A Seismology-inspired Approach to Study Change Propagation

Salima Hassaine*, Ferdaous Boughanmi§, Yann-Gaël Guéhéneuc§, Sylvie Hamel*, and Giuliano Antoniol§
*DIRO, Université de Montréal, Québec, Canada
Email: {hassaisa,hamelsyl}@iro.umontreal.ca
§DGIGL, École Polytechnique de Montréal, Québec, Canada
Email: {ferdaous.boughanmi,yann-gael.gueheneuc}@polymtl.ca, antoniol@ieee.org

***Abstract*—Change impact analysis aims at identifying software artefacts that are being affected by a change. It helps developers to assess their change efforts and perform more adequate changes. Several approaches have been proposed to aid in impact analysis. However, to the best of our knowledge, none of these approaches have been used to study the scope of changes in a program. We present a metaphor inspired by seismology and propose a mapping between the concepts of seismology and change propagation, to study the scope of change propagation. We perform three case studies on Pooka, Rhino, and Xerces-J to observe change propagation. We use ANOVA and Duncan statistical tests to assess the statistically significance of our observations, which show that changes propagate to a limited scope.**

***Keywords*-Software evolution; Change Impact Analysis; Change Propagation; Earthquake Forecasting;**

I. Introduction

Although object-oriented programming has met great successes in modeling and implementing complex programs, developers face problems with maintenance [1]. In particular, making changes without understanding their effects can lead to poor effort estimation and delays in release schedules because of their dire consequences, *e.g.*, the introduction of bugs [2], [3]. Therefore, both managers and programmers must be aware of the ripple effects caused by a change. Thus, they need help to identify the classes that must be changed to perform maintenance changes more accurately. Change impact analysis aims at identifying software artefacts being affected by a change; it provides the potential consequences of a change and estimates the set of artefacts that must be modified to accomplish a change [4].

Existing approaches for change impact analysis are based on class dependencies and use static, dynamic, and–or textual analysis [4]–[7]. However, in object-oriented programs, the relationships between classes make change impact difficult to anticipate because of possible hidden propagation [8]. In this case, historical analysis [9]–[12] of data from software repositories provides useful information that complement static and dynamic analyses. Such techniques learn change impact relations based on the co-changes of software artefacts within a change-set. However, they may fail to capture how changes are spread over "space" (*e.g.*, in class diagram) and through class levels (*i.e.*, the distance between co-changed classes). Consequently, they could not help developers prioritise their changes according to the forecast scope of changes, which can lead to poor effort and cost estimations.

To the best of our knowledge, none of these approaches have been used to study how far changes propagate from a given class to the others, *i.e.*, whether two co-changing classes are in direct relation or are separated through a long chain of relationships. In particular, if we analyse the relationships chain of a changed class, we may have the intuition that the classes impacted by a change are near the changed class. However, in some cases, the actual classes that must be modified are far away from the changed class. If these classes are not considered before implementing a change, bugs will occur.

The following *motivating example*, illustrates the difficulty that developers face in identifying change propagation: the bug ID200551[1] reports a bug in Rhino[2], that was introduced by a developer when he implemented a change to class `Kit` and missed a required change to class `DefiningClassLoader`. In this case, information passes from class `Kit` to class `DefiningClassLoader` through an intermediary class `ContextFactory`. Thus, a change in `Kit` should trigger a change in `DefiningClassLoader`, while the class `ContextFactory` remains unchanged.

This motivating example presents a situation where a bug was introduced by a developer who missed changing a class, that must be considered before implementing the change task. This example confirms that change propagation is difficult to anticipate between two classes separated by an intermediary class. This problem would be more difficult if the two classes (`Kit` and `DefiningClassLoader`) were separated by a long chain of relationships.

Studying the scope of change propagation could help developers prioritise their changes according to the forecast scope of changes. Understanding change propagation requires source code analysis, which is a difficult, error-prone, and expensive activity [13]. We propose an approach to change propagation analysis specifically designed to study the scope of change propagation, based on a metaphor between seismology and change impact analysis. Our approach considers changes to a class as an earthquake that propagates through a long chain of

[1]https://bugzilla.mozilla.org/show_bug.cgi?id=200551
[2]http://www.mozilla.org/rhino/

intermediary classes. It combines static dependencies between classes and historical co-change relations to study the scope of change propagation in a program *i.e.*, how far a change propagation will proceed from an "epicenter class" to other impacted classes.

In this paper, we perform a qualitative and two quantitative studies, to show the applicability and usefulness of our approach. We apply our approach on three open-source systems: Pooka, Rhino, and Xerces-J, and answer the following research questions:

- **RQ1:** Does our metaphor allow us to observe the scope of change impact?
- **RQ2:** What is the level most impacted by a change?
- **RQ3:** What is the most reachable level by a change?

We answer these research questions as follows:

- **RQ1**: Like earthquakes the change impact seems to be more severe near the epicenter class and decreases through class levels.
- **RQ2**: We applied $ANOVA$ and $Duncan-Multiple-Range$ tests, and we can conclude that: a) level 1 is the most impacted, b) level 2 is the second most impacted, but significantly less than level 1, c) levels 3, 4, 5, and 6, are significantly less impacted than level 1 and 2.
- **RQ3**: We conclude that almost change propagation stops at the level 1 and 2. But, there are some earthquakes that propagate until 3, 4, 5 or 6 level.

This paper is organised as follows: Section II summarises work related to change impact. Section III describes our metaphor and mapping. Section IV presents our approach and its implementation. Section V presents the three research questions derived from our metaphor while Section VI presents our study results and answers to the questions. Finally, Section VIII concludes and presents future work.

II. RELATED WORK

In this section, we review and discuss related work.

A. Structure-based Change Impact Analyses

Arnold and Bohner proposed several models of change propagation [4], [5]. These models are based on code dependencies and algorithms, including slicing and transitive closure, to assist in assessing the impact of changes. Dependency analysis of source code is performed using static analyses or dynamic program analyses. When performing change impact analysis with call graphs, the impact of a change in a method is the transitive closure of all callers and callees. Therefore, it can be inaccurate, by reporting false candidates that do not change and failing to estimate some classes that actually do change because the analysis is restricted to method calls.

