which represents the initial change. Although there are reports on the usability of some variants of program slicing on large programs [1], usually these algorithms pose serious challenges in the case of present day big and complex software projects, which is due to the inherent complexity of the algorithms.

Because of these problems, other, more light-weight approaches are often used such as those based on a call graph of the program. Many of them determine the impact sets of the modified procedures of a program from the call graph [8], [22]. Although these methods are simple, they are not safe, and it is easy to show that they miss to identify a set of real dependences [2]. We presented empirical evidence that the SEA relations can be a good approximation of program slices at the procedure level, while its computational complexity is better than that of program slicing [2], [13], [14].

Our method and the above mentioned approaches concentrated only on the source code and its changes in a system. But in many cases we can gain important information from other software artifacts like software repositories, bug tracking systems, or natural language texts [18], [29]. This information can then be used to derive special kinds of impact sets, but this work is not concerned with this area.

B. Dependence Clusters

The research on fine granularity dependence clusters has been initiated by Binkley and Harman [6], who defined dependence clusters as a maximal set of program elements that each depend on the other (on the SDG program representation). They have been associated with backward program slices, and eventually, with the program slice sizes. This approximation turned out to be a good instrument to work with dependence clusters.

The current view is that large dependence clusters are detrimental to the software development process, in particular they hinder impact analysis because in any dependence-related examination encountering any member of cluster immediately pulls in the whole cluster [1], [5], [7], [11]. Furthermore, it seems that dependence clusters are independent of the programming language and the type of the system, hence they are universally present as an inherent property of programming practice [1], [2], [10].

Despite these observations, at present there are relatively few results reported on how to find dependence cluster causes effectively, and subsequently refactor them into smaller dependence formations. In this work, we further support the universality of SEA-based dependence clusters by identifying them in the WebKit system, and discuss their effects in the context of impact analysis and defects.

C. Regression testing

In this work, we apply the SEA method also to regression test selection and prioritization. Both areas have been studied and different methods have been proposed. An excellent overview of regression test selection techniques has been presented by Rothermel and Harrold, who introduced a framework to evaluate the different techniques [19]. They defined (among others) an important evaluation criterion called *inclusiveness*, which we rely on as well as one of the primary evaluation aspects of our methods (it shows the ratio of failing test cases included in the selection). Another survey on regression test selection and prioritization has been presented by Yoo and Harman [28].

Inclusiveness is the focus of many researchers in the field of regression testing. Wong *et al.* suggested that regression test reduction techniques can lower the number of executed test cases without significantly reducing the fault-detection capabilities of test suites [27]. However, Rothermel *et al.* examined the costs and benefits of test-suite reduction techniques and their results show that the fault-detection capabilities of test suites can be severely compromised by test-suite reduction [20].

Test case prioritization techniques have also been studied thoroughly. Wong *et al.* proposed a hybrid technique combining modification, minimization and prioritization-based selection to identify a representative subset of all test cases that may result in different output behavior on the new software version [26]. Case studies on regression test prioritization have also been done. For example, Rothermel *et al.* [21] conclude that prioritization techniques can be effectively used, but the usefulness of a particular technique on a specific test suite depends on many attributes of the technique and the test suite. Our prioritization strategies use greedy algorithms, just as most other authors' as well, however there has been some research presented to use search algorithms instead [17].

The use of impact analysis for regression testing can be found in various forms in the literature; see for an overview Rothermel and Harrold for instance [19], which lists, among others, program dependence based methods that our method is also based on. There are also some simpler methods that take into consideration only close relationships [25]. Regression test selection completeness, *i. e.* safety, has been especially the topic of various researchers [4], [19], which is one of the reasons impact analysis is considered for this application.

III. Background

A. Static Execute After and dependence clusters

The Static Execute After relation is defined as follows. For program elements (procedures, classes, statements, etc.) f and g, we say that $(f, g) \in$ SEA if and only if it is possible that any part of g is executed after any part of f in any one of the executions of the program. The SEA relation is formulated as follows:

Definition (SEA relation):

$$\text{SEA} = \text{CALL} \cup \text{SEQ} \cup \text{RET}$$

where

$$\left.\begin{array}{lll} (f,g) & \in & \text{CALL} \\ (g,f) & \in & \text{RET} \end{array}\right\} \iff f \text{ calls } g \text{ (or } g \text{ returns into } f\text{)}$$

$$(f,g) \in \text{SEQ} \iff \exists h : f \text{ returns into } h, \text{ then } h \text{ calls } g$$

It can easily be seen that the SEA relation covers all possible cases when one procedure can be called after the other. Computing the SEA relation actually means following all possible control flow paths from a procedure to the rest of the system. Also, it is more convenient to consider the reflexive closure of this relation, since any change in a particular procedure can (conservatively) affect any other part of it from an impact analysis viewpoint. In the following, we consider procedures (functions and methods in C++) exclusively, and for any procedure the set of procedures that it is related to will be called its *SEA set*.

The computation of the SEA relation is based on the *Interprocedural Component Control Flow Graph (ICCFG)* [2], a program representation that contains sufficient information to extract the required relations, while being much smaller and simpler than other graphs including the System Dependence Graph [12] used for slicing.

