Blending Conceptual and Evolutionary Couplings to Support Change Impact Analysis in Source Code

Huzefa Kagdi[1], Malcom Gethers[2], Denys Poshyvanyk[2], Michael L. Collard[3]

[1]Department of Computer Science
Winston-Salem State University
Winston-Salem, NC 27110
kagdih@mst.edu

[2]Computer Science Department
The College of William and Mary
Williamsburg, VA 23185
{mgethers, denys}@cs.wm.edu

[3]Department of Computer Science
The University of Akron
Akron, OH 44325
collard@uakron.edu

***Abstract* — The paper presents an approach that combines conceptual and evolutionary techniques to support change impact analysis in source code. Information Retrieval (IR) is used to derive conceptual couplings from the source code in a single version (release) of a software system. Evolutionary couplings are mined from source code commits. The premise is that such combined methods provide improvements to the accuracy of impact sets. A rigorous empirical assessment on the changes of the open source systems *Apache httpd*, *ArgoUML*, *iBatis*, and *KOffice* is also reported. The results show that a combination of these two techniques, across several cut points, provides statistically significant improvements in accuracy over either of the two techniques used independently. Improvements in recall values of up to 20% over the conceptual technique in *KOffice* and up to 45% over the evolutionary technique in *iBatis* were reported.**

I. Introduction

According to Arnold and Bohner [8] software-change impact analysis, or simply impact analysis (IA), is defined as the determination of potential effects to a subject system resulting from a proposed software change. The premise of impact analysis is that a proposed change may result in undesirable *side effects* and/or *ripple effects*. A side effect is a condition that leads the software to a state that is erroneous or violates the original assumptions/semantics as a result of a proposed change. A ripple effect is a phenomenon that affects other parts of a system on account of a proposed change. The task of an impact analysis technique is to estimate the (complete closure of) ripple effects and prevent side effects of a proposed change. The scope of the analyzed and estimated software artifacts may include requirements, design, and source code.

IA is a key task in software maintenance and evolution. Decades of research efforts have produced a wide spectrum of approaches, ranging from the traditional static and dynamic analysis techniques [10, 28, 30, 31, 37, 38] to the contemporary methods such as those based on Information Retrieval (IR) [11, 20, 34] and Mining Software Repositories (MSR) [46]. Although ample progress has been made, there still remains much work to be done in further improving the effectiveness (*e.g.*, accuracy) of the state-of-the-art IA techniques. Our goal is to develop a new and improved IA approach by reusing some of the existing contemporary solutions. Central to our approach are the information sources that are developer centric (*e.g.*, comments and identifiers, and commits and commit practices), rather than artifact centric (*e.g.*, static and dynamic dependencies such as call graphs).

In this paper, we present an approach that combines conceptual and evolutionary couplings to support IA in source code. Conceptual couplings capture the extent to which domain concepts and software artifacts are related to each other. This information is derived using Information Retrieval based analysis of textual software artifacts that are found in a single version of software (*e.g.,* comments and identifiers in a single snapshot of source code). This analysis focused on a single version is consistent with its previous usages in IA [3, 34]. Evolutionary couplings capture the extent to which software artifacts were co-changed. This information is derived from analyzing patterns, relationships, and relevant information of source code changes mined from multiple versions in software repositories.

The core research philosophy behind our approach is that *present+past* of software systems leads to better IA. For IA, both single (present) and multiple versions (past) analysis methods have been utilized independently, but their combined use has not been previously investigated. Our larger research objective is focused on the investigation of these combinations of IR and MSR techniques for IA. The combinations presented in this paper are a fundamental and necessary baseline step in this direction. We investigate two different combinations, *i.e.*, disjunctive and conjunctive, and compute impact sets at varying source code granularity levels (*e.g.*, files and methods). Our principal research hypothesis is that such combined methods provide improvements to the accuracy of impact sets.

An extensive empirical study on hundreds of changes from the open source systems, such as *Apache httpd*, *ArgoUML*, *iBatis*, and *KOffice* was conducted to test the research hypothesis. The results of the study show that the disjunctive combination of IR and MSR techniques, across several cut points (impact set sizes), provides statistically significant improvements in accuracy over either of the two standalone techniques. For example, the disjunctive method reported improvements in recall values of up to 20% over the conceptual technique in *KOffice* and up to 45% improvement over the evolutionary technique in *iBatis*. These results are encouraging considering that the combinations do not require an overly complex blending of two separate approaches.

