Grouping Semantically Related Change-Sets to Enhance Identification of Logical Coupling

by

Neeraj Mathur, Sai Anirudh Karre, Y.Raghu Babu Reddy

in

The 31st International Conference on Software Engineering Knowledge Engineering

Hotel Tivoli Oriente, Lisbon, Portugal

Report No: IIIT/TR/2019/-1

Centre for Search and Information Extraction Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2019

Grouping Semantically Related Change-Sets to Enhance Identification of Logical Coupling

Neeraj Mathur, Sai Anirudh Karre, Y. Raghu Reddy
Software Engineering Research Center
IIIT Hyderabad, Telangana, India
neeraj.mathur, saianirudh.karri{@research.iiit.ac.in}, raghu.reddy@iiit.ac.in

***Abstract*—Identifying dependency between various artifacts in a large scale software system is a non-trivial task. As the software evolves, multiple artifacts like files, docs, classes, database scripts, etc., are likely to undergo change concurrently. Such artifacts tend to have a dependency between them, otherwise referred to as logical coupling. Researchers have used *Support* and *Confidence* as an association rule based measurement to predict the levels of logical coupling among the software artifacts. However, employing a single change on a software artifact can span across various closely related changes when many code contributors are working on the same change. Thus it is important to pre-process and group these semantically related change-sets before identifying logical coupling. In this paper, we propose a method to identify logical coupling and group semantically related change sets. We evaluate our method on real-world git repositories and document our observations.**

***Index Terms*—Cosine similarity, dependency, logical coupling, repository mining, software evaluation, software maintenance, reverse engineering**

I. Introduction

Large Software systems invariably comprise of artifacts written in different programming languages integrated by diverse technologies using a variety of implementation methods. Identifying the dependency between artifacts written in two different programming languages is a non-trivial task. For example, a JavaScript method may depend on a web service written in C#. Tracking such dependencies gets increasingly difficult over a period of time and as they tend to become the origins of defects in an overall software project.

Heuristically, identifying dependencies from code-revision history is considered to be lightweight than conducting a structural analysis of the entire artifact. In the case of code-revision history, a small amount of information is required to be analyzed in order to understand the dependency. Such information is typically stored via the log files of version control system like GIT, SubVersion, TFS etc. Almost in all cases, such dependencies are primarily documented in the form of a free-text comment. Thus, dependency analysis can be performed between two or more artifacts written in different languages without having trouble in parsing and analyzing the content of the artifacts. Logical coupling is one such implicit dependency observed between two or more software artifacts. It has been found that artifacts that are considered to be logically coupled artifacts when they change together frequently during the evolution of a system [9]. It can *reveal dependencies* that are *not structural* and therefore are not present in the code or in the documentation.

DOI reference number:10.18293/SEKE2019-166

The reliability of all the existing studies on logical dependencies is inherently connected to the accuracy of the approach used to identify such dependencies [14]. Version Control Systems (VCS), that are atomic-featured in their nature have a *change-set* - which is comprised of mutually checked-in files that result in a single commit. In general, software practitioners often rely on the existence of the atomic commit feature and consider the change-set as the actual set of files that were changed together by a code-developer while working on a given code-based task. In the case of multiple developers implementing the same change, such code-change can span across a series of consecutively connected and closely related individual change-sets. Therefore, by simply inspecting the change-sets in isolation may lead to incomplete or incorrect results with respect to the association of logical coupling.

In the past, researchers have proposed approaches to understand logical coupling. Grouping commits with the same authors using sliding time window concept was widely used [14]. However, semantic relationships between the artifacts - like the same work item or issue number, cosine similarity of two revision comments, etc. have not been explored by researchers. *Cosine similarity* is a popular measure in Information retrieval and Data Mining areas. It can be used to measure the similarity between two documents with respect to their textual content [11]. In this paper, we present our preliminary work by utilizing such techniques to identify the logical dependency between artifacts leading to an increase in change-set identification and accuracy. In this paper, we:

- suggest an approach to improve logical coupling detection using semantic relations drawn from the revision comments and cosine similarity.
- perform a preliminary evaluation of the proposed approach for grouping semantically related change-sets.