We analyzed the precision of the SEA relation compared to static slicing and showed [13], [14] that the SEA relations can be a good approximation of static slices at the procedure level. The precision values demonstrated that there is only a small amount of additional dependences produced by the SEA method due to its conservative nature. Since SEA does not produce false negatives, compared to slices, we always get 100% recall.

As mentioned earlier, (slice-based) dependence clusters were found to be detrimental to program comprehension. Originally, dependence clusters were defined at the instruction level, but it is possible to define them considering higher granularity of a program using a suitable "depends on" relation. In particular, it is possible to use SEA as the basis of dependence, but as SEA has a representation that contains procedures as units, a SEA-based dependence cluster analysis will be based on procedures.

Definition (SEA-based dependence cluster):

> A SEA-based dependence cluster of a program is a maximal set of procedures in which any two have the same (reflexive) SEA sets.

Informally, a SEA-based dependence cluster is a maximal set of procedures that each depend on the other in the ICCFG program representation, *i. e.* each can be executed after (some part of) each other. It is important to realize that the cluster is not necessarily identical to the SEA sets of the individual procedures; the cluster can have at most as many elements as the (common) SEA sets. Practically, this definition can be substituted [6] by checking only the sizes of the dependence sets as the comparison basis instead of the sets themselves. Of course, it can happen that by chance two or more different dependence sets have the same size, and this way false clusters can be formed, however it turned out that the sizes are a good approximation in practice. In this work, we follow this approach when comparing SEA-based dependence clusters.

B. SEA-based impact analysis experiment setup

In this research we used a very pragmatic way to assess the applicability of the SEA-based impact sets: we used real defect data based on historical regression test execution results, and compare them to the respective changes that introduced and later eliminated the defects. We then verified whether the impact sets of the failure introducing changes contained the modifications at the fixing revisions (the *prediction capability* of the method). In earlier work [15], we used this approach and performed initial experiments with a limited data set from the WebKit system. In the present paper we extend these measurements with additional data and various improvements in the methods as explained later.

Figure 1 serves as a guide to summarize our approach to evaluate the SEA method in a real-world environment. The revisions of the system under examination are denoted along horizontal lines. We can examine the differences in subsequent revisions to arrive at a set of procedures that were modified

from one revision to the next. We depict these sets between two pairs of revisions, $revision_m$ and $revision_{m+1}$, then later between $revision_n$ and $revision_{n+1}$.

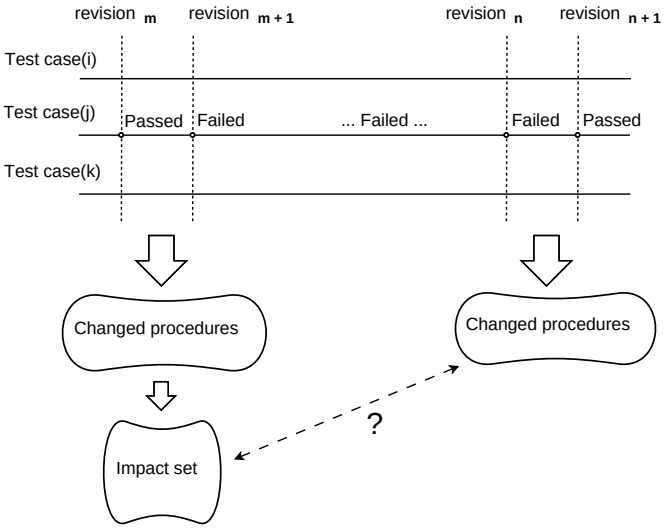

Figure 1. Verifying impact analysis prediction capability using real defects

The test cases of the system under examination can be seen vertically. All test cases are run on every revision to find out whether any regression errors have been introduced by the latest modifications. The outcome of running a test case can be either $Passed$ or $Failed$. Let us consider a scenario in which there is a test case tc_j that produces the following outcomes: $Passed$ in $revision_m$, then $Failed$ a number of times from $revision_{m+1}$ up to $revision_n$, then $Passed$ again in $revision_{n+1}$. In this scenario we can assume that the changes made between $revision_m$ and $revision_{m+1}$ are responsible for the failed test case tc_j. The error that was introduced in $revision_{m+1}$ is worked on by the programmers, then it is corrected in $revision_{n+1}$, when test case tc_j passes again. Our hypothesis is that the impact set of the modified procedures at the time the error was introduced in $revision_{m+1}$ contains the procedures that were modified between $revision_n$ and $revision_{n+1}$. If that hypothesis is found to be correct, then we would have a strong evidence that impact analysis in general and the SEA algorithm in particular is very useful in predicting where developers should look for errors in their code.

C. Evaluation environment: WebKit

Our evaluation environment is WebKit [23], a layout engine that renders web pages in web browsers, used in Google Chrome or Nokia mobile devices for example. WebKit contains about 1.9 million lines of C++ code (this amounts to about 86% of the whole source code) and it has a relatively big collection of regression tests, which helps developers keep code quality. The regression test suite consists of nearly 25 thousand active test cases; its purpose is to maintain compatibility, standards compliance gains, and check some stability, performance and other issues. Theoretically the regression tests must pass before patches can land in the revision control system, but unfortunately this requirement cannot be met in many cases, so selective retesting is often applied. WebKit has a large development community geographically spread around the world with very lively development activity. The development environment is a typical one for such a big, distributed open source team which includes serious configuration management and strict integration rules. There is a huge body of version information and historical defect data available in the version control repository, automatic test execution logs and the issue management database. As of April 11, 2012 there are 113914 revisions, and about 90 revisions are created on average each day, all of which are followed by either a full or selective regression test execution. The issue management database contains nearly 90 thousand entries.