Weiser *et al.* [14] proposed also slicing techniques to to determine all the code locations which may affect a reported location of a failure. Static slicing is typically based on data- and control-flow graphs that are computationally expensive to process and analyse and can report large slices [15]. Thus, dynamic slicing [16], [17] and probabilistic slicing [18], have been proposed in literature to reduce the size of slices. However, their analysis is expensive.

Law *et al.* [6] argued that static slicing is much more precise than transitive closure on call graphs, but it may return large sets of classes that are impacted by a change. Dynamic slicing can improve the conservative behavior of static slicing. However, it is subject to the risk of lower precision and recall as it depends on the chosen scenarios and–or executed test cases. Consequently, Law *et al.* [6] introduced a new approach to method-level change impact analysis. They used path profiling technique [7] to compress dynamic traces, then they applied PathImpact algorithm to predict dynamic change impact. Their approach can provide potentially more useful predictions of change impact than method-level static slicing in situations where specific program behaviors are the focus.

Zaidman *et al.* [19] proposed a technique for uncovering important classes in a program's architecture. They used a technique that was originally developed to identify important hubs on the Internet *i.e.*, pages with many links to "authorative" pages [20]. They verified that important classes in the program correspond to the hubs in the dynamic call-graph of a program trace.

B. History-based Change Impact Analyses

Ying *et al.* [9] and Zimmermann *et al.* [10] proposed to mine version-control systems. They applied association rules that identify logical couplings [11] between classes. A change occurring in class *A* may have an impact on another class *B* if in the past they changed together. Such historical analysis is often able to capture change couplings that cannot be captured by static and dynamic analyses.

Bouktif *et al.* [12] used a technique from speech recognition to infer cause–effect relationships from the revision histories. Their approach relies on the technique of dynamic time warping to group files with histories of changes of different lengths. The values of their approach precision and recall are higher than previous approaches [9], [10].

Girba *et al.* [21] proposed an approach, named Yesterday's Weather, to identify classes that are likely to change in the next version. This approach is based on the retrospective empirical observation that classes which changed the most in the recent history also suffer important changes in the near future.

Canfora *et al.* [22] proposed an approach based on information-retrieval techniques to derive the set of classes impacted by a proposed change request. They argued that the histories of change requests is a useful descriptor of classes when it is used for change impact analysis.

German *et al.* [23], proposed a method which determines the impact of previous code changes on a particular code segment based on a change impact graph. Given a location of failure, their method annotates the neighbours of this failure in the graph by marking the recent changes. Thus, it determines all the changed areas of the software system which affect the reported location of a failure.

C. Probabilistic Approaches

Zhou *et al.* [24] and Mirarab *et al.* [25] presented a change propagation analysis based on Bayesian networks that incorporates static source code dependencies as well as different features extracted from the history of programs and uses a sliding window algorithm to group them. Their change propagation model is able to predict future change couplings. Mirarab *et al.* [25] used Bayesian belief networks as a probabilistic tool to make such predictions systematically. Their approach mainly relies on dependency metrics calculated using static analysis and change history extracted from a version-control system.

Antoniol *et al.* [26] incorporated static source code dependencies and other features extracted from the release history of a program, such as author information. Then, they applied the LPC/Cepstrum technique to mine a version-control system for classes having evolved in the same or very similar ways. Their approach can find classes having very similar maintenance evolution histories.

Ceccarelli *et al.* [27] proposed the use of a generalisation of univariate autoregression model to capture the evolution and inter-dependencies between multiple time series. They applied the bivariate Granger causality test [28] to infer the mutual dependencies between classes, analysing the time series representing the change histories of a class A to predict the changes of another class B. Their preliminary results showed that change impact relationships inferred with the Granger causality test are complementary to those inferred with association rules.

D. Hybrid Approaches

Malik and Hassan [29] proposed the use of adaptive change propagation heuristics. These heuristics combine the use of a set of change propagation heuristics (history heuristic, containment heuristic, call use depends heuristic, code ownership heuristic). The proposed adaptive heuristics can be adapted to the current state of a programand to the varying characteristics of the different entities in the program. This adaptive heuristic uses a best heuristic table to track for each change entity the best heuristic. It uses the development replay framework [30] that uses change sets to measure the performance of change propagation heuristics.

III. The Earthquake Metaphor

We now present a mapping between the concepts of seismology and change impact analysis. We use this mapping to observe and identify the scope of change propagation, *i.e.*, how a change to a program will impact the rest of the program, using seismology techniques.

A. Seismology

Seismology is the study of earthquakes and of the propagation of seismic waves. A seismic wave, or shaking, is the vibration that occurs from the epicenter of an earthquake until a damaged site. Seismic waves propagate along the surface and through the Earth at varying speeds, depending on the types of soil through which they move. In general, shaking is most severe near the epicenter and decreases away from the epicenter, *i.e.*, more a site is near to the epicenter, more the debris are important [31].

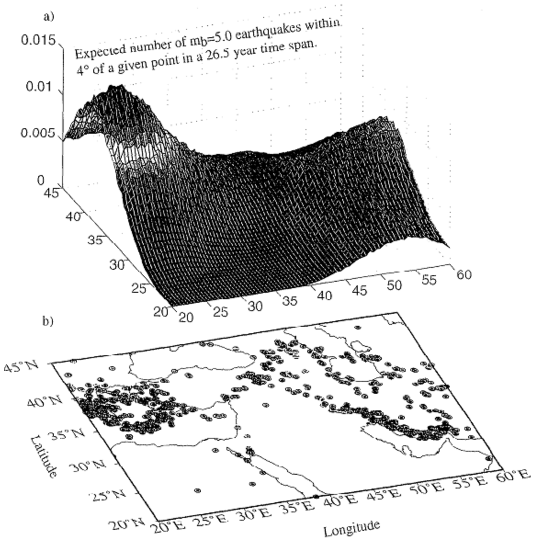


Fig. 1. Epicenters distribution in space and time. The map shows the expected number of earthquakes of a given magnitude occurring within a given radius from each point. (From [35].)

Seismology includes three main research directions:

1) **Earthquake predictions**: Creating effective approaches for precise earthquake predictions, *i.e.*, forecasting the probable timings, locations, and magnitudes of earthquakes [32], [33].
2) **Debris forecasting**: Predicting and estimating debris, or structural damage, after an earthquake, to assist debris managers in planning large scale debris removals[3].
3) **Earthquake behaviour analysis**: Studying short- or medium-term interactions among earthquakes (fore shock, main shock, after shock), and long-term behaviour of earthquakes [34].