The rest of the paper is organized as follows. Section II provides a brief discussion of the related work, whereas section III presents our combined approach. The empirical assessment is presented in IV. We conclude in Section V.

II. Background and Related Work

The paper addresses a key maintenance task of software change impact analysis by involving conceptual and evolutionary couplings. There is a rich volume of literature covering each of these areas. Our intention is not to cover every individual work exhaustively, but to provide a breadth of the solutions offered to the problem and by the solutions.

A. Software Change Impact Analysis (IA)

Dependency analysis and *traceability* analysis are the two primary methodologies for performing impact analysis. Broadly, dependency analysis refers to impact analysis of software artifacts at the same level of abstraction (*e.g.*, source code to source code or design to design). Traceability analysis refers to impact analysis of software artifacts across different levels of abstractions (*e.g.*, source code to UML). Various dependency-analysis methods based on call graphs, program slicing [19], hidden dependency analysis [14, 41, 45], lightweight static analysis approaches [29, 31], concept analysis [40], dynamic analysis [28], hypertext systems, documentation systems, UML models [9], and Information Retrieval [3] are already investigated in the literature. Queille et al. [36] proposed an interactive process in which the programmer, guided by dependencies among program components (*i.e.*, classes, functions), inspects components one-by-one and identifies the ones that are going to change – this process involves both searching and browsing activities. This interactive process was supported via a formal model, based on graph rewriting rules [13].

Coupling measures have been also used to support impact analysis in OO systems [10, 44]. Wilkie and Kitchenham [44] investigated if classes with high CBO (Coupling Between Objects) coupling metric values are more likely to be affected by change ripple effects. Although CBO was found to be an indicator of change-proneness in general, it was not sufficient to account for all possible changes. Briand et al. [10] investigated the use of coupling measures and derived decision models for identifying classes likely to be changed during impact analysis. The results of an empirical investigation of the structural coupling measures and their combinations showed that the coupling measures can be used to focus the underlying dependency analysis and reduce impact analysis effort. On the other hand, the study revealed a substantial number of ripple effects, which are not accounted for by the highly coupled (structurally) classes.

More recent work appears in [20, 38], where proposed tools can help navigate and prioritize system dependencies during various software maintenance tasks. The work in [20] relates to our approach in as much as it also uses lexical (textual) clues from the source code to identify related methods. Several recent papers presented algorithms that estimate the impact of a change on tests [27, 39]. A comparison of different impact analysis algorithms is provided in [30].

B. Conceptual Information in Software

Identifiers used by programmers for names of classes, methods, or attributes in source code or other artifacts contain important information and account for approximately half of the source code in software [17]. These names often serve as a starting point in many program comprehension tasks [12], hence it is essential that these names clearly reflect the concepts that they are supposed to represent, as self-documenting identifiers decrease the time and effort needed to acquire a basic comprehension level for a programming task [4].

The software maintenance research community recently recognized the problem of extracting and analyzing conceptual information in software artifacts. IR-based methods have been applied to support practical tasks. For instance, IR methods have been successfully used to support feature location [32, 35], traceability link recovery [2, 16] and impact analysis [3, 34]. We do not discuss other applications of IR-based techniques in the context of software maintenance due to space limitations, however, interested readers are referred to [7] for such an overview.

C. Evolutionary Information in Software Repositories

The term MSR has been coined to describe a broad class of investigations into the examination of software repositories (*e.g.*, *Subversion* and *Bugzilla*). The premise of MSR is that empirical and systematic investigations of repositories will shed new light on the process of software evolution, and the changes that occur over time, by uncovering pertinent information, relationships, or trends about a particular evolutionary characteristic of the system.

We now briefly discuss some representative works in MSR for mining of evolutionary couplings. Zimmerman et al. [46] used *CVS* logs for detecting evolutionary coupling between source code entities. Association rules based on itemset mining were formed from the change-sets and used for change-prediction. Canfora et al. [11] used the bug descriptions and the *CVS* commit messages for the purpose of change prediction. An information retrieval method is used to index the changed files, and commit logs, in the *CVS* and the past bug reports from the *Bugzilla* repositories.

