A Case Study of a Seismology-based Approach to Change Impact Analysis

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*—The maintenance of large programs is a costly activity because their evolution often leads to two problems: an increase in their complexity and an erosion of their design. Impact analysis is crucial to make decisions among different alternatives to implement a change and to assess and plan maintenance activities by highlighting artefacts that should change when another artefact changes. Several approaches were proposed to identify software artefacts being affected by a change. However, to the best of our knowledge, none of these approaches have been used to study two pieces of information: (1) the scope of a change in a program and (2) the propagation of the change in time. Yet, these pieces of information are useful for developers to better understand and, thus, plan changes. In this paper, we present a metaphor inspired by seismology and propose a mapping between the concepts of seismology and software evolution. Our metaphor relate the problems of (1) change impact and earthquake's debris and (2) change propagation and damaged site predictions to observe the scopes and the evolution in time of changes. We show the applicability and usefulness of our metaphor using Rhino and Xerces-J.**

***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 software programs, developers face problems with maintenance [1]. In particular, understanding the evolution of a program is a time- and resource-consuming activity, despite its benefits to perform adequate changes [2], [3]. Approaches exist to study software evolution phenomena, including evolution trends, traceability between artefacts, and the impacts of changes.

In particular, understanding the propagation and the impact of changes is crucial for estimating the effort required to implement a change and for planning maintenance activities. Change impact analysis aims at identifying software artefacts being affected by a change; it provides the potential consequences of a change and an estimate of the artefacts that must be modified to accomplish a change [4]. As in previous work, *e.g.*, [5]–[7], for the sake of simplicity and without loss of generality, we focus on C++ and Java classes.

Existing approaches for change impact analysis are based on the software structure and use static, dynamic, and–or textual analyses [4], [8]–[10]. However, change impact analyses should also take into account classes that are semantically coupled but that may not be structurally depend on one another. Thus, recently, researchers proposed to analyse the content of version-control systems to identify semantically coupled classes (*i.e.*, logical couplings) by assessing whether they co-change together within change sets [11]–[13].

However, to the best of our knowledge, none of these approaches have been used to study two pieces of information: (1) the scope of changes in a program and (2) the propagation of changes in time. First, they do not take into account a "distance" among classes, for example whether two co-changing classes are in direct relation or separated through a long chain of relationships. Yet, studying the scope of changes could help developers prioritise their changes according to the forecast scope of changes. Second, they do not report the trends of the co-changes among files in time, for example whether changing a class impacts more and more co-changing classes as the program evolves. Yet, studying the propagation of changes in time could help developers plan their maintenance activities according to the forecast time that a change may take to *settle in* a program.

We propose a novel approach to change impact analysis specifically designed to study the scope and the propagation of changes, based on a metaphor between seismology and change impact analysis. Our approach considers changes to a class as a seism that propagates through the program following the class levels, defined by direct relationships among classes, and that impacts other classes with different extent, depending on (1) their relationships with the "epicenter class" and other impacted classes and (2) the moments in time at which developers change impacted classes. With our approach, we can observe and, in future work, will predict the scopes and the propagations of changes through the class levels of a program and in time.

To show the applicability and the usefulness of our approach, we apply our approach on three epicenter classes in Rhino and six epicenter classes in Xerces-J and study its approach results to answer the three research questions:

- RQ1: Is it possible to find external information that support our observations on the impact of changes according to our metaphor?
- RQ2: Does our metaphor allow us to observe interesting change patterns in scope?
- RQ3: Does our metaphor allow us to observe interesting change patterns in time?

Our answers to these research questions confirm that: (1) like with earthquakes dramatic consequence, we can find external information in mailing lists and but reports confirming the observed change propagation and its negative impact on the impacted classes and (2) like earthquakes, changes propagate in programs with decreasing impact as the levels from the changed class increase and as time passes.

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 with future work.

II. Related work

Change impact analysis [4] aims at identifying classes being affected by a change. In this section, we review and discuss related work with respect to our approach.

A. Structure-based Change Impact Analyses

Several change impact analyses are based on class dependencies and use static, dynamic, and textual analyses.

Arnold and Bohner proposed several models of change propagation [4], [8]. 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 typically 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 its callers and callees. Therefore, it can be highly 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. Static slicing is typically based on data- and control-flow graphs that are computationally expensive to process and analyse.