Our approach is inspired from debris forecasting to identify where the impact is located (*i.e.*, the most risky classes). Figure 1 illustrates how the magnitude of an earthquake varies in space and time. We see that the magnitude is maximum in the epicenter and it decreases in function of the distance from the epicenter to the considered site.

B. Change Impact Analysis

Change impact analysis has two main goals: supporting the processing of changes and enabling the traceability of changes. It is important during development and maintenance to help developers in assessing their effort to implement change requests (typically, the more impacted classes by a change, the greater the effort) and in performing the most adequate changes. Thus, it limits the risk of introducing bugs

[3]http://www.calema.ca.gov/WebPage/OESWebsite.nsf/Content/88892A0B623B1F77882574270081DD56?OpenDocument

Change Impact Analysis	**Seismology**
“Important” classes	Active seismic areas
Software change	Earthquake
“Important” changed class	Epicenter
Change propagation	Seismic wave propagation
“Other” changed classes	Damaged sites
Class level	Distance from an epicenter to a damaged site

TABLE I
MAPPING BETWEEN CHANGE IMPACT ANALYSIS AND SEISMOLOGY

by clearly identifying classes that could be impacted as shown by the motivating example in Section I.

Change propagation is due to changes “moving” from one class to another through program classes. For example, if a method is renamed because of some change request, then all the classes that call this method must be modified to use its new name; in turn, other classes may change because of the changes in these methods.

C. Change Impact Analysis and Seismology

We now define a mapping between impact analysis and seismology, summarised in Table I.

In seismology, the epicenter of an earthquake is located in a most active seismic areas. To determine the epicenter (location) of an earthquake, seismologists analyse seismograms, which record the seismic activities for a given areas.

In change impact analysis, any class is potentially subject to changes. Yet, changes to “important” classes will impact more a program than “peripheral” classes. A class can be characterised as important according to different measures. If an “important” class changes, then it is analogous to an epicenter in seismology. In the following Section IV, we identify important classes and also apply our approach to all the classes in three programs.

In seismology, seismic waves propagate from the epicenter of an earthquake until a damaged site, depending on the types of soil through which they move. Shaking is usually most severe near the epicenter and drops off away from the epicenter, *i.e.*, more the infrastructures are near the epicenter more the debris is important.

In change impact analysis, changes propagate from the epicenter class to the impacted class, depending upon the class relationships, other logical couplings among these classes, and their distance (called class level) to the epicenter class. We define the distance between classes using the concept of class level: with respect to a class A, a class B is in the level 1, if it is in direct relation with A (inheritance, call, etc), and in the level 2, if it is related to A through an intermediary class C.

Change propagation is analogous to seismic waves propagation in seismology. As in seismology, we assume that change impact is most severe near the epicenter class, *i.e.*, more the classes are near to the epicenter class more the impact is important. Yet, to the best of our knowledge, no previous work reported observations on the propagation of changes through the programs, which is the aim of our metaphor and the subject of our empirical study. Using this mapping between impact analysis and seismology, we now present an approach to identify the classes impacted by a change to a given class. This approach is complementary to previous work. It uses structural and historical analysis with the specific aim to study the scope of change propagation.

IV. APPROACH

This section presents our approach, each step will be described in detail below. We can apply our approach on any class. Specifically, as a developer starts changing a class, it could analyse the change and recommend additional classes for consideration. Also, before performing any change in a program, it could help a developer to identify critical classes that are regularly changed and could have an impact on the program.

Our approach consists of three steps. Given an object-oriented program, first, we compute metric values to rank classes according to their importance and identify “epicenter classes ”. Second, given an epicenter class, we compute the “distance” from the epicenter class to each class of the program, using a bit-vector algorithm: all classes that are directly connected to the epicenter class are assigned to the level 1, the classes that have a direct relationship with one of these classes are put in level 2, and so on. Third, we use a time window of duration T and collect the set of classes that are modified after any change of the epicenter class and within T. Finally, we report the set of classes that are involved in any change and their number of changes.

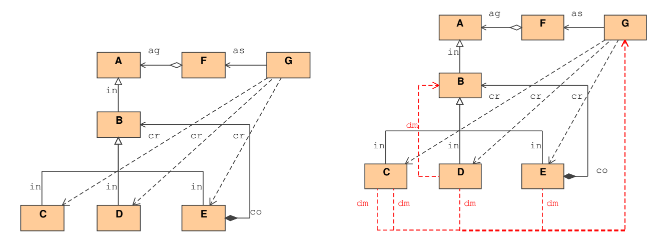
A. Step 1: Measuring Class Importance

In this Section, we identify the most “important” classes in a program using a combination of history-based and PageRank-based metrics.

1) History-based Metric: We define a history-based metric as the number of all commits related to a given class in the entire history of a program, extracted from the program version-control system. This measure represents the quantity of changes to a class. It does not consider the size and the type of a change, which are future work.

We use Ibdoos, our framework for the analysis of control-version systems, to compute the numbers of changes to every classes in a program. Ibdoos provides parsers for various format of change logs, including CVS, Git, and SVN, and stores all commits in a database, then we query this database to obtain the numbers of changes per classes. We define the number of changes to a class as $h(c)$ for any class c.

2) PageRank-based Metric: PageRank [36] is one of the main algorithms used by the Google search engine to measure the relative importance of Web pages. PageRank takes backlinks into account and propagates the PageRank value of a page through links: a page becomes important if the sum of the values of its backlinks is high. Using PageRank, we can measure the relative importance of each class in a program. A class is important if it has incoming calls from other (preferably important) classes. If a class is the only reference of a very important class, it might be ranked higher than another class in relationship with low-ranked classes. We use the algorithm previously developed by Kpodjedo [37] to



(a) UML-like model (b) Eulerian model

```
A in B in D dm B in E co B in C dm G cr C dm G cr D dm G cr E dm G as F ag A
```

(c) String representation of the Eulerian model

Fig. 2. Representations of a simple example program (from [38]).

compute for each class c, the PageRank value $pr(c)$. Then, we rank all classes in descending order of their $pr(c)$ and we compute their class rank value $r(c)$: the most important class c has the highest $pr(c)$ value.