The rest of the paper is organized as follows. In Section 2, we introduce the cosine similarity and grouping semantically related change-sets. In Section 3, we present the results of our preliminary evaluation. In Section 4, we present some related work. Finally, in Section 5, we state our conclusions and future work.

TABLE I
TERM FREQUENCIES WEIGHT(S)

Term	SaS	PaP	WH
1. Term Frequencies Count			
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38
2. Log Frequency Weight			
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58
3. Weight After Length Normalization			
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

II. GROUPING CHANGE-SETS

Software Practitioners tend to work on code usually spread across multiple change-sets, with multiple developers working on it at the same time. To group these change-sets, the semantic relations of the change-sets' comments can be utilized. For large scale software projects, the following can be observed as grouping criteria for change-sets (i) if and only if two comments have higher semantic similarity (or) (ii) if and only if two comments have same referencing work item or an issue number mentioned in it.

A. Cosine similarity

Cosine similarity is considered as a common measure for similarity computation [6]. Cosine similarity is used to fetch the similarity of words with respect to input query in regards to the text documents queried. To perform this computation, both the input query and the documents are converted into their respective unit vector of words ($\vec{x}_i$,$\vec{y}_i$). We use the below equation 1 to compute the cosine similarity here, where x_i, y_i are the term frequency (tf) weight of a unit vector (term) in the revision comment x, y. Term frequency is the number of times a term occurs in a revision comment.

$$sim(x, y) = \cos(\vec{x}, \vec{y}) = \vec{x} \cdot \vec{y} = \sum_{i=1}^{|V|} x_i y_i \tag{1}$$

Let's look at the case where we wish to calculate cosine similarity of the books: *Sense and Sensibility* (SaS), *Pride and Prejudice* (PaP) and *Wuthering Heights* (WH). Table I lists the Term Frequencies in each of the documents, Log Frequency Weight (calculated by formula "$w = 1 + \log_{10}(tf)$") and Length Normalized Weight. A vector can be length-normalized by dividing each of its components by its length. We use Level-2 normalization (commonly referred as Euclidean norm) as defined in equation 2:

$$\|\vec{x}\| = \sqrt{\sum_i x_i^2} \tag{2}$$

Finally we compute cosine similarity of documents as listed below:

- $cos(SaS, PaP) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$
- $cos(SaS, WH) \approx 0.79$

The resultant values can be used to assess the extent of similarity based on a predefined threshold values. In equation 3, we used '0.8' as a threshold to measure semantic similarity. We have conducted a preliminary data analysis using the available datasets and have considered 80% as a reasonable threshold for desirable similarity values. This value can be provided by the developers assessing the similarity. However, reducing the threshold can result in poor precision, where as increasing the threshold may result in poor recall. So, it is required for developers to follow the standard threshold which is widely accepted to avoid poor precision and poor recall.

$$x \overset{GC}{\rightsquigarrow} y \overset{def}{=} \begin{cases} 1 & \text{if } sim(x,y) \geq 0.8 \ \& \ datediff(x,y) < \alpha \\ 0 & otherwise \end{cases} \tag{3}$$

However, in some cases relying on the cosine similarity index can result in false positives. For example, in some project like SignalR project [4] developers tend to put generic comments like "*addressed code review comments* , *fixed formatting*, *made changes as per code review feedback*" for ongoing activities. As part of our study, we exclude these kind of change-sets from grouping. Also, we compare the time difference of change-sets before grouping them i.e. if the time difference is more than a few months then we consider such changes as not related.

B. Hash tags

Another technique for grouping change-sets is based on the hashtags associated with the commits of co-changed artifacts. For instance, if we consider the instances of commits from Table II of two large scale open source projects like SignalR [4] and NopCommerce [2] hosted on GitHub and Codeplex, we see that in the example-1, commits '`203cafc`' and '`5ad051a`' have similar comments. In case of example-2, the developers tend to associate work item or issue number in the comment for future references, like in Git repository, the hash tags #2376) are associated with the comments.