Law *et al.* [9] argued that static slicing is much more precise than transitive closure on call graphs but it may return sets of classes that are impacted by a change that are too large. Dynamic slicing can improve the conservative behavior of static slicing. However, it is subject to the risk of lower precision and lower recall as it depends on the chosen scenarios and–or executed test cases. Consequently, Law *et al.* [9] introduced a new approach to method-level change impact analysis, based on whole-path profiling. In their approach, dynamic traces are compressed using the path profiling technique [10], then they apply their 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.

Hassan *et al.* [14] proposed a model of change propagation, based on several heuristics, for predicting the set of classes that should change after a particular class has been changed. In their approach, they combined various sources of data for change impact analysis, such as static dependencies between classes and historical co-change relations.

B. History-based Change Impact Analyses

Historical analyses add complementary information to static and dynamic analyses, *i.e.*, information about changes to classes over time, such as the co-changing artefacts.

Ying *et al.* [12] and Zimmermann *et al.* [11] proposed to mine version-control systems. They applied association rules that identify logical couplings [15] between classes. A change occurring to a 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.* [7] 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, including association rules [11], [12].

Canfora *et al.* [16] 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.

C. Probabilistic Approaches

Zhou *et al.* [17] and Mirarab *et al.* [18] 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 a program, such as change comments and author information. Zhou *et al.* [17] used the Evolizer system that retrieves all modification reports from a CVS and uses a sliding window algorithm to group them. Their change propagation model is able to predict future change couplings. Mirarab *et al.* [18] 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.* [19] 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.* [20] proposed the use of a generalisation of univariate autoregression model, to capture the evolution and the inter-dependencies between multiple time series. They applied the bivariate Granger causality test [21] to infer

the mutual dependencies between classes by measuring the statistical confidence that the time series representing the histories of changes of a class *A* can be used 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. Discussion

Our approach is complementary to this previous work: we propose a metaphor to specifically study (1) the scope of changes in a program and (2) the propagation of changes in time. We get inspiration from this previous work and from seismology to propose a novel approach for the analysis of the impact of changes dedicated to observing the scope and propagation of changes.

III. The Earthquake Metaphor

In this section, we relate earthquakes and software changes by presenting a mapping between concepts in seismology and in change impact analysis. We use this mapping to observe the scope and propagation of changes as a global phenomena both physically and temporally.

A. Seismology

Seismologists study earthquakes and seismic waves that move through and around the Earth. 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 an epicenter, more the debris are important.

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.
2) **Debris forecasting**: Predicting and estimating debris, or structural damage, after an earthquake, to assist debris managers in planning large scale debris removals.
3) **Earthquake behaviour analysis**: Studying earthquake clustering and quasi-periodicity. Clustering is observed as short- or medium-term interactions among earthquakes (fore shock, main shock, after shock). Quasi-periodicity is a long-term behaviour of earthquakes caused by a same seismic source.

Figure 1 illustrates how the magnitude of an earthquake varies in space and time. If we observe the variation of magnitude around one epicenter for a specific time, we can observe that the magnitude is maximum in the epicenter and decreases with the distance from the epicenter to the considered site.

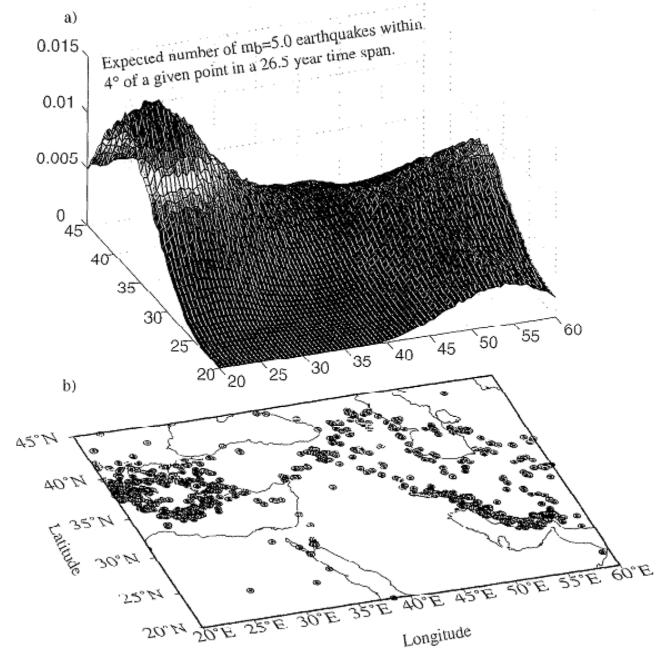


Figure 1. Epicenters distribution in space and time. The map shows the expected number of earthquakes of a given magnitude occurring within a given radius from each point. (From [22].)