3) Combination of the History- and PageRank-based Metrics: We combine the history- and PageRank-based metrics by dividing the rank of each class by the number of changes to the class. Thus, given two classes with equal ranks, the most important class of the two is the one with the greater number of changes, which lead to its rank to become higher than that of the other class. We thus define $rh(c) = \frac{r(c)}{h(c)}$.

The metric rh provides an assessment of the class importance in a program, taking into account both the program structure (through the PageRank-based metric) and the program history (through the history-based metric). The combination rh ranks the most important classes first. A lower value of $rh(c)$ indicates a lower value $r(c)$ and a higher value $h(c)$, *i.e.*, rh ranks the most important classes that are often changed. Identifying these important classes helps reveal what classes of the program are regularly evolved and should be analysed to identify their change propagation.

B. Step 2: Identifying Class Levels

We assume that change propagation depends on the "distance" between classes (called level). We represent a level as the number of the intermediary relationships between a given class and the epicenter class, *i.e.*, this distance indicates whether the two classes are in direct relation (level 1) or are separated by a long chain of relationships of length n.

We use an existing tool, PADL [39], to automatically reverse-engineer class diagram from the source code of object-oriented programs. A model of a program is a graph with nodes being the classes and edges representing the relationships between classes, see Figure 2(a). To identify direct and indirect relationships with the epicenter class, we first convert the program model into its string representation, as illustrated in Figure 2(c), using the algorithm previously developed by Kaczor [38]. Then, we apply a bit-vector algorithm [40]: we build the characteristic vectors of each token in the string representation. The characteristic vector of a token t associated with the string $s = s_1 ... s_m$, is $t = (t_1 ... t_m)$:

$$t_i = \begin{cases} 1 & \text{if } s_i = t \\ 0 & \text{otherwise.} \end{cases}$$

For the example shown in Figure 2, the characteristic vectors of tokens A, in, and B are defined as:

$$\begin{aligned} A &= \mathbf{1}\underbrace{0000000000000000}_{30}\mathbf{1} \\ in &= 0\mathbf{1}0\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{0000000}_{19} \\ B &= 00\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{00000000}_{20} \end{aligned}$$

Lets assume a class A as an epicenter class, to identify all classes that are directly connected to this epicenter class: for each class X—using conjunctions and shifts—between the characteristic vectors of A, X, and all relationships: if X is directly related to A through a relationship, then we put X in level 1. For example, to identify whether class B is directly related to class A through the inheritance relationship in, we compute:

$$\begin{aligned} (\rightarrow\rightarrow A) &= 0\mathbf{1}\mathbf{1}\underbrace{00000000000000000}_{29} \\ (\rightarrow in) &= 00\mathbf{1}0\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{000000}_{18} \\ B &= 00\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{00000000}_{20} \\ R &= (\rightarrow\rightarrow \mathbf{A}) \wedge (\rightarrow \mathbf{in}) \wedge B \\ &= 00\mathbf{1}\underbrace{000000000000000000}_{29} \end{aligned}$$

and assess whether the bit vector R is null (contains only zeros). If R is not null, then class B is in level 1 with respect to A. Once all classes of level 1 are defined, we repeat this process, considering each class of level 1 as epicenter class to identify all classes at level 2, *i.e.*, classes that have a direct relationship with a class of level 1. The same process is repeated to identify level 3, and so on.

C. Step 3: Identifying Impacted Classes

We mine the version-control system of a program to identify epicenter classes: we first define a time window T of observation as the median of time between two subsequent changes to the important epicenter class. We choose the median because it is robust to outliers. Then, we extract all the commits that happened after any change to an epicenter class and within the chosen time window T. Finally, we collect: (1) the names of all classes that are involved in any change and (2) the number of changes to each class.

We use Ibdoos to implement queries to collect the set of classes changed after any change to the epicenter class and during T. The names of all subsequently changed classes and the number of changes that these classes underwent.

V. Empirical Study Design

Following the Goal Question Metric (GQM) [41], the *goal* of this study is to show the applicability and usefulness of our

Programs	Nbr. Classes	Start Dates	End Dates	Last Version
Pooka	298	2000-01-02	2010-08-30	2.0
Rhino	132	1999-12-18	2009-01-17	$1R6.0$
Xerces	685	2005-10-12	2010-11-26	11.0

TABLE II
STATISTICS FOR THE PROGRAMS.

approach, with the *purpose* of gathering interesting observations on the scope of change propagation and confirming these observations statistically. The *quality focus* is the accuracy of the identified scope of change propagation (*i.e.*, how far the propagation proceed from a given class to the others), and also the variation of changes depending on the level of classes. The *perspective* is of both researchers and practitioners who should be aware of the scope of a change to estimate the effort required for future maintenance tasks. The observed phenomena can help for making decisions concerning the process of future software projects. The *context* consists of Pooka, Rhino, and Xerces-J. Pooka[4] is an email client written in Java, using the Javamail API. Rhino[5] is an open-source implementation of a JavaScript interpreter written entirely in Java and developed for the Mozilla/Firefox browser. Xerces[6] is an open-source family of software packages for parsing and manipulating XML. Characteristics of these programs are reported in Table II. We choose these programs because they have been studied in previous work, they are open-sources, thus we can find external information, such as bug reports.

A. Research Questions

This study aims at answering the research questions:

- **RQ1:** Does our metaphor allow us to observe the scope of change impact?
  We investigate whether it is possible to apply our approach to observe change propagation through class levels. We perform a qualitative study to confirm our observations of change propagation, using external information. Thus, we can show that, indeed, like in seismology, certain levels are more impacted by a change than others.
- **RQ2:** What is the level most impacted by a change?
  We perform a quantitative study to confirm our observations of change propagation, using statistical tests to investigate which level may be the most impacted by a change, and classifying the levels having similar impact. Thus, we can deduce all classes with a higher risk to be impacted by any change to epicenter class.
- **RQ3:** What is the most reachable level by a change?
  As in **RQ2**, we perform a quantitative study to confirm our observations of change propagation, using statistical tests to investigate, for each level, the number of earthquakes that propagate until a given level. Thus, we can deduce the most reachable level.

B. Analysis Methods

To answer RQ1, RQ2, and RQ3, we apply our approach on Pooka, Rhino, and Xerces-J.