We group change-sets having same hash tag values based on the mathematical formulae given in equation 4.

$$x \overset{GC}{\rightsquigarrow} y \overset{def}{=} \begin{cases} 1 & \text{if } HashTags(x) \cap HashTags(y) \neq \emptyset \\ 0 & otherwise \end{cases} \tag{4}$$

C. Our Approach

The proposed approach is described in Algorithm 1. As shown in the algorithm, the initial steps required to build a postings list [6] for each revision so as to help in identifying the change-sets having same tokens. Once the posting list is built using Algorithm 2, the entire revision history is looped through (shown in line numbers 6-12) to get semantically related change-sets and group them in a

TABLE II
MOTIVATING EXAMPLES

Example 1
Commit: 203cafc **DumpingDisconnect.Net** Comment: Removing handling Disconnect message in the .NET Client as the server no longer sends Disconnect messages.
Commit: 5ad051a **DumpingDisconnect.JS** Comment: Removing handling Disconnect message in the .JS client as the server no longer sends Disconnect messages.
Example 2
Commit:9c762c4 #2376 Reject failed invocation with a single JS object representing **HubExce...**
Commit:f9bfbe3 #2376 Tests verifying **HubException** details are sent to clients
Commit: 35cfccb #2376 Flow **HubExceptions** which to clients even with detailed errors disabled

list. The '`getRelatedChangeSets`' subroutine returns semantically related change sets for the current iteration of the revision based equation (3, 4). The change-sets are then added to a dictionary of the list of semantically grouped change-sets.

Algorithm 1: Change set grouping algorithm

```
Result: Grouped Change Sets List
procedure group_changeset
revisionLists ← git.GetRevisionListIterator();
while rev in revisionLists do
    addToPostingList(rev.Id,rev.Comment);
end
Map(groupId, revisionList) changeSetGroups;
int groupId ← 1;
while revision rev in revisionLists do
    revList ← getRelatedChangeSets(rev.comment);
    changeSetGroup.Add(groupId,revList);
    groupId++;
end
end procedure
```

Algorithm 2 describes the subroutine used to create Postings List Index. The revision id and the associated comment are passed as input parameters to this routine. The parameters are further processed to remove all characters except '[0-9a-b]w' using regular expression match thereby tokenizing the string with words and their occurrences in the comments. Each token with its revision id to the Postings List Index is added to the postings list. The sample posting list is listed in Table III.

TABLE III
SAMPLE POSTING LIST

Token	Postings List
commerce	doc1, 4; doc2, 5;
tokenize	doc1, 50; doc3, 23;

Algorithm 2: Postings list builder subroutine

```
Result: Generate Postings List
Input: RevId and Comment
Procedure addToPostingsList
// get hashmap of terms with frequency count
tokensList ← getTokens(comment);
while token in tokensList do
    if postingsList.get(token.key) then
        postingsList.Get(token.key).Add(token);
    else
        postingsList.Add(token.Key, token);
    end
end
End Procedure
```

Algorithm 3 describes the subroutine responsible for returning semantically related change-sets. It accepts revision '`comment`' as a parameter. It converts the comments to tokens and queries the postings list dictionary for these tokens to fetch the posting list. Each posting list returned is reviewed against the criteria mentioned in the equation to return the related change-set list.

Algorithm 3: Fetch semanticaly related change-set

```
Result: Generate Postings List
Input: Comment
procedure getRelatedChangeSets
tokens ← getTokensList(comment);
postingsList ← getPostingsList(tokens);
List < revID, Comment > revisionList = null;
while post in postingsList do
    bool isRelevant ← false
    if hasSameHashTag(comment,post) then
        isRelevant ← true;
    end
    if cosine(comment,post.comment) > 0.8 then
        isRelevant ← true;
    end
    if isRelevant then
        revisionList.Add(post.revID,post.Comment);
    end
end
end procedure
```

III. PRELIMINARY EVALUATION

In this section, we provide details of our preliminary evaluation using our proposed approach to group change-sets in few C# programs. we used `NGit` [1] to traverse the `Git` repository change-sets. We used two open source projects (`SignalR` and `NopCommerce`) that had substantial revision history to evaluate our approach. `SignalR` is an ASP.Net library that provides real-time communication support to a web application and `NopCommerce` is an ASP.Net based e-commerce web system.