B. Change Impact Analysis

Change propagation begins with a class being changed. This change propagates and forces other classes to change. These changed classes, in turn, may yield to other changes. Thus, change propagation is due to changes “moving” from one class to another through a 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 these changes in these methods.

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 the effort to implement some change request (typically, the more impacted classes by a change, the greater the effort) and in performing the most adequate changes, limiting the risk of introducing bugs by clearly identifying classes that could be impacted. It could also allow the prediction of future changes and of their propagations. However, to the best of our knowledge, no previous work studied the scope of changes in a program and their propagation in time.

C. Change Impact Analysis and Seismology

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

In seismology, the epicenter of an earthquake is located at most active seismic areas. To determine the epicenter (loca-

Concepts	
Change Impact Analysis	**Seismology**
"Important" classes	Active seismic areas
Software change	Earthquake
"First" changed class	Epicenter
Change propagation	Seismic wave propagation
"Other" changed classes	Damaged sites
Class level	Distance from an epicenter to a damaged site
Relationships between classes	Types of soil
Types of changes	Types of debris
Quantity of changes	Structural damage (debris)
Scope of changes	Extent of the damaged sites
Propagation of changes in time	Earthquake behaviour

Table I
MAPPINGS BETWEEN CONCEPTS OF SEISMOLOGY AND SOFTWARE CHANGE IMPACT

tion) of an earthquake seismologists analyse seismograms, which record the seismic activities for a given location.

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 measure. In the following, we use four different measures to identify important classes although our approach applies on any classes in a program.

In seismology, several factors determine the structural damages resulting of an earthquake, including the proximity to the active area. Shaking is most severe near the epicenter and drops off away from the epicenter, *i.e.*, more the infrastructures are near to the epicenter more the debris is important, and in time, *i.e.*, as time passes, the earthquake energy decreases to zero.

In change impact analysis, changes propagate through a program via the class relationships and other logical couplings among classes. Yet, to the best of our knowledge, no previous work reported observations on the propagation of changes through the program and in time, which are the aim of our metaphor and the subject of our empirical study.

In seismology, seismic waves propagate from the epicenter of an earthquake until a damaged site, depending upon the types of soil through which they move. An earthquake potential damages are determined by the types of soil and by the construction of the buildings within the damaged sites.

In change impact analysis, changes propagate from the epicenter class to the impacted class, depending upon the types of relationships among these classes. The potential impact of a change on other classes may be determined by the design quality of the classes, typically reflected by the cohesion of and coupling among classes.

Using this mapping between change 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 in that it uses structural and historical analyses with the specific aim to study the scope of changes in programs and the propagation of changes in time.

IV. Approach

Our approach to change impact analysis based on our metaphor divides in four steps. Given an object-oriented program, first, we compute various metric values to rank classes according to their importance. 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 extract all the commits that happened after any change to the epicenter class and within the chosen time window. Fourth, we build the graph describing the propagation of changes from the epicenter class to all other related classes, both through the class levels and in time. With this graph, we can then observe and discuss (1) the scopes of the changes and (2) the propagation of changes in time. We now give more details of each step.

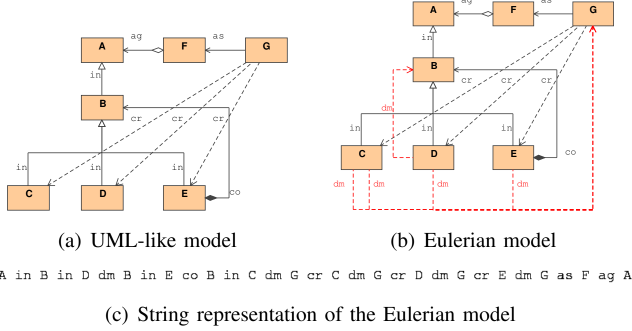
A. Step 1: Measuring Class Importance

Our approach applies to any class, chosen as epicenter class, which is different from earthquakes that are due to specific seismic sources. Yet, as with earthquakes, we are interested in the most important seismic sources, which could cause the most damage. Thus, in this paper, we identify the most "important" classes in a program using a history-based metric and a PageRank-based metric and a combination thereof.