[4]http://www.suberic.net/pooka/
[5]http://www.mozilla.org/rhino/
[6]http://xerces.apache.org/

Epicenter Classes	$pr(c)$	$r(c)$	$h(c)$	$rh(c)$
Context	0.043243	2	135	67.5
IdScriptableObject	0.057838	1	9	9
Kit	0.038378	3	14	4.66
BaseFunction	0.027838	7	37	5.28

TABLE III
EPICENTER CLASSES IN RHINO

Epicenter Classes	$pr(c)$	$r(c)$	$h(c)$	$rh(c)$
TypeValidator	0.020261	3	25	0.12
XMLEventImpl	0.017062	6	21	0.28
DeferredDocumentTypeImpl	0.006441	25	68	0.36
XMLEntityScanner	0.002602	76	194	0.39

TABLE IV
EPICENTER CLASSES IN XERCES-J

*a) **RQ1**:* As in seismology, we are interested in the most important seismic sources. Thus, we use metrics combination $rh(c)$ to rank classes according to their importance. We observed the two most important epicenter classes in Rhino, and Xerces-J. To illustrate our observations, we selected four representative epicenter classs, in Rhino, and Xerces-J, as shown in Table III and Table IV. For each epicenter class, we report information from the programs bug trackers and mailing lists confirming the propagation of a change to other classes in different levels.

Using the R statistical system[7], we build the 3D graph visualising the change propagation from the epicenter class to other classes, through their levels. The axes of the graphic are the time, levels, and numbers of changes. Thus, we study the graph of a representative epicenter class in each program to assess whether, as in seismology, change impact is most severe near the epicenter and decrease far away from the epicenter.

*b) **RQ2**:* We perform an exhaustive study to confirm our observations, using a statistical test, for all classes in Pooka, Rhino, and Xerces-J. We consider each class as an epicenter class. We compute, for each level, the number of classes that changed after any change to the considered epicenter class and within the chosen time window.

For each level, we create a subset that contains the number of changes per class. We then apply ANOVA on these subsets to determine whether there are significant difference between subset means. When differences between subsets exist, the null hypothesis "H_0: the number of changes is similar for each level" is rejected. Then, we conduct Duncan's multiple range test to classify the subsets with respect to the differences between them. We choose Duncan's multiple range test because it can maintain a low overall type I error [42] and it uses a studentized range statistics within a multiple range test. We interpret the range-value as follow:

- Range 1, if subsets mean value is the minimum.
- Range 2, if subsets mean is adjacent to Range 1 mean.
- Range 3, if subsets mean is adjacent to Range 2 mean.

*c) **RQ3**:* This research question aims to verify whether most earthquakes would propagate through all class levels or just the first level. For each level, we create a subset that contains the number of earthquakes that stop at this level.

[7]http://www.r-project.org

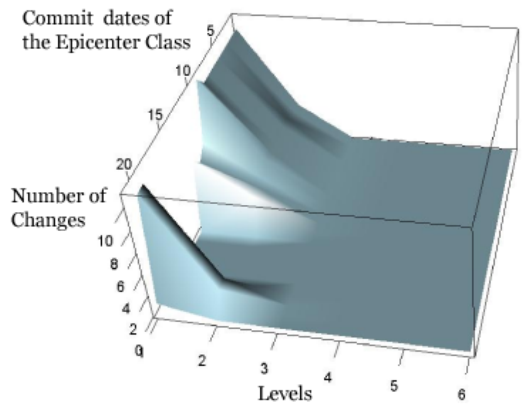


Fig. 3. Change propagation from XMLEventImpl

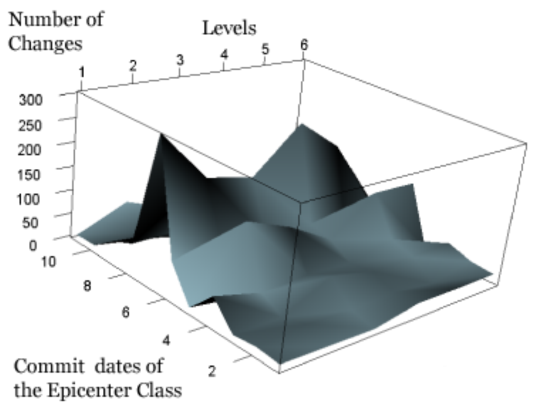


Fig. 4. Change propagation from TypeValidator

We then apply again ANOVA on these subsets to determine whether there are significant differences between their means. When differences between subsets exist, the null hypothesis "H_0: the number of earthquakes is similar for each level" is rejected. If the ANOVA results yield significant differences, we again apply Duncan's multiple range test to classify the subsets with respect to their differences. For example, in Xerces-J, the most reachable level by the epicenter class `XMLEventImpl` is level 3, while the epicenter class `TypeValidator` reaches at maximum level 6.

VI. Empirical Study Results

We now present the results of our empirical study.

A. *RQ1: Does our metaphor allow us to observe the scope of change impact?*

We answer positively to this question using the epicenter classes selected in Rhino, and Xerces-J.

In one hand, we build the 3D graphs visualising the change propagation from the selected epicenter classes in Pooka, Rhino, Xerces-J. Due to the lack of space in this paper, we present just two graphs in Xerces-J. Figure 4 and Figure 3 illustrate the change propagation phenomena, in Xerces-J, for the epicenter classes `XMLEventImpl` and `TypeValidator`. In `XMLEventImpl`, number of changes is very important for the first level, after that it decreases through the 2*nd* and 3*rd* levels. In `TypeValidator`, we observe that changes propagate until the 4*th* level and then it decreases significantly. However, in some cases, they may propagate to 6*th* level. Thus, we conclude that change propagation is different

In the other hand, we found external information in the program bug trackers and mailing lists. Thus, we report four examples (in Rhino and Xerces-J) of external information illustrating the propagation of a change from the chosen epicenter class to another class (in different level).

1) In Rhino:

- Epicenter class `IdScriptableObject`: we found the bug ID256321[8] that confirms that a change to the epicenter class propagated to `ScriptableObject` (level 1) to make Rhino objects serialisable.
- Epicenter class `BaseFunction`: we found the bug ID236117[9] that reports that a change to the epicenter class propagated to classes `Context` and `ScriptRuntime` (level 1), and `ContextFactory` and `WrapFactory` (level2).
- Epicenter class Context: we found the bug ID255891[10] that relates the epicenter class to three classes of level 1 (`CompilerEnvirons`, `ContextFactory`, and `ScriptRuntime`).
- Epicenter class Kit: we found the bug ID200551[11] that reports that a change to the epicenter class propagates to the class `DefiningClassLoader` (level 2).