In addition, conceptual information has been utilized in conjunction with evolutionary data to support several other tasks, such as assigning incoming bug reports to developers [5, 21, 25], identifying duplicate bug reports [42], estimating time to fix incoming bugs [43] and classifying software maintenance requests [18]. Finally, we conducted a comprehensive literature survey on MSR approaches during the prologue of this work [22]. Xie's online bibliography and tutorial[1] on MSR is another well-maintained source.

The above discussion shows that both IR and MSR have been used for impact analysis. Also, IR techniques have been applied to software repositories. Our work differs in that we limit the use of IR to a single snapshot (*i.e.*, to derive

[1] https://sites.google.com/site/asergrp/dmse

conceptual couplings) of source code and data mining techniques are used on past commits of source code (*i.e.*, to derive evolutionary couplings). To the best of our knowledge, such a combined use of IR and MSR has not been presented elsewhere or empirically investigated before in the research literature. Our approach builds on existing solutions, but synergizes them in a new holistic technique.

III. A Combined Approach to Impact Analysis

A typical IA technique takes a software entity in which a change is proposed or identified and estimates other entities that are also potential change candidates, referred to as an estimated impact set. Our general approach computes the estimated impact set with the following steps:

Step 1: **Select** the first software entity, e_s, for which IA needs to be performed. For example, this first entity could be a result of a feature location activity. Note that IA starts with a given entity.

Step 2: **Compute** conceptual couplings with IR methods from the release of a software system in which the first entity is selected. Let $EI(e_s)$ be the set of entities that are conceptually related to the entity from Step 1.

Step 3: **Mine** a set of commits from the source code repository and compute evolutionary couplings. Here, only the commits that occurred before the release in the above step are considered. Let $EM(e_s)$ be the set of entities that are evolutionary coupled to the entity from Step 1.

Step 4: **Compute** the estimated impact set, $E(e_s)$, from the combinations of couplings computed in steps 3 and 4.

We now discuss the details of these steps, especially conceptual and evolutionary couplings, and their combinations.

A. Conceptual Couplings

We use *conceptual similarity* as a primary mechanism of capturing conceptual coupling among software entities. This measure is designed to capture the conceptual relationship among documents. Formally, the conceptual similarity between software entities e_k and e_j (where e_k and e_j can be methods), is computed as the cosine between the vectors ve_k and ve_j, corresponding to e_k and e_j in the vector space constructed by an IR method (*e.g.*, Latent Semantic Indexing - LSI):

$$CSE(e_k, e_j) = \frac{ve_k^T ve_j}{|ve_k|_2 \times |ve_j|_2}$$

As defined, the value of $CSE(e_k, e_j) \in [-1, 1]$, as *CSE* is a cosine in the Vector Space Model (VSM). For source code documents, the entities can be attributes, methods, classes, files, etc. Computing attribute-attribute or method-method similarities, *CSE* is straightforward (*e.g.*, e_k and e_j are substituted by a_k and a_j in the *CSE* formula), while deriving method-class or class-class *CSE* requires additional steps. We define the conceptual similarity between a method m_k and a class c_j (*CSEMC*) with t number of methods as follows:

$CSEMC(m_k, c_j) = \sum_{q=1}^{t} CSE(m_k, m_{jq})$ /t, which is an average of the conceptual similarities between method m_k and all the methods from class c_j. Using *CSEMC* we define the conceptual similarity between two classes (*CSEBC)* $c_k \in C$ with r number of methods and $c_j \in C$ (where C is a set of classes in software) as: $CSEBC(c_k, c_j) = \sum_{l=1}^{r} CSEMC(m_{kl}, c_j)$ /r, which is the average of the similarity measures between all unordered pairs of methods from class c_k and class c_j. The assumption, which is used in defining *CSE*, *CSEMC*, and *CSEBC*, is that if the methods of a class relate to each other, then the two methods or classes are also related. For more details and examples, please refer to our preliminary work on conceptual coupling measures [33, 34].