1) History-based Metric: We define the history-based metric as the number of all commits related to a given class in the entire history of the program, extract 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.

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, which we then query 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 [23] is the 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 relationships from other (preferably important) classes. If a class is the only reference of a very important class, it might



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

(c) String representation of the Eulerian model

Figure 2. Representations of a simple example program (from [26]).

be ranked higher than another class in relationship with low-ranked classes. We use the algorithm previously developed by Kpodjedo [24] to compute the PageRank value, $pr(c)$, and rank, $r(c)$, of any class c: the lower its rank, the more important the class.

3) Combination of the History- and PageRank-based Metrics: We combine 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 lower than that of the other class. We thus define $rh(c) = \frac{r(c)}{h(c)}$.

The combination of the two metrics rh provides an accurate assessment of the importance of a class 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 critical classes first, which should then be analysed to analyse their impact on the other classes of a program when they are changed by developers.

B. Step 2: Identifying Class Levels

Change propagation depends on the type and level of coupling among classes. We represent the "distance" between two classes as the difference of class levels between the two classes, with respect to the epicenter class. We use a bit-vector algorithm [25] that applies bounded number of operations to an input vector, regardless of the length of the input, to compute the class levels, given an epicenter class.

We fist convert a program into a string because bit-vector algorithms work with strings. Figure 2 shows an example of a string representation of a program design. We obtain such a string by applying the Simplex and the Chinese postman algorithm, as explained in our previous work [26].

Using the program string representation, we build the characteristic vectors of each token in the representation. The characteristic vector of a token t associated with the string $x = x_1 \ldots x_m$, is $t = (t_1 \ldots t_m)$:

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

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

$$\begin{aligned} A &= \mathbf{1}\underbrace{0000000000000000}_{18} \\ in &= 0\mathbf{1}0\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{0000000}_{7} \\ B &= 00\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{00000000}_{8} \end{aligned}$$

Using the characteristic vectors, we identify all classes that are directly connected to some class A by computing conjunctions between the characteristic vector of A, that of all the relationships (*e.g.*, *in*, *co*, *cr*, *as*, *ag*), and that of all the other classes (*e.g.*, B). For example, to identify all classes directly related to A through the inheritance relationship *in*, we compute:

$$\begin{aligned} (\rightarrow\rightarrow A) &= 00\mathbf{1}\underbrace{0000000000000000000}_{16} \\ (\rightarrow in) &= 00\mathbf{1}0\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{000000}_{6} \\ B &= 00\mathbf{1}000\mathbf{1}000\mathbf{1}\underbrace{00000000}_{8} \\ R &= (\rightarrow\rightarrow \mathbf{A}) \wedge (\rightarrow \mathbf{in}) \wedge B \\ &= 00\mathbf{1}\underbrace{00000000000000000000}_{16} \end{aligned}$$

and assess whether the bit vector R is null (contains only zeroes). If R is not null, then class B is in level 1 with respect to A, else it belongs to another level. We repeat this process with all classes, all relationships, and all possible shifts to identify the set classes at level 2, *i.e.*, the set of classes that have a relationship with one of those of level 1, and so on.

C. Step 3: Identifying Impacted Classes

We mine the version-control system of a program to extract data related to the history of the epicenter class. We collect, for each change impacting the epicenter class and within a time window after the change for T milliseconds: (1) the names of all classes that are involved in any change and (2) the number of changes of each class changed after the epicenter class changed. We use for time window a value of T that we compute as the median of the durations between two changes to the epicenter class.

We again use Ibdoos framework to write queries against its database to collect at each point in time where a given epicenter class changes and during T milliseconds, the names of all subsequently changed classes and the number of changes that these classes underwent.

For example, given an epicenter class A, as shown in Figure 2, and assuming that, within T milliseconds of a change $C1$ to A, both classes B and F changes. Then, we

Programs	Classes	Representations (in tokens)	Releases	Dates
Rhino v1.6.R5	163 449	40,803 266,265	11	10/05/1999 19/11/2006
Xerces-J v2.11.0	5,100 12,585	162,583 1,195,353	39	05/11/1999 01/01/2010

Table II
STATISTICS FOR THE PROGRAMS FROM THEIR FIRST TO THEIR LAST STUDIED RELEASES.

will associate the sets $\{B,F\}$ and $\{nc_T(B), nc_T(F)\}$ to $C1$, where *nc* returns the number of changes to a class within a time frame T.