TABLE IV
MANUAL EVALUATION OF IDENTIFIED GROUPING

	NopCommerce		SignalR	
#	**Appropriate?**	**Commits & Notes**	**Appropriate?**	**Commits & Notes**
1	Yes	8bacf936baf4, efc731eeecb6, fd4ab9f67cce Enhancement for store owner to search unpublished and published products	Yes	ae9d5f7d57db, 8a2245b17d41 Modification for Forever Frame JS client
2	Yes	e6ec8e0b83ee, 782d3b87a6e3, c87537559940, d2792ada31c8 : Changes for product search and user friendly product name	Yes	0611ce61abe9, 261bb48fbca8 Changed the logic of addQs question mark query string detection
3	Yes	d40a89b56610, d0c04fc618d4 Modification related to custom validation	No	7996107ffbea, 877bcd59c454 Different instances of 'Removed unused code'
4	Yes	c5fe44ff76b8, 74e4a8e6c154, 28fb664d5c5f Modifications related to Shipping Address	Yes	240f6c58a5c3, de40de96a6b0, ff0bc98c2d34 Fix to ensure connection with LongPolling client
5	Yes	5e559b08a9ea, 050ddf2133e2 Enhancement related to filter shipments and Orders by warehouse	Yes	0b71d56, a60d923, 9c762c4, f9bfbe3e, 35cfccb Modifications related to HubExceptions
6	Maybe	305d4c1268b9, 3bf134f2c758, 9cdd0b2699b3 First two commits are related to store mapping to setting and third is related to store mapping to categories	Yes	8dc620093097, dd39b641bc27, 94ff30dd5821 Updates to crank for automation, and more Stress metrics
7	Yes	7a62bbdc9b24, c55d3897bf68 Localization changes for hard coded string	Yes	4148ea70f62d, 43aac84329b7 Handling of security errors in websockets
8	Yes	f020e98231f5, 95263d99979f Enhancement for friendly name of Affiliates	MayBe	a596aeb43c55, 5b47f9ba24a8 Updated self host sample.
9	No	188f4243ba1d, 8330d952235e First is the Fluent library update and second is the Entity Framework library update	Yes	143aa036367b, 7be649699e1d Changed Web Socket implementation to not send an empty frame at the end
10	Yes	57b7c7fdc445, 410eae6cd1f8 Modification for manufacturer store mapping	Yes	f81f6c2, 1118b15, cdaaa33,3 5b65d0 Ensure JS SSE & WS transports will attempt reconnecting multiple times
11	Yes	d2f341323abc, fbd49df6f2dd, f02244dc5fae Enhancement for shipping rate computation for 'FedEx', 'UPS' and 'by weight'	Yes	4cf7d2b, 730aa53, c6ffb08, 41317c9, 56e4002, 8dc3ac4 :Exception handling unresolved endpoints in ServiceBus scale-out
12	No	bcecd04eb644, 5cfc2977138, b559ca0aa2e4 Author forgot a file to checkin in previous commit, but comments were same in all the commits	Yes	5ad051a9822b, 203cafcd1cbe Handling of disconnect message in JS and .Net
13	Yes	c3782eef4eba, b80d4cdecfa1 Added "Order paid" message template sent to a customer	Yes	1162f9163ba8, 47c7084ec3a4, ccdade4488ef added checks for null and empty values in the send and group methods in hub and persistent connection
14	-	-	Yes	bddcb70, 7dfa876, 13f2ffe, 9ff238e, 61c0c5c, 212fb4e Modification as per the static code analysis tool 'FxCop', Usually these are not related changes as FxCop is used to check naming conventions

We calculated the basic descriptive statistics for the number of commits grouped together. As listed in Table V, in case of NopCommerce project, 357 change-set groups were created with 1017 commits which are 19.44% of the actual commits with a maximum of 11 revisions and an average of 2.85 commits per change-set. In SingnalR project, 301 change-set groups were created with 969 commits which are 21.49% of the actual commits with a maximum of 22 revisions and an average of 3.21. The figure 1 displays a Scatter Plot view of length versus number of change-set groups created for NopCommerce and SignalR projects respectively. Overall, there is a little dispersion in the values as evident by the standard deviation.