The above mentioned features make the WebKit system suitable for our experimental study. In our measurements we used the Qt port of WebKit called QtWebKit on x86_64 Linux platform. For the compilation we used Qt 4.7.4, the latest stable version of Qt.

D. Test selection and prioritization

Recently, we have been investigating regression testing in an industrial environment by applying one of the fundamental techniques, regression test selection and prioritization based on code coverage [3]. In particular, we experimented with different algorithms that are able to provide good failure detection rate by selecting only a (small) subset of the test cases. The algorithms favored those test cases that covered the methods which had been modified just prior to testing. If the number of these covering test cases was too high, then it was reduced further based on assigned priority values. A promising prioritization strategy was found to be to assign priorities according to code coverage percentage, *i. e.* a test case that covered more procedures has been assigned a higher priority rate.

In the present research, we extended this approach with an additional selection and prioritization algorithm, which also takes into account the clusters associated with impact sets of the modification, not only the modification itself. This is an important application of impact analysis and has been studied in the past [8], [19], however we were able to compare our impact analysis-based method to other methods that do not take impact sets into account. The actual algorithms we used are presented later in this article.

E. Research questions

We formulate three research questions which we seek answers for in the context of our experimental software system WebKit:

RQ1 What is the prediction capability of the SEA relation and what are the typical impact set sizes the algorithm produces? Clearly, we hope for a result showing that we can produce sufficiently small impact sets while maintaining high prediction capability. If this is not the case we investigate the causes and possible applications and improvements.

RQ2 We established that the SEA relation can serve as a basis to define a variant of the notion of dependence cluster. Related to this, we are interested in large dependence clusters, in particular we would like to investigate whether their presence can be demonstrated and what role they play in impact set formation.

- RQ2-A Are there any large SEA-based dependence clusters in WebKit?
- RQ2-B What is the relationship between SEA-based dependence clusters and impact sets of defect introducing changes?

Answers to this question could then support or question the significance of dependence clusters in terms of hindering the successful application of SEA-based impact analysis for regression testing and change propagation.

RQ3 Can SEA impact sets be used to improve the performance of selective regression testing and test prioritization? If the selection and prioritization improved by impact analysis outperforms other strategies based on changes only, this could support the significance of impact analysis in this application.

IV. Measurements

Initial investigations about SEA focused on how it compares to other impact analysis methods, particularly program slicing. Later we moved on to investigate its applicability in real-life situations, for large programs such as GCC and Mozilla [14], and recently we have been investigating the properties of the algorithm on a different but equally complex software system, the WebKit open source web browser engine [15].

In the current stage we solved a number of technical issues in order to be able to analyze the system automatically in a regular manner. With the data collected, we were able to measure the prediction capability of the relation, which we verified by computing impact sets of actual revision changes where regression errors were introduced, and compared them to the actual changes where the regression errors were fixed.

A. Revision pair matching

We tried the approach explained in the previous section in an earlier set of experiments [15], where the results showed that the method is applicable for the analysis of the system, and that the impact sets can predict the required changes in a fair amount of cases, but there also remained open issues for improvement. One such area was the small number of actual revision pairs taken from the WebKit revision control system that provided the bases of evaluation for prediction capability. We examined several hundred revisions, but it turned out that we could arrive at only a relatively small number of usable revision pairs that matched all the criteria.

To arrive at a suitable number of defect introducing and defect correcting revision pairs, we examined a much wider range of revisions (r79171–r112713) and used two different methods:

- An automatic method searched through the full range of revisions above and extracted change information from the version control system, as well as examined the execution logs of the regression test suite. The execution logs contain the $Passed/Failed$ status of every test case, so it is possible to identify defect introducing/correcting revision pairs this way. We examined 33542 revisions and found 477 candidate revision pairs.
- We examined the WebKit issue management database, which is based on Bugzilla [24], by manually looking for entries that reported the successful elimination of bugs. Such entries identify a defect correcting revision, then searching backwards from that point, we tried to find the defect introducing revision. There are a lot of uncertainties with this method, primarily because Bugzilla entries are often incomplete or unreliable. Nevertheless, we examined 370 bug entries, from which we managed to identify 275 candidate revision pairs in the same interval as above.

The revision pairs have to conform to a number criteria in order to be suitable for our purposes. For this reason, the initial set of revision pairs were subjected to further filtering:

- We deal with the Qt port of WebKit presently, so revisions related to other ports were dropped.
- The construction of the SEA relation necessitates the analysis of the source code of the program. Currently we only have the infrastructure to analyze C/C++ code, so we drop revisions that contain non-C++ code exclusively, as it happens when only JavaScript code, configuration files, or test cases are modified.
- Due to inherent inaccuracies in the representation of the analyzed program, sometimes there are procedures (*e. g.* instantiated template functions, some overloaded operators) that show up in change logs, but are not contained in our internal program representation, hence in the SEA relations. These unrecognizable procedures have to be eliminated from the change sets, and we occasionally had to drop whole revisions whose change set reduced to the empty set this way.
- Sometimes the correction of a bug is to cancel the changes that introduced it. Our search method correctly identifies revision pairs that represent this kind of activity, but it is not interesting from an impact analysis point of view, because the bug correcting revisions have the same change set as that of the bug introducing ones. As there is no need for change propagation in this case, we do not include these revision pairs in our experiments.