To analyze conceptual information in a given release of a software system, the source code is parsed using a developer-defined granularity level (*i.e.*, methods or files). A corpus is created, so that each software artifact will have a corresponding document in it. We rely on *srcML* [15] for the underlying representation of the source code and textual information. *srcML* is an *XML* representation of source code that explicitly embeds the syntactic structure inherently present in source code text with *XML* tags. The format preserves all the original source code contents including comments, white space, and preprocessor directives, which are used to build the corpus.

B. Evolutionary Couplings

We mine the change history of a software system for evolutionary relationships. In our approach, evolutionary couplings are essentially mined patterns of changed entities. We employ *itemset* mining [1], as the specific order of change between artifacts is not considered. This unordered set allows the computed evolutionary couplings to be consistent with the conceptual couplings (with no change order between coupled artifacts).

Formally, a *software change history*, *SCH*, is a set of change-sets (commits) submitted to the source-control repository during the evolution of the system in the time interval λ. Also, let $E = \bigcup_{i=1}^{m} csi$ be the set of m entities, each of which was changed in at least one change-set. An unordered evolutionary coupling is a set of source code entities that are found to be recurring in at least a given number (σ_{min}) of change-sets, $ec_u = \{e_p, e_q, \ldots, e_o\}$ where each $e \in E$ and there exists a set of related change-sets, $S(ec) = \{c \in SCH \mid ec \subseteq c\}$ with its cardinality, $\sigma(ec) = |S(ec)| \geq \sigma_{min}$. The $\sigma(ec)$ value of a mined pattern is termed its *support* value in the data mining vocabulary. Similarly, the σ_{min} value is termed as *minimum support value*. Also, let $EC = \bigcup_{i=1}^{k} eci$ be a set of all the evolutionary couplings observed in *SCH*.

For any given software entity from E, which could be the first point e_s for impact analysis, we compute all the association rules from the mined evolutionary couplings where it occurs as an antecedent (*lhs*) and another entity from E as a consequent (*rhs*). Simply put, an association rule gives the conditional probability of the *rhs* also occurring when the *lhs* occurs, measured by a *confidence* value. That is, an association rules is of the form $lhs \Rightarrow rhs$.

When multiple rules are found for a given entity, they are first ranked by their confidence values and then by their support values; both in a descending order (higher the value, stronger the rule). We allow a user specified cut-off point to pick the top *n* rules. Thus, $EM(e_s)$ is the set of all consequents in the selected *n* rules.

Broadly, the presented approach for mining fine-grained evolutionary couplings and prediction rules consists of three steps:

1) Extract Change-sets from Software Repositories

Modern source-control systems, such as *Subversion*, preserve the grouping of several changes in multiple files to a single change-set as performed by a committer. This information can be easily obtained (*e.g.*, *svn log* and *pysvn*).

2) Process to Fine-grained Change-sets

The differences in a file of a change-set can be readily obtained at a line-level granularity (*e.g.*, with *diff* utility). In this case, the line differences need to be mapped to the corresponding fine-grained differences in the syntactic constructs. Our approach employs s*rcDiff*, a lightweight methodology for fine-grained differencing of files in a change-set. *srcDiff* extends the *srcML* representation by also marking the regions of changes to the code in a collection of difference elements. Information about the syntactic changes to the code is found using an XPath query.

3) Mine Evolutionary Couplings

A mining tool, namely *sqminer,* was previously developed to uncover evolutionary couplings from the set of commits (processed at fine-granularity levels with *srcDiff* should the need be). The basic premise of *sqminer* is if the same set of source code entities frequently co-changes then there is a potential evolutionary coupling between them. *sqminer* supports mining of both unordered and ordered patterns. These patterns are used to generate association rules that serve as prediction rules for source code changes. *sqminer* has already been applied previously to mine co-changes at the file level [26], uncover/discover traceability links [24], and mine evolutionary couplings of localized documents [23].

```
# procedure to get the n ranked concep couplings for the given entry
function getConceptual(anentity, n)
   CCBE := conceptualBase (src_code, granularity, ia_params)
   # get the top n couplings for the only the given entity
   return slice (CCBE, anentity, n)

# procedure to get the n ranked evol couplings for the given entry
function getEvolutionary(anentity, n)
   ECBE := evolutionaryBase (history, granularity, m_params)
   # get the top n couplings for the only the given entity
   return slice (ECBE, anentity, n)

# procedure to form a corpus with LSI and compute conc coupling
function conceptualBase (src_code, granularity, ia_params)
   # recomputed only if needed
   # form a corpus with LSI
   corpus := lsi (src_code, granularity, ia_params)
   # compute conceptual couplings between all pairs of
   # entities in the corpus
   CCBE := formConceptual (corpus)
   # CCBE is sorted by similarity values
   return CCBE

# procedure to mine evolutionary couplings and form
# association rules from a given commit history
function evolutionaryBase (history, granularity, m_params)
   # recomputed only if needed
   # mine patterns of co-changes entities from the history
   # and then form binary association rules
   ECBE := mineEvolutionary (history, granularity, m_params)
   # ECBE is sorted by confidence and support values
   return ECBE
```