2) In Xerces-J:

- Epicenter class `TypeValidator`: we found a message[12] in the mailing list stating that changes to `TypeValidator` propagate to `PrecisionDecimalDV`.
- Epicenter class `XMLEventImpl`: we found a SVN log commented in mailing list[13][14][15] that confirms the change propagation from the epicenter class to the classes: `EndDocumentImpl`, `EntityReferenceImpl` (level 1), and `StartElementImpl` (level 2).
- Epicenter class `DeferredDocumentTypeImpl`: we found the on-line discussion[16] showing that changes to the epicenter class must propagate to four classes in level 2: `DeferredElementImpl`, `DeferredEntityImpl`, `DeferredNotationImpl`, and `DeferredTextImpl`.

[8]https://bugzilla.mozilla.org/show_bug.cgi?id=256321
[9]https://bugzilla.mozilla.org/show_bug.cgi?id=236117
[10]https://bugzilla.mozilla.org/show_bug.cgi?id=255891
[11]https://bugzilla.mozilla.org/show_bug.cgi?id=200551
[12]http://comments.gmane.org/gmane.text.xml.Xerces.devel/5855
[13]http://xerces.markmail.org/message/eskpra4vcmaugtx6?q=XMLEventImpl+StartElementImpl
[14]http://xerces.markmail.org/message/33uxkvhfomocrngj?q=XMLEventImpl+StartElementImpl
[15]http://xerces.markmail.org/message/3hkhyuwj6v5oa7hw?q=XMLEventImpl+StartElementImpl+&page=2
[16]http://xerces.markmail.org/search/?q=DeferredDocumentTypeImpl#query:DeferredDocumentTypeImpl+page:2+mid:iyb37kwel5rdmaod+state:results

	Homogenous subsets for alpha = 0.1		
Levels	Range 1	Range 2	Range 3
5	107.5410		
4	147.7778		
3	150.0000		
2		202.0408	
1			354.4828

TABLE V
DUNCAN'S TEST APPLIED ON "NUMBER OF CHANGE IMPACTS" IN RHINO

	Homogenous subsets for alpha = 0.1		
Levels	Range 1	Range 2	Range 3
6	6.4015		
5	10.8485		
4	24.8333		
3		50.2789	
2		83.7273	
1			895.2652

TABLE VI
DUNCAN'S TEST APPLIED ON "NUMBER OF CHANGE IMPACTS" IN XERCES-J

	Homogenous subsets for alpha = 0.1		
Max Level	Range 1	Range 2	Range 3
5	.5833		
4	1.3712		
3	1.7500		
2		4.6136	
1			11.7121

TABLE VII
DUNCAN'S TEST APPLIED ON "NUMBER OF EARTHQUAKES" IN RHINO

	Homogenous subsets for alpha = 0.1		
Max Level	Range 1	Range 2	Range 3
6	10.5333		
5	16.3333		
4	21.6667		
3		30.0033	
2		43.2000	
1			54.8667

TABLE VIII
DUNCAN'S TEST APPLIED ON "NUMBER OF EARTHQUAKES" IN XERCES-J

- Epicenter class `XMLEntityScanner`: we found the bug ID1099[17] that relate the changes to the epicenter class with changes to `XMLParser` (level 3).

B. ***RQ2:*** *What is the level most impacted by a change?*

We apply our approach on Pooka, Rhino, and Xerces-J. Then, we create subsets that contains numbers of changes per class that propagates from each class, then we apply ANOVA on these subsets. The ANOVA results yield significant difference between subset means. We therefore conduct Duncan's multiple range to classify these subsets in each three programs. TableV summarises the results of Duncan's test applied on Rhino. We observe that change propagation, in some cases, reach level 5. This table shows that all the sample means are significantly different for levels 1 and 2 (because they are classified in different ranges), except the means of levels 3 and 4, for which there is no evidence of a difference, and thus they are grouped together in the same range. The non significant difference between levels 3 and 4 suggests that the number of changes are similar. The number of changes is much higher in level 1 (corresponding to the mean value 354.4828, and range 3) and this high difference (with respect to other means) results in a separate range. The same goes for the second level (corresponding to the mean value 202.0408, and range 2).

TableVI summarises the results of Duncan's test applied on Xerces-J. We observed that change propagation, in some cases, reach level 6. This table shows that level 1 differs significantly from the others, by being the most impacted. The levels 4, 5 and 6 are classified in range 1, thus the number of changes is almost similar at these levels. But, they are less impacted than levels 2 and 3 that are classified in range 2 (corresponding to the means values 50.2789 and 83.7273). The number of changes is much higher in level 1 (corresponding to the mean value 895.2652).

We can observe the same trend for Pooka. Due to the lack of space in this paper we provide our result in this link[18].

From the results of the three programs, we can conclude that: a) level 1 is the most impacted, b) level 2 is the second most impacted, but significantly less than level 1, c) levels 3, 4, 5, ... are significantly less impacted than level 1 and 2, and d) levels 3, 4, 5, ... are classified in the same range, because the number of changes seems similar in these levels.

C. ***RQ3:*** *What is the most reachable level by a change?*

In this research question, we apply the same approach as for RQ2. Here, for each level, the subset is the number of earthquakes that stop at this level. The ANOVA results yield significant differences. Thus, we apply Duncan's multiple range test.

Table VII summarises the results of Duncan's test applied on Rhino. We observe that the number of earthquakes that reach at maximum level 1 is greater (corresponding to the highest mean 11.7121) than those that reach the other levels. But, some changes propagate through levels 3, 4, and 5. The means of the number of earthquakes that reach levels 3, 4, 5 are too similar (corresponding means are: .5833, 1.3712, and 1.7500), consequently, they are grouped in the same range 1. The earthquakes that reach level 2 are less than those that reach level 1, but significantly greater than the others (levels 3, 4 and 5). As a result, earthquakes that reach levels 1 and 2 cannot be classified with earthquakes that reach other levels.