The two methods for pair determination produced a different set of successful matches, summarized in Table I. We combined the results and used the 240 identified revision pairs as a basis for determining the prediction capability of the Static Execute After relation.

B. Full SEA set computation

We used the SEA relation defined over the set of procedures of a program. Typically, impact analysis requires the

Table I
REVISION PAIRS FOR IMPACT ANALYSIS EXPERIMENTS

	Automatic	Manual	Total
Examined entries	33542	370	–
Revision pairs identified	477	275	752
Revision pairs after filtering	161	79	240

computation of the impact sets for members of a small set of procedures, the change set of a revision. This is feasible and efficient enough to enable us to incorporate these computations into the build and test processes of a real system. However, occasionally we need the impact sets of all procedures in a program, to determine the dependence clusters for example. Unfortunately the computation of such a complete relation might take a long time indeed as there are approximately 92000 procedures in WebKit, and it takes a few hours to compute the full SEA relation for such a large set on a typical workstation (see Table II). This time requirement prohibits the computation of the complete SEA relation for every revision, so we resort to computing it for only a few selected revisions at most. Fortunately the set of procedures changes very little from revision to revision (approximately 20 procedures change on average), so the complete SEA relation computed on a specific revison can be used for a number of neighbouring revisions as a good approximation.

C. Improved selective regression tests

Our last research question deals with the extension of a test selection and prioritization method for WebKit regression tests [3] with SEA-based impact analysis results. Initially, we determined the full code coverage relationship between procedures and test cases, which will be the basis for test selection. Then, for the set of changed procedures at a given revision we selected those test cases that cover any of the changed procedures. Prioritization was then performed on this initial set of test cases by calculating various attributes about the test cases and assigned an "award" value based on them. Once we have the ordering based on the award values, we set a fixed threshold to select only the first N tests for execution. We experimented with different N values to get a picture of what threshold would be small enough while maintaining reasonable inclusiveness ratios. Two notable values, 1% and 20% of the test cases, were used as the comparison basis. The effectiveness of the method was checked by using two measures: inclusiveness, that shows how many failing tests are included in the selection and selection size in terms of percentage of the total number of test cases.

We developed various prioritization strategies and compared them to find out which provides the best inclusiveness with the same number of tests. As a baseline, we also used random prioritization to find out whether any of the more advanced prioritization strategies shows significant difference. For the experiments in the present work we chose to use the most traditional prioritization strategy (named *General*), which favoures test cases that cover the most procedures besides the changed ones. Here, the assumption was that test cases with higher overall coverage are better.

We improve the General strategy in such a way that we prioritize test cases not by how much they cover from *all* the procedures but from those procedures that are members of some of the large dependence clusters. The idea behind this strategy is that if clusters capture some important computation and many changes are applied to it, then tests that are somewhat specialized to cover mostly the clusters could be more effective. Other similar strategies are possible as well (*e. g.* extending the change set with the impact set), with which we plan to deal with in the future.

D. Measurement tools overhead

We can observe that SEA impact analysis incurs some time overhead compared to the original build process.

The basis of our experiment is the analysis of the relevant revisions (those that contain relevant changes at procedure level) of the system, and the computation of the corresponding impact sets. In parallel to the build process, we compute the ICCFG graph. Additionally, we need to execute the regression tests as well. In Table II, we can see the times typically required for the mentioned steps for one revision (on an Intel Xeon X5670 (12 core 2.93 GHz) machine with 96 GB RAM).

Table II
APPROXIMATE EXECUTION TIMES OF THE ANALYSES

Activity	Time (depends on revision)	
Build	20	minutes
Build with static analysis	1.5–2	hours
Regression tests	16–20	minutes
Change set SEA computation	1–10	minutes
Complete SEA computation	5	hours
Prioritization	under 1	minute

As it can be seen, the time required to compute the static information increases the build time, but it is still much better than any more accurate analysis method such as computing program slices based on a dependence graph. In fact, many other algorithms or tools would probably be unable to complete the analysis at all.

The time required for computing the SEA sets varies in a wide range, which is obviously due to the different number of changed procedures per revision. The whole analysis also includes the execution of additional tools such as the extraction of the changed procedure names for example, but the costs of these are negligible.

V. RESULTS

A. Impact analysis on WebKit

The first step to perform the impact analysis experiments was to determine the potentially interesting revision pairs, as explained in the previous section. For the final 240 revision pairs we performed ICCFG computation on the first element of each pair (the failure inducing revision). SEA sets were then computed for each changed procedure at these revisions, which resulted in the total of 3792 sets.

We computed individual impact sets for the changed procedures in order to be able to compute the prediction ratios for

individual procedures, but we also computed a union of these SEA sets to show overall percentages for the revisions.

To answer our first research question (RQ1) we essentially needed a series of two measures: the sizes of the impact sets and the corresponding prediction capabilities. Both measures can be expressed in percentage, relative to the program size and the ratio of correctly identified procedures, respectively.