Figure 1. The procedures for computing conceptual and evolutionary couplings

C. Disjunctive and Conjunctive Combinations

With regards to combining conceptual and evolutionary dependencies, there is a pertinent research question. *Should the union or intersection of the two estimations be considered*, *i.e.*, $EI(e_s) \cup EM(e_s)$ or $EI(e_s) \cap EM(e_s)$? This question may not be an issue, if both $EI(e_s)$ and $EM(e_s)$ predict the same estimation set. If the estimation sets differ, taking their union could result in increased recall; however, at the expense of decreased precision (if a large number of false-positive are estimated). Alternatively, taking only the intersection imposes a stricter constraint that could result in increased precision; however, at the expense of decreased recall.

The combined approaches for IA that use the union and intersection of estimations of conceptual and evolutionary estimations are termed as *disjunctive approach* and *conjunctive approach* respectively (see Figure 2). That is, $E(e_s)_{\cup} = EI(e_s) \cup EM(e_s)$ and $E(e_s)_{\cap} = EI(e_s) \cap EM(e_s)$. Our approach supports both of these combinations. Both approaches require the user to specify a starting entity as well as a cut point for deriving an estimated impact set. For a

```
# procedure to compute a disjunctive impact set
# the ranking parameters that control the appropriate entities
# (recursion) to get from both couplings are discarded for brevity
function disjIA(anentity, cutpoint)
   # anentity: initial entity for IA;  # cutpoint: size of the impact set
   # look for equal contributions from both
   # get the top cutpoint/2 conceptual couplings
   EI := getConceptual(anentity, cutpoint/2)
   # get the top cutpoint/2 evolutionary couplings
   EM := getEvolutionary(anentity, cutpoint/2)
   # did we get the equal share?  If not try again.
   if |EI U EM| < cutpoint
      return (EI U EM U disjIA(anentity, cutpoint - |EI U EM|)
   # a disjunctive set is the union of the sets EI and EM
   return (EI U EM)

# procedure to compute a conjunctive impact set
# the ranking parameters that control the appropriate entities to get
function conjIA(anentity, cutpoint)
   # anentity: initial entity for IA
    # get the top cutpoint conceptual couplings
   EI := getConceptual(anentity, cutpoint)
   # get the top cutpoint evolutionary couplings
   EM := getEvolutionary(anentity, cutpoint)
   if | EI ∩ EM)| < cutpoint
      return (  (EI ∩ EM) U conjIA(anentity, cutpoint - |EI ∩ EM|)
   # a conjunctive set is the intersection of the sets EI and EM
   return (EI ∩ EM)
```

Figure 2. The procedures for disjunctive and conjunctive impact analysis

Table I. Example showing the accuracy gains of the disjunctive impact analysis method on the bug# 47087 in *Apache httpd*

	Conceptual	Evolutionary	Disjunctive
1	**/server/protocol.c**	/modules/http/byterange_filter.c	**/server/protocol.c**
2	/modules/proxy/mod_proxy_http.c	**/modules/http/http_protocol.c**	/modules/proxy/mod_proxy_http.c
3	/modules/debugging/mod_bucketeer.c	/modules/proxy/mod_proxy_ftp.c	**/modules/http/byterange_filter.c**
4	/server/core_filters.c	/server/core.c	/modules/http/http_protocol.c
5	/modules/http/byterange_filter.c	/include/ap_mmn.h	/server/core_filters.c

given cut point, μ, provided by the user, we compute the impact set of the disjunctive method $E(e_s)_{\cup}$ by determining $EI(e_s)$ and $EM(e_s)$ such that the cardinality of each set is equal (or the cardinality $EI(e_s)$ is larger by one entity) and the cardinality of their union equals μ. A similar approach is taken to obtain the impact set of the conjunctive method; however, in this case we ensure the cardinality of the intersection equals μ. The procedures used to compute the underlying conceptual and evolutionary couplings are shown in Figure 1. They use typical sets of parameters needed for LSI and itemset mining algorithms.