TableVIII summarises the results of Duncan's test applied on Xerces-J. We observed that the number of earthquakes that reach at maximum level 1 are the most frequent (corresponding to the mean 54.8667). But, the number of earthquakes that reach at maximum the levels 4, 5, and 6, are almost similar, and the least (corresponding means are 21.6667, 16.3333,and 10.5333). The number of earthquakes that reach at maximum levels 2 and 3, are significantly similar, and thus are regrouped in the same range.

[17]https://issues.apache.org/jira/browse/XERCESJ-1099

[18]http://www.ptidej.net/Members/hassaisa/docs/ICSM11-pooka.pdf

Thus, we conclude that almost change propagation stops at the level 1 and 2. But, there are some earthquakes that propagate until 3, 4, 5 or 6 level.

VII. Discussions

We now discuss our approach and its empirical study. With our approach, we analysed change propagation in three different programs belonging to different domains and with different sizes, and histories. We observed that changes did not propagate through different class levels with the same proportion in each program. We observe that the numbers of earthquake propagations that stops at the level 1 and 2 is the greatest. However, the number of earthquakes that stop at higher levels (3, 4, and 5) are the least. By determining what levels might have been affected by certain changes, we can help maintainers to rapidly pinpoint the source of a bug. Consequently, maintainer needs to only examine the indicated levels in priority instead of inspecting all the source code: our method is time saving.

We observed that some classes changes frequently and are changed periodically by developers. This observation could help developers be aware of classes that they should consider changing even if their changes is not directly linked to these classes (see scenario in Section I).

A. Threats to Validity

Several threats potentially impact the validity of our empirical study.

a) Construct validity: Construct validity concerns the relation between theory and observations. In this study, they could be due to the chosen time windows which may affect our observations. A too long time window would be misleading while a very narrow time window would not allow to observe interesting facts. Conservatively, we chose a class-dependent time window: the median of time between two subsequent changes, because the median is robust to extreme outliers and thus minimises spurious changes induced by too large time windows on classes connected to epicenter classes. However, we may not have used the most revealing time windows. More investigation is needed to better understand the role of time windows. Finally, it is possible, despite the confirmation using external sources of information, that some classes reported as impacted by a change did actually change for reason independent of the changes to the epicenter class classes.

b) Internal Validity: The internal validity of a study is the extent to which a treatment impacts the dependent variable. The internal validity of our study is not threatened because we have not manipulated the independent variable, extent of the change propagation.

c) External Validity: The external validity of a study relates to the extent to which we can generalise its results. The main threat to the external validity of our study that could affect the generalisation of the presented results relates to the analysed programs. We performed our study on four different Java programs belonging to different domains and with different sizes. However, we cannot assert that our results can be generalised to other larger programs and programs in other programming languages. Future work includes replicating this study on other programs to confirm our results.

d) Conclusion validity: Conclusion validity threats deals with the relation between the treatment and the outcome. We paid attention not to violate assumptions of the performed statistical tests. We applied ANOVA and Duncan's multiple range tests. ANOVA assumes that the data are normally distributed. We may have a problem of assuring this assumption. Thus, we improved our conclusion validity by increasing the risk of making a Type I error (increase the chance that we will find a relationship when in fact there is not), we can do that statistically by raising the alpha level. For instance, instead of using a 0.05 significance level, we use 0.10 as our cutoff point.

VIII. Conclusion and Future Work

Change propagation analysis in object-oriented programs is important to estimate the effort required for future maintenance tasks. The observed phenomena can help for making decisions concerning the process of future software projects, and reducing the overall cost of source code inspection [13].

Existing approaches for change impact analysis are based on the software structure and use static, dynamic, textual, and–or historical analyses [4]–[7], [9], [10], [12]. However, to the best of our knowledge, none of these approaches have been used to study the scope of change propagation.

In this paper, we proposed an approach to analyse change propagation and to study how far a change propagation will proceed from a given class to the others. Our approach considers changes to a class as an earthquake that propagates through the class levels, defined by the length of relationships chain that relate the epicenter class to the other classes.

We performed a qualitative and two quantitative studies on three open sources: Pooka, Rhino and Xerces-J, and thus, we answered the following research questions: **RQ1:** Does our metaphor allow us to observe the scope of change impact? **RQ2:** What is the level most impacted by a change? **RQ3:** What is the most reachable level by a change?

We showed that our intuition, about the impacted classes by a change must be near to the changed class, is incorrect in some cases. Therefore, Duncan's multiple range test confirms that level 1 has the highest number of changes. However, there are some change propagations that reach the 5th level in Rhino (and 6th in Xerces-J). Identifying the scope of change propagation could help, both developers and managers. Developers could locate easily the change impact, and thus they do not have to analyse the whole source code to understand the ripple effect of a change. They could include in their change set the classes belonging to the identified levels. Managers could estimate the efforts required to perform maintenance changes more accurately.

Future work includes applying our metaphor and our approach to other programs to confirm our observations. We will also adapt seismology models to predict changes to classes. In the case of earthquakes, seismologists are interested in debris forecasting, to predict the quantity of damage and seek to minimise the earthquake impact through the improvement of construction standards. Earthquake prediction technique generally use probabilistic methods to predict earthquake risk using past history. We will study the possibility of using the data about previous changes to forecast "earthquakes" that could occurs in a program, their potential damages, and the factors influencing their propagations.

ACKNOWLEDGMENT

This research was partially supported by FQRNT, NSERC, and Research Chairs in Software Patterns and Patterns of Software and in Software Evolution.

REFERENCES

[1] T. M. Pigoski, *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. Wiley, 1996.

[2] D. Bell, *Software Engineering, A Programming Approach*. Addison-Wesley, 2000.

[3] D. Hamlet and J. Maybee, *The Engineering of Software*. Addison-Wesley, 2001.

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

[5] R. S. Arnold and S. A. Bohner, "Impact analysis - towards a framework for comparison," in *the Conference on Software Maintenance*. IEEE Computer Society, 1993, pp. 292–301.

[6] J. Law and G. Rothermel, "Whole program path-based dynamic impact analysis," in *the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 308–318.

[7] J. R. Larus, "Whole program paths," *SIGPLAN Not.*, vol. 34, no. 5, pp. 259–269, 1999.