D. Step 4: Visualising Change Propagation

In this last step, we build the the 3D graph visualising the propagation of changes to a given epicenter class to all other classes, through the levels and time. The axes of the graph are time, levels, and number of changes.

For example, Figure 3 shows the graph obtained using class TypeValidator as epicenter. Given a date, it shows the number of changes to the epicenter class and the propagation of these changes to classes in other levels around the epicenter class. We compute the number of changes at each level as the sum of the number of changes to each class belonging to that level. Thus, in Figure 3, we observe that:

- At date 0, TypeValidator was changed a bit and classes in other levels did not change dramatically.
- At date 8, TypeValidator was changed dramatically and its change propagated to classes at levels 2 to 5.

We further discuss this figure and others in our following empirical study to answer our three research questions.

V. EMPIRICAL STUDY DESIGN

Following the Goal Question Metric (GQM) methodology [27], the *goal* of our study is to show the applicability and usefulness of our metaphor and mapping, with the *purpose* of gathering interesting observations on change propagation and confirming these observations with external information. The *quality focus* is that observing the propagation of changes helps developers assess their change efforts and perform more adequate changes. The *perspective* is of both researchers and practitioners who should be aware of the scope and the potential consequences in time 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* of our study is both the comprehension and the maintenance of programs.

A. Objects

The *objects* of our study consist of two open source programs Rhino and Xerces-J. Table II presents some descriptive statistics of these programs.

Epicenter Classes	$pr(c)$	$r(c)$
IdScriptableObject	0.057838	1
Context	0.043243	2
Kit	0.038378	3
BaseFunction	0.027838	7

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

Rhino[1] is an open-source implementation of JavaScript written entirely in Java. Xerces-J[2] is an open-source family of software packages for parsing and manipulating XML.

We choose these programs because they are medium-sized open-source programs, yet small enough to manually verify our observations on the propagation of changes and to manually confirm our observations using external information from various sources, such as bug reports.

B. Research Questions

We investigate if it is possible to apply our metaphor and mapping to observe the propagation of changes in scope and in time. We want to confirm that the observations of change propagation using our approach, based on our metaphor, provide useful insights to developers regarding the scope and time of impacted classes given a change. This study aims answering the following research questions:

- RQ1: Is it possible to find external information that support our observations on the impact of changes according to our metaphor?
- RQ2: Does our metaphor allow us to observe interesting change patterns in scope?
- RQ3: Does our metaphor allow us to observe interesting change patterns in time?

C. Analysis Methods

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

For Rhino, we only use the PageRank-based metric and we select the four top epicenter classes shown in Table III. For Xerces-J, we combine the history-based and PageRank-based metrics and we select also the top four epicenter classes, shown in Table IV. We use two different measures of the importance of classes to show that our approach does not depend on the chosen classes, yet to also provide interesting observations by selecting classes that deemed "important"

[1] http://www.mozilla.org/rhino/
[2] http://xerces.apache.org/

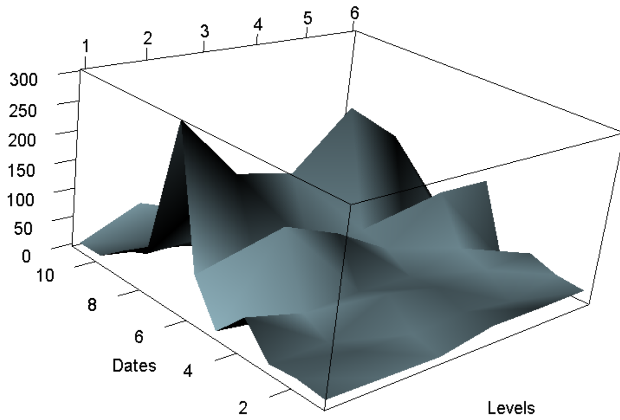


Figure 3. Change propagation from TypeValidator

given some measure. Future work includes performing an exhaustive study of change propagation for any classes and using other measures of a class importance.