D. Examples

In order to explain what each technique finds and the issues that arise in the combination of the techniques, we present an example from a real system. In Apache httpd commit# 888310 addresses the bug#47087 [2] regarding "*Incorrect request body handling with Expect: 100-continue if the client does not receive a transmitted 300 or 400 response prior to sending its body*." In this revision to fix the bug there were three source code files which needed to be changed (/modules/http/http_filters.c, /modules/http/http_protocol.c, and /server/protocol.c). In order to perform impact analysis, the developer must have a starting entity. For this example, let us assume the developer discovers, through feature location, that fixing the problem requires modifying /modules/http/http_filters.c. From this point the developer can perform impact analysis to discover other entities which also require modification. Using conceptual and evolutionary couplings for impact analysis, we obtain the results in Table I. As standalone techniques neither conceptual nor evolutionary coupling are capable of establishing 100% recall. Conceptual coupling ranks /server/protocol.c as first in the ranked list, but ranks /modules/http/http_protocol.c as 91st, whereas evolutionary coupling ranks /modules/http/http_protocol.c second in the ranked list, but ranks /server/protocol.c as 16th. We can combine the results using our disjunctive approach. This results in the set of entities that also appear in Table I. Here we can see that when combined, the couplings are capable of identifying all methods requiring modification within an impact set, i.e., cut point, of five methods. Note that our disjunctive and conjunctive approaches result in sets as opposed to ranked lists (*i.e.,* the entities are unordered).

[2] https://issues.apache.org/bugzilla/show_bug.cgi?id=47087

IV. Case Study

In this section we describe the empirical assessment of our approach. We describe our study following the Goal-Question-Metrics paradigm [6], which includes *goals*, *quality focus, and context*. In the context of our case study we aim at addressing the research questions (RQs):

- **RQ1**: Does combining conceptual and evolutionary couplings improve the accuracy of IA when compared to the two standalone techniques?
- **RQ2**: Does the choice of granularity, *i.e.*, file or method, affect the accuracy of IA of standalone techniques and their combination?

The *goal* of the case study is to investigate these research questions. The *quality focus* is on providing improved accuracy, while the *perspective* was of a software developer performing a change task, which requires extensive impact analysis of related source code entities. Our two research questions directly address the effectiveness and expressiveness of an IA solution. With regards to effectiveness, it is desirable to have a technique that provides all, and only, the impacted entities, i.e., prevents false positives and false negatives in the estimated impact set as much as possible. Additionally, it is desirable to provide the developers with the ability to apply the IA technique at various source code granularities. Our approach offers this feature; however, an important issue is to assess the change in effectiveness at different levels of granularity.

A. Accuracy Metrics

1) Precision and Recall

Impact analysis techniques are typically assessed with the two widely used metrics *precision* (*i.e.*, inverse measure of false positives) and *recall* (*i.e.,* inverse measure of false negatives). These metrics are computed from the estimated impact set produced from a technique and the actual impact set from the established ground truth (*e.g.*, change-sets/patches after the proposed change is actually implemented or developer verification).

For a given entity e_s (*e.g.,* file and method) let $EI(e_s)$ be the set of entities that are conceptually related to the entity e_s. Let R_i be the set of actual or correctly changed entities with the entity e_s. The precision of conceptual couplings, P_{EI}, is the mean percentage of correctly estimated changed entities over the total estimated entities. The recall of conceptual couplings, R_{EI}, is the mean percentage of correctly estimated changed entities over the total correctly changed entities.

$$P_{EI} = \frac{1}{n}\sum_{i=1}^{n}\frac{|EI_i \cap R_i|}{|EI_i|}\times 100\% \qquad R_{EI} = \frac{1}{n}\sum_{i=1}^{n}\frac{|E_i \cap R_i|}{|R_i|}\times 100\%$$

The precision and recall values for evolutionary couplings, disjunctive, and conjunctive methods can be similarly computed. The set $EM(e_s)$ would indicate the set of entities that are related to a known entity e_s based on evolutionary couplings. The sets $E(e_s)_{\cup}$ and $E(e_s)_{\cap}$ would

Table II. Characteristics of the subject systems considered in the empirical evaluation.