TABLE V
CHANGESET GROUP STATISTICS

Project	**NopCommerce**	**SignalR**
Total commits	5229	4509
Active Since	2009	2011
# of groups	357	301
# of commits grouped	1017	969
% of commits grouped	19.44	21.49
Max	11	22
Avg	2.85	3.21
StDev	1.48	2.7

To corroborate the results of our algorithm, we have made attempts to conduct an inspection of a few random samples [3] from the 658 change-set groups identified by our algorithm. We inspected 13 samples from each NopCommerce and SignalR to verify whether the revisions were grouped to a single

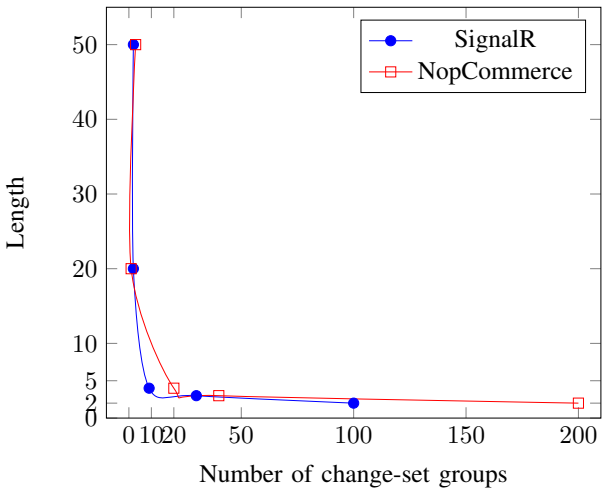

Fig. 1. Scatter plot of length versus change set groups

purpose or not. By verifying the revision files, comments, and diff code - we were able to judge whether the grouping made sense. We have used a statistical random sample size calculator [3], which enabled us to generalize our results with a margin of an error of 20% and confidence level of 80%. The results of our inspection are listed in Table IV.

In '`NopCommerce`', out of 13 samples 11 were semantically related. In two samples which were not semantically related, '`Sample 6`' had two revisions '`305d4c1268b9, 3bf134f2c758`' related to '`store mapping changes for settings`' but its third revision '`9cdd0b2699b3`' was related to '`store mapping for categories`'. Additionally, their comments appeared to be semantically related but as per the code paths, it seemed that they were not actually related to each other. In the other '`Sample 12`',the first revision '`188f4243ba1d`' was related to the latest update of '`fluent`' library and the second revision '`8330d952235e`' was related to the latest update of '`EntityFramework`' library and hence they were not semantically related.

In '`SignalR`', out of 14 samples, 11 were semantically related. In three samples that were not semantically related, '`Sample 8`' had two revisions related to self-host changes as per the comments but post analyzing the changed code a semantic relation could not be established in both the revision changes, hence we marked it as '`MayBe`'. In '`Sample 3`', the revisions were done to remove unused code in multiple places but the author provided the same comments for both of them which resulted in high cosine similarity responsible for their grouping. In '`Sample 14`', it had the fixes of the naming conventions identified by the static analysis tool named '`FxCop`'.

We have observed that our approach is able to semantically group change-sets which has the potential to enhance the identification of logical dependency with a margin of 20% error in detection. As per our sample analysis, we did not notice any instance of grouping by `HashTag` that is not semantically related, however the groups which were created by semantic (`cosine`) similarity had few instances that were not related.

IV. Related Work

Gall et al. are the first to introduce the concept of logical coupling [9] by analyzing the dependencies in 20 different product releases of a telecommunications switching system. D'Ambros et al. have made attempts to visualize logical coupling using an interactive visualization approach called Evolution Radar [7]. This approach was not composite and extensive enough for large software systems. Graves et al. [10] have shown that the future occurrence of faults can be easily predicted by the past revision histories of the software system. However, this reliability of this prediction can be questioned as it was never evaluated against a real-world evolving software product. Logical coupling has also been employed to predict changes in the software product [15] and was used to infer code decay [8]. There are many such use-cases which are introduced and practiced by software researchers. Mockus et al. [12] have found that the rate of widespread of a change over sub-systems and its related artifacts is a strong indicator for a presence of defects in a respective change. However, it requires a strong empirical validation on a real-world data set.