We present the data about set sizes and prediction on an XY-plot in Figure 2. This plot contains 3792 data points which correspond to the individual procedures at the failure revisions from the identified pairs (x axis is the prediction, y is SEA set size).

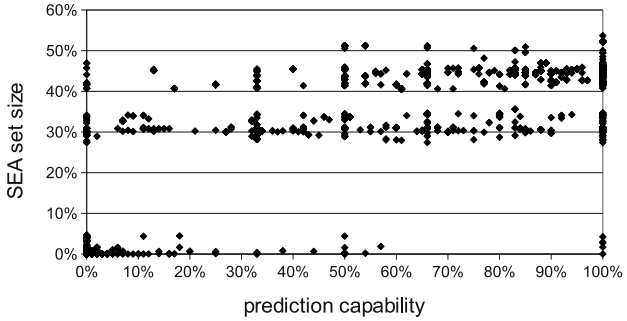

Figure 2. Relating prediction capability and impact set sizes for individual procedures.

We can observe a number of different things on this plot. First, it would be beneficial to have many small impact sets with high prediction (towards the lower right corner). If we produced a large number of randomly selected subsets of the procedures in different sizes, theoretically the expected values of this data would be by the diagonal (assuming uniform distribution of the selection and the defects). Therefore anything below the diagonal is good, and it can be determined that 74% of the data points are below the diagonal, so we can conclude that the results are promising as far as prediction capability is concerned. We can also observe that a lot of impact sets that produce good prediction are pretty large, but more interestingly there are two clearly distinguishable vertical stripes around 30% and 45%, which hint at the existence of large dependence clusters. We will discuss impact set sizes and their relationship to dependence clusters later.

There is, however, an issue with investigating the impact sets this way, using individual procedures. Namely, we cannot identify which procedure(s) in a change set actually caused the new failure, nor the fixing procedure(s), so it is a better comparison if we calculate impact sets together for all procedures in a change set (by computing their union). Figure 3 shows a similar XY-plot but with single data points per revision pair (there were 240 data points for this graph). An obvious difference is that we get fewer results with low prediction capability, which can be explained by the fact that we are using unions of impact sets, so if at least one of the sets captures any of the changes we get a positive result. For the following two figures we also used this revision-based data set.

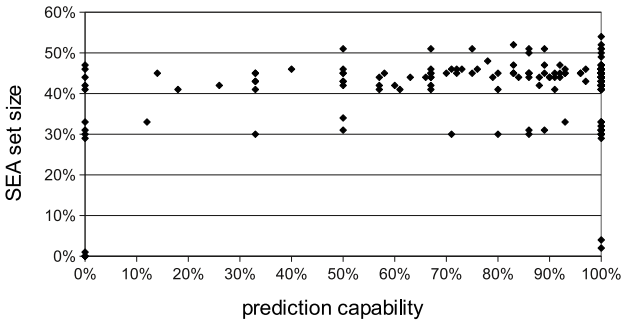

Figure 3. Relating prediction capability and impact set sizes for complete change sets.

The prediction numbers vary greatly within the whole range 0-100%; the average is 83.9% with a deviation of 27.7%. Figure 4 shows the distribution for this data with one hundred value ranges. There are several cases where the prediction is 0% or close to it, and we can observe different values at all of the ranges, but in most of the cases the prediction was high, mostly 100%. Low prediction values, even complete misses are expected to occur in a few cases for a number of reasons:

- source code analysis is not completely accurate
- procedures in the impact set may get deleted eventually
- defects may be corrected by introducing new procedures, which cannot be predicted

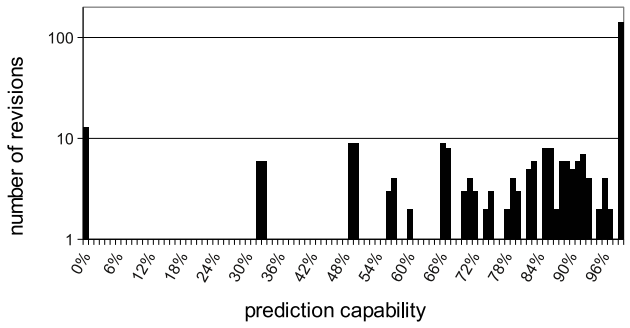

Figure 4. Prediction histogram (number of revisions shown on logarithmic scale)

Based on these data, we can conclude that overall the prediction was quite successful, albeit in many cases with a relatively large impact set.

B. Change sets, impact sets, and clusters

We have seen that the SEA set sizes were not evenly distributed but they showed some grouping behavior around specific values, which leads us to the question of dependence clusters in WebKit. Considering the sizes of set of all impact sets, more than half of them were below 1% of the system size, *i. e.* the number of all procedures in the system. The average size of impact sets was 17203, which is just below 19% of the system size.

For answering the first part of our second research question (RQ2-A) we first determined the most visible dependence clusters. We started off by visually investigating the Monotone Size Graph based on computing SEA impact sets for all procedures in the system (Figure 5, revision 91555 was used for this purpose). By visual inspection we identified three clearly distinguishable clusters that we marked by, from left to right, Cluster_1, Cluster_2 and Cluster_3.