To answer RQ1, we use the eight epicenter classs selected in Rhino and Xerces-J and, for each epicenter class, we looked for information in the program bug trackers and mailing lists for confirmation of our observation: that classes found by our approach as being impacted by changes to the epicenter classes indeed are related to the epicenter classes. Thus, we report for each epicenter class at least one example of external information confirming the propagation of a change to at least one other class in another level. To answer RQ2 and RQ3, we study the graphs of two representative epicenter classs of Xerces for lack of space. We analyse the graphs to assess whether, as seisms, change propagate in clusters or with quasi-periodicity. We thus show the applicability and usefulness of our metaphor and mapping using external sources of information, including mailing lists and bug reports.

VI. Empirical Study Results

we now present the results of our empirical study.

A. RQ1: Is it possible to find external information that support our observations on the impact of changes according to our metaphor?

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

In Rhino:

- Epicenter class IdScriptableObject and class ScriptableObject at level 1: we found the bug ID 256321[3] entitled "Regression: no serialization for IdScriptableObject" that confirms that a change to IdScriptableObject propagated to ScriptableObject to make Rhino objects serialisable.

[3]https://bugzilla.mozilla.org/show_bug.cgi?id=256321

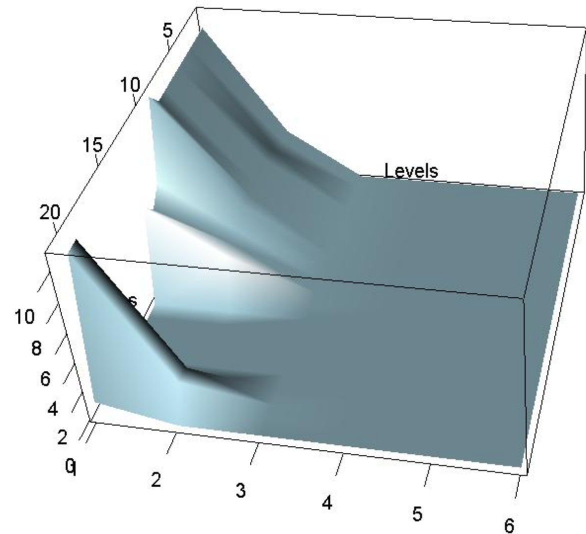


Figure 4. Change propagation from XMLEventImpl

- Epicenter class BaseFunction and classes Context and ScriptRuntime at level 1 and ContextFactory and WrapFactory at level 2: we found the bug ID 236117[4] entitled "Context sealing API for Rhino" that relate BaseFunction with the classes to seal contextes.
- Epicenter class Context and classes CompilerEnvirons, ContextFactory, and ScriptRuntime at level 1: we found the bug ID 255891[5] entitled "Compatibility issue: non JS values stored as properties" that relate these classes.
- Epicenter class Kit and classes NativeJavaPackage and DefiningClassLoader at level 2: we found the bug ID 200551[6] that confirms that the classes are impacted by a change to Kit.

In Xerces-J:

- Epicenter class TypeValidator and classes PrecisionDecimalDV at level 1: we found a message[7] in the mailing list stating that changes to TypeValidator impact PrecisionDecimalDV.
- Epicenter class XMLEventImpl and classes EntityReferenceImpl and EndDocumentImpl at level 1 and StartElementImpl at level 2: we found a SVN log commented in mailing list[8][9][10] that confirm the change propagation from the epicenter class to the

[4]https://bugzilla.mozilla.org/show_bug.cgi?id=236117
[5]https://bugzilla.mozilla.org/show_bug.cgi?id=255891
[6]https://bugzilla.mozilla.org/show_bug.cgi?id=200551
[7]http://comments.gmane.org/gmane.text.xml.xerces-j.devel/5855
[8]http://xerces.markmail.org/message/eskpra4vcmaugtx6?q=XMLEventImpl+StartElementImpl
[9]http://xerces.markmail.org/message/33uxkvhfomocrngj?q=XMLEventImpl+StartElementImpl
[10]http://xerces.markmail.org/message/3hkhyuwj6v5oa7hw?q=XMLEventImpl+StartElementImpl+&page=2

other classes.