System	Version	LOC	Files	Methds	Terms
Apache(httpd)	2.2.3	311K	782	n/a	6583
ArgoUML	0.28	367K	1,995	n/a	9384
iBatis	3.0.0-216	70K	774	n/a	3772
KOffice 2.0.91	2.0.91	231K	6.5K	n/a	48513
KOffice 2.0.1	2.0.1	257K	6.7K	68.4K	32212

indicate the couplings from the disjunctive and conjunctive methods respectively.

B. Evaluated Subject Systems

The *context* of our study is characterized by a set of four open source software systems, namely *Apache httpd*, *ArgoUML*, *iBatis*, and *KOffice*. The selected set of systems represents different primary implementation languages (*e.g.,* C/C++ and Java), size, development environment, and application domain. *Apache httpd* is an open source implementation of an HTTP server, which focuses on providing a robust and commercial-grade system. *ArgoUML* is a Java implementation of a UML diagramming tool. The *iBatis Data Mapper* framework provides a mechanism that simplifies the use of relational database systems with *Java* and *.NET* applications. *KOffice* is an application suite that includes various office productivity applications such as word (*i.e.*, *KWord*) and spreadsheet (*i.e.*, *KSpread*) processing. Specifics of various system characteristics appear in Table II.

C. Evaluation Procedure

The source code changes in software repositories, *i.e.*, commits, are used for the evaluation purpose. Our general evaluation procedure consists of the following steps:

1. Compute conceptual couplings on a release (*e.g.*, *KOffice 2.0.91*) of a subject system – *Conceptual Training Set*.
2. Mine evolutionary couplings (and association rules) from a set of commits in a history period prior to the selected release in Step 1 – *Evolutionary Training Set*.
3. Select a set of commits in a history period after the selected release in Step 1 –*Testing Set*. Each commit in the testing set is considered as an actual impact set, *i.e.*, the ground truth, for evaluation purposes.
4. Derive disjunctive and conjunctive impact sets from the two training sets for each commit in the testing set.
5. Compute accuracy metrics for the two standalone techniques and their two combinations.
6. Compare standalone and combination accuracy results.
7. Repeat the above steps for all the considered subject systems and releases.

The details of the training and testing sets are detailed next.

1) Conceptual training sets - Corpora

We generated two sets of corpora from the subject systems corresponding to the granularity of *documents* at the *file* and *method* levels. The process of generating a corpus consisted of extracting textual information, *i.e.*, identifiers and comments, from the source code for the specific granularity level. The identifiers and comments, *i.e.*, *terms*, from each file (or a method if that is the chosen granularity) formed a *document*, whereas a complete collection of these documents formed a corpus. Once a corpus was built, LSI was used to index its *term-by-document* co-occurrence matrix. Conceptual couplings between source code documents, *i.e.*, files or methods, were then computed (see section III). Details of the corpora, including the releases indexed, are provided in Table II. The associated computing time was consistent with the previous uses [3, 34].

Table III. Evolutionary training and (testing) datasets used for the empirical evaluation

System	History	# of Commits	# of Entities
Apache(httpd)	2.2.9-2.3.5	1736 (287)	2086 (982)
ArgoUML	0.24-0.28	3375 (773)	4217 (621)
iBatis	3.0.0-190_b1 - 3.0.0-240_b10	108 (40)	461 (118)
KOffice 2.0.91	2.0.0-2.0.91	2749 (522)	5580 (1072)
KOffice 2.0.1	2.0.0-2.0.2	763 (255)	1233 (533)
KOffice 2.0.1*	2.0.0-2.0.2	577 (192)	5530 (1438)

2) Evolutionary training sets

In order to obtain evolutionary training sets we selected a period of history, which preceded the version of the system used to build the corpus. For example, the corpus created for *Apache httpd* used the source code from version 2.2.3. The commit history from releases 2.2.9 to 2.2.3 was considered for the evolutionary training set. Commits with more than ten files were discarded. This type of filtering is a common heuristic used in mining techniques to mitigate factors such as updating the license information on every file or performing merging and copying [24, 46]. Furthermore, because commits may contain non source code files, only source code files were considered and other types discarded.