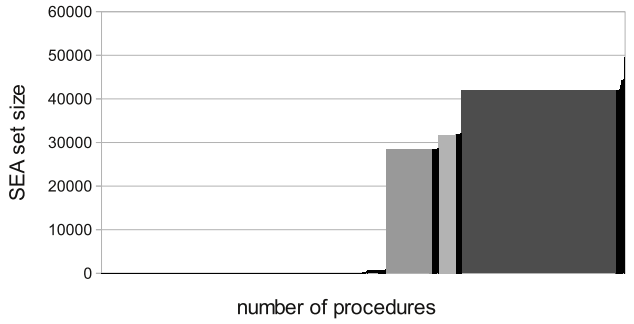

Figure 5. MSG graph of SEA impact set sizes. Grey boxes represent the three clusters, black regions are non-cluster sets.

To obtain the graph shown in the figure we performed an automatic processing of the raw data in the following way:

- We did not consider impact sets smaller than 1% of the system as candidates for inclusion in any cluster. Impact sets of the same size were expected to be equal if their size is large enough, so small ones had to be filtered out in order to get reliable results. We checked the three largest groups of impact sets of the same size and found that the elements of each group were equal, so "same size" turned out to be a perfect indicator of equality for SEA impact sets, similarly to that of PDG-based slices [6].
- Next, we allowed a tolerance of 0.1% in the size of the impact sets (about 92 procedures) to belong to the same cluster. The impact sets of size within this tolerance have most of their elements in common, so it is reasonable to regard them as "equal" for the purposes of the present investigation.

This way, we can partition the MSG into 7 distinct regions, the three clusters and the regions around them. We investigated the sizes of the clusters and the corresponding SEA set sizes in detail. The first two columns of Table III show these sizes both in the number of impact sets of these regions and in percentage relative to all impact sets (rows are listed in the order of increasing impact set sizes). The largest cluster takes almost 1/3 of all impact sets while the sum of all three is 41.53%, which means that almost half of all impact sets reside in one of the large clusters. The last column of the table shows the impact set sizes for these three clusters. Since there are about 92000 procedures in WebKit we can conclude that although there are a lot of large impact sets, the largest one contains only about half of the whole program.

The second part of this research question (RQ2-B) dealt with the defect correcting change sets captured by impact analysis. In other words, we wanted to find out what is typical to those changes that correct failures: are they mostly elements of impact sets belonging to a cluster or not? For this, we filtered the list of all impact sets to those which captured at least one fixing change at the corresponding failure fixing revision. There were 2021 such procedures altogether. The third and fourth data columns of Table III show how many of these filtered impact sets belonged to the identified regions of the MSG. We can observe that in terms of percentage all clusters contain more impact sets from the filtered set, most notably Cluster_3, which doubled. Altogether, 77.93% of defect predicting impact sets belong to some of the big clusters. This difference to the overall percentage of 41.53% is mostly due to the relatively few small impact sets kept by this filtering (there are more large predicting sets than small ones).

To summarize, almost half of all of the impact sets belong to big clusters while significantly more, over 3/4 of failure predicting impact sets belong to this category.

Table III
DEPENDENCE CLUSTER SIZES

	All procedures	Percent	Predicted procedures	Percent	SEA size
Region_1	49623	54.42%	360	17.81%	–
Cluster_1	8028	8.80%	180	8.91%	28395
Region_2	1127	1.24%	28	1.39%	–
Cluster_2	2981	3.27%	156	7.72%	31720
Region_3	1040	1.14%	15	0.74%	–
Cluster_3	26864	29.46%	1239	61.31%	41892
Region_4	1530	1.68%	43	2.13%	–
Sum	91193	100.00%	2021	100.00%	–

C. Improved regression test selection and prioritization

The third research question (RQ3) concerned the performance of selective regression testing. Our hypothesis for the impact set-based improvement was that clusters could represent some distinct computation, and if there are test cases present in the system specialized for these computations then prioritizing based on the clusters' coverage could be more effective than a general whole program coverage based approach.

We performed test selection and prioritization with the method described in the previous section using the three clusters individually and using their union as well. Altogether 70 revisions have been measured this way (as in our earlier experiments [3]), and the average inclusiveness values were calculated for two prioritization thresholds, 1% and 20%. There were about 25000 test cases in this period, so these thresholds mean a maximum of 250 and 5000 test cases selected, respectively (less if the change based selection is smaller). Table IV shows the results of these measurements.

Table IV
REGRESSION TEST PRIORITIZATION INCLUSIVENESS

TCs	General	Cluster_1	Cluster_2	Cluster_3	Cluster union
1%	5.13%	5.64%	8.21%	4.36%	4.36%
20%	40.00%	41.54%	61.79%	40.00%	40.26%

The original inclusiveness of the General strategy was 5.13% and 40.00% for the two thresholds. We could not observe any significant improvement with the cluster based prioritization, except for the case when we used Cluster_2 for this purpose. The union of the clusters did not show any improvement either.

We investigated what could be the reason for Cluster_2 showing an inclusiveness improved by 55–60%? To do this, we investigated the relationships between the different final prioritized test lists, whether there were any significant differences between them. We summarize our findings in Table V, where T_1, T_2 and T_3 denote the final test lists prioritized using Cluster_1, Cluster_2 and Cluster_3, respectively.