- Epicenter class DeferredDocumentTypeImpl and classes DeferredElementImpl, DeferredEntityImpl, DeferredNotationImpl, and DeferredTextImpl at level 2: we found the on-line discussion[11] showing that changes to the epicenter class must propagate to the other classes.
- Epicenter class XMLEntityScanner and class XMLParser at level 3: we found the bug ID 1099 [12] with subject "XMLEntityScanner is not correctly processing QName" that relate the changes to XMLEntityScanner with changes to XMLParser.

B. RQ2: Does our metaphor allow us to observe interesting change patterns in scope?

The shape of the curve of the change propagation in Figure 1 is similr to that in the Figure 3. TypeValidator changes dramatically at the 8th period and we observe that its change impacts other classes up to level 6. This behaviour is similar to that of earthquakes with high magnitude and that impact site far away. Also, we could argue that the major changes at level 5 could be an earthquake on its own, triggered by the changes to TypeValidator as in an after shock. Such after shock are common with earthquake with high magnitude, such as with the 2004 Tsunami that hit Indonesia[13], which had an after shock of magnitude 8.7, leading seismologists to debate as to whether this earthquake was an after shock or a "triggered" earthquake.

For epicenter class XMLEventImpl, we observe a different behaviour: the impact of changes are limited to near classes even if the magnitude of the change is high. The maximal level reached by the change propagation is 2. The numbers of changes at level 1 are greater that that at the level 2. The change of the class XMLEventImpl do not lead to after shocks and do not trigger other earthquakes. The interpretation of this observation is that class XMLEventImpl could be well designed or is little coupled to other classes.

C. RQ3: Does our metaphor allow us to observe interesting change patterns in time?

We answer positively to the question about identifying the maximum distance of a change propagation.

Figure 3 illustrates the change propagation phenomena for the epicenter class TypeValidator, we noticed that the change impact propagate until the fourth level (at most) and then it decreases significantly. We find quasi-periodic behavior in the avalanche time series, *i.e.*, we notice the a quasi-periodic recurrence of large earthquakes (with high magnitude) that are separated the sequences of small earthquakes.

[11] http://xerces.markmail.org/search/?q=DeferredDocumentTypeImpl#query:DeferredDocumentTypeImpl+page:2+mid:iyb37kwel5rdmaod+state:results

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

[13] http://en.wikipedia.org/wiki/2004_Indian_Ocean_earthquake_and_tsunami

Figure 4 shows that the epicenter class XMLEventImpl change impact is very important for the first level, after that it decreases through the second and third levels. It shows also that the epicenter class XMLEventImpl.java is like an active earthquake zone that is always subject to frequent earthquakes with almost the same magnitude.

VII. Discussions

We now discuss our approach and its empirical study.

A. Empirical Study Observations

For each epicenter we tracked the classes that changes until another change of this class and noticed sometimes after shocks between the changes of the epicenter classes. Future work includes studying these after shocks and assessing their "back" impact on the epicenter classes as well as on other classes.

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.

B. Other Metrics to Identify Epicenter Classes

With our approach, we identified epicenter classes using two measures. However, the number of commit is not necessarily significant because it can be due to minor changes to the classes. So we suggest also other measures to identify epicenter classes.

In future work, we plan to use structural-based and relationship-based metrics. We define the structural-based metric as the percentage of public methods and public attributes that are added in, modified, or deleted from a class between one release and the next. This measure is based on the the observation that any change in the public interface of a class can trigger a large amount of changes in other classes, due to the coupling among classes. Let c_i be a class in release i and $I(c_i)$ be the set of public and protected, local and inherited, methods of c_i. We could compute the structural-based metric of c_i as:

$$s(c) = \frac{2 \times |I(c_{i+1}) - I(c_i)|}{I(c_i) + I(c_{i+1})}.$$

We could also use a relationship-based metric computed as the percentage of a relationships that were added to or deleted from a class between one release and the next. It refers to the extent to which part of a design is not preserved throughout the evolution of the program.

C. Threats to Validity

Several threats potentially impact the validity of our empirical study.

Construct validity: Construct validity concerns the relation between theory and observations. We took care to manually confirm the observations resulting from our approach, including using previously manually-validated data [28]. The time windows of observation 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. Further analyses are necessary to study the precision and recall of our approach.

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.

External Validity: The external validity of a study relates to the extent to which we can generalise its results. Our investigation is limited to two Java programs and four epicenter classes and, thus, we cannot claim any generalisation. Future work includes replicating our study to other programs, all their classes, and confirming our observations with other external sources of information.