[8] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[9] A. T. T. Ying, J. L. Wright, and S. Abrams, "Source code that talks: an exploration of eclipse task comments and their implication to repository mining," in *the 2005 international workshop on Mining software repositories*. ACM, 2005, pp. 1–5.

[10] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *the 26th International Conference on Software Engineering*. IEEE Computer Society, 2004, pp. 563–572.

[11] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings of the International Conference on Software Maintenance*, ser. ICSM '98. IEEE Computer Society, 1998, pp. 190–.

[12] S. Bouktif, Y.-G. Gueheneuc, and G. Antoniol, "Extracting change-patterns from cvs repositories," in *the 13th Working Conference on Reverse Engineering*, 2006, pp. 221–230.

[13] S. L. Pfleeger, *Software Engineering: Theory and Practice*. Prentice-Hall, 1998.

[14] M. Weiser, "Programmers use slices when debugging," *Commun. ACM*, vol. 25, pp. 446–452, 1982.

[15] D. Binkley and M. Harman, "A large-scale empirical study of forward and backward static slice size and context sensitivity," in *the International Conference on Software Maintenance*. IEEE Computer Society, 2003, pp. 44–54.

[16] H. Agrawal and J. R. Horgan, "Dynamic program slicing," in *Proceedings of the ACM SIGPLAN 1990 conference on Programming language design and implementation*. ACM, 1990, pp. 246–256.

[17] X. Zhang, N. Gupta, and R. Gupta, "A study of effectiveness of dynamic slicing in locating real faults," *Empirical Software Engineering*, vol. 12, pp. 143–160, 2007.

[18] R. Santelices and M. J. Harrold, "Probabilistic slicing for predictive impact analysis," Georgia Institute of Technology. Center for Experimental Research in Computer Systems, Tech. Rep. GIT-CERCS-10-10, 2011.

[19] A. Zaidman, T. Calders, S. Demeyer, and J. Paredaens, "Applying webmining techniques to execution traces to support the program comprehension process," in *In Proceedings of the Conference on Software Maintenance and Reengineering*, ser. CSMR '05. IEEE Computer Society, 2005, pp. 134–142.

[20] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of ACM*, vol. 46, pp. 604–632, 1999.

[21] T. Girba, S. Ducasse, and M. Lanza, "Yesterday's weather: guiding early reverse engineering efforts by summarizing the evolution of changes," in *In Proceedings of the 20th IEEE International Conference on Software Maintenance*, ser. ICSM '04, 2004, pp. 40–49.

[22] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," in *Proceedings of the 11th IEEE International Software Metrics Symposium*. IEEE Computer Society, 2005, pp. 29–.

[23] D. M. German, A. E. Hassan, and G. Robles, "Change impact graphs: Determining the impact of prior codechanges," *Information and Software Technology.*, vol. 51, pp. 1394–1408, 2009.

[24] Y. Zhou, M. Würsch, E. Giger, H. C. Gall, and J. Lü, "A bayesian network based approach for change coupling prediction," in *Proceedings of the 15th Working Conference on Reverse Engineering*, ser. WCRE '08. IEEE Computer Society, 2008, pp. 27–36.

[25] S. Mirarab, A. Hassouna, and L. Tahvildari, "Using bayesian belief networks to predict change propagation in software systems," in *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, 2007, pp. 177–188.

[26] G. Antoniol, V. F. Rollo, and G. Venturi, "Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories," in *the 2005 international workshop on Mining software repositories*. ACM, 2005, pp. 1–5.

[27] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta, "An eclectic approach for change impact analysis," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. ACM, 2010, pp. 163–166.

[28] C. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424–38, 1969.

[29] H. Malik and A. E. Hassan, "Supporting software evolution using adaptive change propagation heuristics." in *ICSM'08*, 2008, pp. 177–186.

[30] A. E. Hassan and R. C. Holt, "Replaying development history to assess the effectiveness of change propagation tools," *Empirical Software Engineering*, vol. 11, pp. 335–367, 2006.

[31] K. E. Bullen, *An introduction to the theory of seismology*, 3rd ed. Cambridge University Press, Cambridge, 1963.

[32] Y. Y. KAGAN and L. KNOPOFF, "Statistical short-term earthquake prediction," *Science*, vol. 236, no. 4808, pp. 1563–1567, 1987.

[33] A. Morales-Esteban, F. Martínez-Álvarez, A. Troncoso, J. L. Justo, and C. Rubio-Escudero, "Pattern recognition to forecast seismic time series," *Expert System Application*, vol. 37, pp. 8333–8342, 2010.

[34] A. Saichev and D. Sornette, "Theory of earthquake recurrence times," *J.GEOPHYS.RES.*, vol. 112, p. B04313, 2007.

[35] S. C. Myers and W. R. Walter, "Using epicenter location to differentiate events from natural background seismicity," Lawrence Livermore National Laboratory, Technical Report, it was prepared for submittal to the 21 st Seismic Research Symposium: Technologies for Monitoring the Comprehensive Nuclear-Test-Ban Treaty UCRL-JC-134301; GC0402000, 1999.

[36] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, 1999.

[37] S. Kpodjedo, F. Ricca, P. Galinier, and G. Antoniol, "Recovering the evolution stable part using an ecgm algorithm: Is there a tunnel in mozilla?" in *the 2009 European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 179–188.

[38] O. Kaczor, Y.-G. Guéhéneuc, and S. Hamel, "Efficient identification of design patterns with bit-vector algorithm," *csmr*, vol. 0, pp. 175–184, 2006.

[39] Y.-G. Guéhéneuc and G. Antoniol, "DeMIMA: A multi-layered framework for design pattern identification," *Transactions on Software Engineering (TSE)*, vol. 34, no. 5, pp. 667–684, 2008, 18 pages.

[40] A. Bergeron and S. Hamel, "Vector algorithms for approximate string matching," *International Journal of Foundations of Computer Science*, vol. 13, no. 1, pp. 53–65, 2002.

[41] V. R. Basili and D. M. Weiss, "A methodology for collecting valid software engineering data," *IEEE Trans. Software Eng.*, vol. 10, no. 6, pp. 728–738, 1984.

[42] V. Bewick, L. Cheek, and J. Ball, "Statistics review 9: one-way analysis of variance." *Critical care*, vol. 8, no. 2, pp. 130–136, 2004.