Table V
PRIORITIZED TEST SETS' RELATIONSHIPS

TCs		S_1	S_2	S_3	S_4	S_5	S_6	S_7
1%	(228)	30	48	35	6	1	19	173
20%	(3531)	869	1917	879	173	163	1211	1278

where

S_1=$|T_1 \setminus (T_2 \cup T_3)|$ S_2=$|T_2 \setminus (T_1 \cup T_3)|$ S_3=$|T_3 \setminus (T_1 \cup T_2)|$
S_4=$|(T_1 \cap T_2) \setminus T_3|$ S_5=$|(T_2 \cap T_3) \setminus T_1|$ S_6=$|(T_1 \cap T_3) \setminus T_2|$
S_7=$|T_1 \cap T_2 \cap T_3|$

For the two thresholds, we show the number of elements in the set intersection regions among the test sets for the three prioritization strategies (average values of the 70 measurements are shown). The numbers in parentheses after the thresholds show the average selection sizes. If we focus our attention on the strategy based on Cluster_2, it can be clearly seen that its test set seems to be much more unique than that of the other two clusters. First, set S_2 is bigger than the other two by about 50% for the smaller selection and 1.2 times for the bigger one. Also, the intersection of T_1 and T_3 (S_6) is much bigger than the other two pairwise intersections with T_2.

We did not verify the actual causes for these findings but we speculate that Cluster_2 could represent some kind of a special computation with its dedicated test case subset, so this could explain why prioritizing based on this cluster could provide higher inclusiveness than using a general approach. In the future, we plan to investigate the internal structure of this cluster and the related test cases to verify this explanation and would like to compare our approach to others [28].

D. Threats to validity

There are several threats to the validity of the presented method and experiment, apart from the obvious ones that regard the potential defects in the impact analysis and other supporting measurement tools. Parts of the toolset (including the SEA implementation) has been used, however, in a number of previous research projects so we are confident in the validity of the implementation.

There are some known deficiencies of the measurement tools nevertheless. Currently we can analyze only C/C++ code, furthermore there could be some issues with certain language constructs like template instantiation. We tried to minimize the effect of these weaknesses by removing all those measurement points (complete revisions) from the investigation that could be compromised by this limitation (as discussed earlier).

When verifying the prediction capability rate of the impact analysis, we could not precisely know which atomic changes caused the failure and the fix since

1) we aggregated the changes and the dependences to procedure level, and
2) we could not take into account that a revision may implement more than one functionality at the same time.

This means that we assumed that all changes at a given revision belong to a failure inducing change or a fix.

Our method for finding suitable revision pairs neglects the fact that the fix can be done in more steps during the failing period and the final fix made when the status changes to $Passed$. The Bugzilla-based matching method was mostly manual so this way we could of course miss interesting revisions, and the resulting pairs could not be complete in any sense and it is not clear to how much extent the results derived from them are generalizable.

The analysis of the results in connection to the clusters, defects and testing are based on only one system with only three clusters in it. Hence these results should be generalized to other systems with caution. However, the basic findings about the presence of the clusters are aligned with other research.

Finally, as with all other static code analysis techniques, our method cannot guarantee complete coverage of possible dependences due to various issues related to the dynamic nature of the languages, and other semantic, conceptual or logical dependences in the software system.

VI. CONCLUSIONS

In this paper we dealt with the Static Execute After (SEA) relation, a static code dependence analysis method. Our investigations centered around change impact analysis, dependence cluster identification, and regression test selection for the WebKit software system. We constructed three research questions for which we sought and gave answers for. First we collected a sizeable amount of historical data about actual defects and the corresponding changes (defect introduction and fixing), and assessed the applicability of the SEA relation to change impact analysis of the defect introducing changes with respect to the required changes to fix the defect (prediction capability). We found it to be satisfactory in general (with 83.9% prediction capability on average), albeit the produced impact sets were often quite large (RQ1). Although precise impact sets are often useful even if their size is large (*e. g.* for automated regression test suite reduction), they are detrimental when manual processing may be necessary (*e. g.* in change propagation). This observation led us to investigate the dependence clusters that are formed during SEA computations. It turned out that there are large clusters in the code (RQ2-A) and they are indeed responsible for large impact sets (42% of the sets are in such clusters whose sizes are about 1/3 of the program size), so impact set size reduction methods should consider dependence clusters. The examination of the relationship between defect predicting

change sets and dependence clusters showed that such change sets even more often, in 78% of the cases, reside inside clusters (RQ2-B). Finally, we worked on improving formerly defined regression test selection and prioritization methods by taking into account SEA impact sets of changes (those belonging to the clusters, essentially) in addition to the changes themselves. We were able to improve inclusiveness of the test prioritization algorithm by over 55% with one of the impact sets belonging to a specific cluster (RQ3). This is a preliminary result and needs further investigation to find out the universality of the effect, reasons for it and possible practical applications.