Conclusion validity: Conclusion validity threats deals with the relation between the treatment and the outcome. Our study is essentially qualitative in its nature, it did not require any statistical test and, thus, its conclusion validity is not a concern.

VIII. Conclusion and Future Work

Existing approaches for change impact analysis are based on the software structure and use static, dynamic, textual, and–or historical analyses [4], [8]–[13]. However, to the best of our knowledge, none of these approaches have been used to study: (1) the scope of changes in a program and (2) the propagation of changes in time.

Consequently, we proposed a novel approach to change impact analysis specifically designed to study the scope and the propagation of changes, based on a metaphor between seismology and change impact analysis. Our approach considers changes to a class as a seism that propagates through the program following the class levels, defined by direct relationships among classes, and that impacts other classes with different extent, depending on (1) their relationships with the epicenter class and (2) the moments in time at which developers change impacted classes.

We showed the applicability and the usefulness of our approach by applying it to Rhino and Xerces-J and using its results to answer three research questions: RQ1: Is it possible to find external information that support our observations on the impact of changes according to our metaphor? RQ2: Does our metaphor allow us to observe interesting change patterns in scope? RQ3: Does our metaphor allow us to observe interesting change patterns in time?

Using qualitative evidence from external sources of information and analyses of the graphs built by our approach, we answered these research questions positively and confirmed that: (1) like with earthquakes dramatic consequence, we can find external information in mailing lists and bug reports confirming the observed change propagation and its negative impact on the impacted classes and (2) like earthquakes, changes propagate in programs with decreasing impact as the number of levels from the changed class increase and as time passes.

Future work includes applying our metaphor and our approach to other programs to confirm the adequation of the two phenomena: earthquake and change impact. 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. Seismologists also use the geological composition and structures of the Earth because they are important factors to predict damage. 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 (Research Chairs in Software Patterns and Patterns of Software and in Software Evolution), and Canada Research Chair Tier I in Software Change and Evolution.

References

[1] T. M. Pigoski, *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. New York: 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] G. Antoniol, V. F. Rollo, and G. Venturi, "Linear predictive coding and cepstrum coefficients for mining time variant information from software repositories," in *Proceedings of the 2005 international workshop on Mining software repositories*. New York, NY, USA: ACM, 2005, pp. 1–5.

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

[7] S. Bouktif, Y.-G. Gueheneuc, and G. Antoniol, "Extracting change-patterns from cvs repositories," in *Proceedings of the 13th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 221–230.

[8] R. S. Arnold and S. A. Bohner, "Impact analysis - towards a framework for comparison," in *Proceedings of the Conference on Software Maintenance*, ser. ICSM '93. IEEE Computer Society, 1993, pp. 292–301.

[9] J. Law and G. Rothermel, "Whole program path-based dynamic impact analysis," in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE '03. IEEE Computer Society, 2003, pp. 308–318.

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

[11] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. IEEE Computer Society, 2004, pp. 563–572.

[12] 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 *Proceedings of the 2005 international workshop on Mining software repositories*, ser. MSR '05. ACM, 2005, pp. 1–5.

[13] S. Bouktif, Y.-G. Gueheneuc, and G. Antoniol, "Extracting change-patterns from cvs repositories," in *Proceedings of the 13th Working Conference on Reverse Engineering*, ser. WCRE '06, 2006, pp. 221 –230.

[14] A. E. Hassan and R. C. Holt, "Predicting change propagation in software systems," in *Proceedings of the 20th IEEE International Conference on Software Maintenance*, ser. ICSM '04. IEEE Computer Society, 2004, pp. 284–293.

[15] 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–.

[16] 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–.

[17] 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 2008 15th Working Conference on Reverse Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 27–36.

[18] 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.

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

[20] 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*. New York, NY, USA: ACM, 2010, pp. 163–166.

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

[22] 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.

[23] 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.

[24] 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 *CSMR '09: Proceedings of the 2009 European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 179–188.

[25] 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.

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

[27] 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.

[28] M. Eaddy, T. Zimmermann, K. D. Sherwood, V. Garg, G. C. Murphy, N. Nagappan, and A. V. Aho, "Do crosscutting concerns cause defects?" *IEEE Trans. Softw. Eng.*, vol. 34, pp. 497–515, July 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1446226.1446244