Ambros et al. [5] have reverse engineered a software system using an interactive visualization technique called the Evolution Radar, which can effectively break down the amount and complexity of the logical coupling information. Manishankar et al. [13] proposed new measurement for improving detection accuracy of evolutionary coupling by blending the concept location in a code-base to determine whether the changes to the co-changed entities are corresponding in nature and are thus related. Gustavo Ansaldi Oliva et al. [14] have proposed an approach to group timely-close and semantically-related change-sets containing the same author and commit message by using a sliding time window concept. In spite of such significant research being brought out by a variety of software practitioners, the semantic relationship between the artifacts was never made by linking the approach to detect logical coupling. The existing approaches are built under the assumptions that the developer will check-in all the related files in a single commit, whereas this is not a practical scenario. In contrast to existing state-of-art approaches, our approach is considerably different as it considers grouping of change-sets semantically in order to enhance the detection of logical coupling.

V. Conclusion and Future Work

In this paper, we proposed a simple approach to group semantically related change-sets in atomic-commit featured Version Control Systems before performing logical coupling identification. We used Cosine Similarity and work item hash-tag match to group various change-sets. We were able to group almost 20% of the revisions. As part of our validation, we implemented the proposed algorithms in our approach to group various change-sets and presented preliminary evaluation results for the same. Our evaluation revealed promising results with a margin of error 20% in 'Cosine Similarity'

grouping of change-sets. Whereas in 'HashTag' evaluation, we found that almost all the samples were semantically related. Intuitively, the results could easily be interpreted to conclude that 'HashTag' grouping provided a high degree of accuracy in grouping the change-sets, whereas 'Cosine Similarity' had few instances of false positives.

As part of our future work, we plan to detect hashtags by using pattern recognition techniques using a new algorithm that can reduce the possible HashTag patterns from the revision comments. We also plan to develop a technique to filter semantically unrelated files and exclude false positives. We observed that most of the false positive are related to ongoing design tasks "like refactoring, merge of branches". Therefore we will plan to develop a filtering rule to exclude such change-sets as part of our future data sets. In regards to our present analysis, we have only targeted the main/trunk branch for detecting change-set groups. In the future, we would explore the possibilities to work with multiple branches. In regards to existing approaches, we ought to conduct an empirical study to understand the efficiency of our approach against the rest. We will be working further to enhance the effectiveness of our approach by evaluating it against large code-base.

REFERENCES

[1] ngit. www.github.com/mono/ngit.

[2] nopcommerce - asp.net open-source e-commerce shopping cart solution. www.nopcommerce.com.

[3] Random sample calculator. www.raosoft.com/samplesize.html.

[4] Signalr - incredibly simple real-time web for .net. www.signalr.net.

[5] M. D. Ambros and M. Lanza. Reverse engineering with logical coupling. In *Reverse Engineering, 2006. WCRE'06. 13th Working Conference on*, pages 189–198. IEEE, 2006.

[6] H. S. Christopher Manning, Prabhakar Raghavan. *Introduction to Information Retrieval*. Camebridge University Press, 2008.

[7] M. D'Ambros, M. Lanza, and M. Lungu. Visualizing co-change information with the evolution radar. 35(5):720–735, 2009.

[8] S. Eick, T. Graves, A. Karr, A. Mockus, and P. Schuster. Visualizing software changes. 28(4):396–412, 2002.

[9] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Software Maintenance, 1998. Proceedings., International Conference on*, pages 190–198, 1998.

[10] T. Graves, A. Karr, J. Marron, and H. Siy. Predicting fault incidence using software change history. 26(7):653–661, 2000.

[11] A. Kumar. Modern information retrieval: A brief overview. *Bulliten of IEEE Computer Society Technical Committee on Data Engineering*, 4(24):35–43, November 2001.

[12] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.

[13] M. Mondal, C. K. Roy, K. Schneider, et al. Improving the detection accuracy of evolutionary coupling by measuring change correspondence. In *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*, pages 358–362. IEEE, 2014.

[14] G. A. Oliva, F. Santana, M. Gerosa, and C. de Souza. Preprocessing change-sets to improve logical dependencies identification. In *Proceedings of the 6th International Workshop on Software Quality and Maintainability*, 2012.

[15] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. 31(6):429–445, 2005.