There are numerous possible directions for future work. Our ultimate goal is to provide impact analysis and test optimization tools for WebKit developers, so we plan to enhance the implementations further to be integrated into the development processes. Our research findings in this paper are expected to be generalizable to other systems as well, so we plan to verify them in other real life, industrial environments. Further examinations of cluster properties seem to be interesting, including examining the relationships between clusters and their SEA sets, finding causes for their formation and opportunities for removal. To be more suitable for other impact analysis applications, we plan to develop methods to reduce the size of the impact sets. The research on regression test selection and prioritization is clearly another interesting topic for further research. We plan to analyze the relationships between the internal structure of the clusters, the associated tests and the resulting inclusiveness. We have further ideas for additional prioritization strategies, which we also plan to implement and evaluate.

ACKNOWLEDGEMENTS

The authors would like to thank Attila Kerék, László Langó, Csaba Osztrogonác, John Taylor and Béla Váncsics for their valuable supporting work for this research. This research was supported by the Hungarian national grants GOP-1.1.1-11-2011-0049 and OTKA K-73688.

REFERENCES

[1] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in *Proceedings of the 33rd ACM SIGSOFT International Conference on Software Engineering (ICSE)*, 2011, pp. 746–765.

[2] Á. Beszédes, T. Gergely, J. Jász, G. Tóth, T. Gyimóthy, and V. Rajlich, "Computation of static execute after relation with applications to software maintenance," in *Proceedings of the 2007 IEEE International Conference on Software Maintenance (ICSM'07)*, 2007, pp. 295–304.

[3] Á. Beszédes, T. Gergely, L. Schrettner, J. Jász, L. Langó, and T. Gyimóthy, "Code coverage-based regression test selection and prioritization in the WebKit system," 28th IEEE International Conference on Software Maintenance (ICSM'12). Accepted for publication, manuscript available upon request.

[4] J. Bible, G. Rothermel, and D. S. Rosenblum, "A comparative study of coarse- and fine-grained safe regression test-selection techniques," *ACM Transactions on Software Engineering Methodologies*, vol. 10, no. 2, pp. 149–183, 2001.

[5] D. Binkley and M. Harman, "Identifying 'linchpin vertices' that cause large dependence clusters," in *Ninth IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM '09)*, 2009, pp. 89–98.

[6] ——, "Locating dependence clusters and dependence pollution," in *Proceedings of the 21st International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 177–186.

[7] D. Binkley, M. Harman, Y. Hassoun, S. Islam, and Z. Li, "Assessing the impact of global variables on program dependence and dependence clusters," *Journal of Systems and Software*, vol. 83, no. 1, pp. 96–107, 2010.

[8] S. A. Bohner and R. S. Arnold, Eds., *Software Change Impact Analysis*. IEEE Computer Society Press, 1996.

[9] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "JRipples: A tool for program comprehension during incremental change," in *IWPC*, 2005, pp. 149–152.

[10] Á. Hajnal and I. Forgács, "A demand-driven approach to slicing legacy COBOL systems," *Journal of Software: Evolution and Process*, vol. 24, no. 1, pp. 67–82, Jan. 2012.

[11] M. Harman, D. Binkley, K. Gallagher, N. Gold, and J. Krinke, "Dependence clusters in source code," *ACM Transactions on Programming Languages and Systems*, vol. 32, no. 1, pp. 1:1–1:33, Nov. 2009.

[12] S. Horwitz, T. Reps, and D. Binkley, "Interprocedural slicing using dependence graphs," *ACM Transactions on Programming Languages and Systems*, vol. 12, no. 1, pp. 26–61, 1990.

[13] J. Jász, "Static execute after algorithms as alternatives for impact analysis," *Peryodica Politechnica*, pp. 163–176, 2008.

[14] J. Jász, Á. Beszédes, T. Gyimóthy, and V. Rajlich, "Static execute after/before as a replacement of traditional software dependencies," in *Proceedings of the 2008 IEEE International Conference on Software Maintenance (ICSM'08)*, 2008, pp. 137–146.

[15] J. Jász, L. Schrettner, Árpád Beszédes, C. Osztrogonácz, and T. Gyimóthy, "Impact analysis using Static Execute After in WebKit," in *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, 2012, pp. 95–104.

[16] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, pp. n/a–n/a, 2012. [Online]. Available: http://dx.doi.org/10.1002/stvr.1475

[17] Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization," *IEEE Transactions on Software Engineering*, vol. 33, no. 4, pp. 225–237, 2007.

[18] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An Information Retrieval Approach to Concept Location in Source Code," in *The 11th IEEE Working Conference on Reverse Engineering (WCRE'04)*, 2004.

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

[20] G. Rothermel, M. J. Harrold, J. von Ronne, and C. Hong, "Empirical studies of test-suite reduction," *Software Testing, Verification and Reliability*, vol. 12, no. 4, pp. 219–249, 2002.

[21] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, pp. 929–948, 2001.

[22] B. G. Ryder, "Constructing the call graph of a program," *IEEE Trans. Softw. Eng.*, vol. SE-5, no. 3, pp. 216–226, May 1979.

[23] "The WebKit open source project," http://www.webkit.org/, last visited: 2012-05-03.

[24] "WebKit Bugzilla homepage," https://bugs.webkit.org/, last visited: 2012-05-04.

[25] L. White, K. Jaber, and B. Robinson, "Utilization of extended firewall for object-oriented regression testing," in *Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'05)*, 2005, pp. 695–698.

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

[27] W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in *Proceedings on the 17th International Conference on Software Engineering*, 1995, pp. 41–50.

[28] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[29] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 429–445, 2005.