The tool *sqminer* was employed to mine evolutionary couplings (and association rules) in the *itemset mining* mode with minimum support values of 1, 2, 4, and 8. Also, we considered all the possible association rules with the confidence values greater than zero. Mining was performed at both file and method levels of granularity. The mining time was in the order of a few seconds.

3) Testing set

The testing sets were extracted similar to training sets; however, the periods of history used were different from the training set. The testing set consists of commits extracted from a period of history after the release date of the version of the system used to build the corpus. For example, the commit history of *Apache httpd* after the release 2.2.3 and up to the release 2.2.5 was considered for the testing set. The testing set provides a way to evaluate our proposed approach. Similar approaches for the training and testing sets are previously reported in the literature, for example in [24, 46].

Table III shows the details of the evolutionary training and testing sets considered at the file and method levels. The entries corresponding to the method level are suffixes with a * symbol (same notation in other tables). They include a range of releases corresponding to different history periods. Also, the numbers of commits and files (methods) during those periods of history are provided. The (larger) training sets and (smaller) testing sets were extracted from the *History* (Table III) periods before and after the *Version*s (Table II) used to index with LSI. For the method level, the number of commits corresponds to commits that contained method changes (and so differs from those at the file level).

Table IV. Orthogonality check for various cut points of conceptual (C), evolutionary (E), and their combination. The results show that conceptual and evolutionary couplings provide orthogonal information, and support a strong case for combining them.

		5	10	20	30	40	50			5	10	20	30	40	50
Apache	C\E	32	33	35	35	35	37	KOffice 2.0.91	C\E	60	62	64	64	63	64
	E\C	39	36	28	23	20	17		E\C	26	22	16	13	12	11
	C∩E	29	32	37	42	45	46		C∩E	14	16	20	23	24	26
ArgoUML	C\E	59	51	44	41	41	40	KOffice 2.0.1	C\E	42	41	40	42	43	44
	E\C	28	26	28	25	24	22		E\C	43	38	36	35	33	32
	C∩E	13	23	29	34	35	38		C∩E	16	21	23	23	23	24
iBatis	C\E	67	65	69	70	70	70	KOffice 2.0.1*	C\E	47	48	46	47	46	46
	E\C	15	21	14	14	13	12		E\C	52	51	52	50	51	51
	C∩E	18	13	16	16	17	18		C∩E	1	1	2	3	3	3

D. Results

*1) **RQ1**:Does combining conceptual and evolutionary couplings improve accuracy of IA?*

Prior research efforts have investigated the performance of coupling metrics that use specific sources of information (*e.g.*, structural and textual) to capture couplings in source code. Our first research question focuses on determining if we can improve the accuracy of IA by augmenting metrics based on complementary underlying information.

As a step toward determining the potential benefits of combining conceptual and evolutionary couplings, we analyze the orthogonality of the two standalone couplings. One situation where the combination of the techniques is beneficial is when techniques provide complementary sets of correct entities. If the standalone techniques considered for combination provide identical or very similar information, combining them may not be a worthwhile effort. In order to measure the degree to which the techniques could potentially complement one another we use the following metrics:

$$correct_{m_i \cap m_j} = \frac{|correct_{m_i} \cap correct_{m_j}|}{|correct_{m_i} \cup correct_{m_j}|}\% \quad correct_{m_i \setminus m_j} = \frac{|correct_{m_i} \setminus correct_{m_j}|}{|correct_{m_i} \cup correct_{m_j}|}\%$$

where $correct_{mi}$ represents the set of source code entities correctly identified when using coupling metric m_i for IA. The two metrics capture the overlap between the set of correct source code entities and the percentage of correct entities identified only by m_i respectively.

The results of orthogonality metrics between the two metrics for the various systems are given in Table IV. Based on our datasets, the overlap between the set of correct links for the two approaches did not exceed 46%. Minimal overlap indicates potential orthogonality between the two techniques. One exception is the case where virtually all correct entities identified by one technique make up a small subset of the correct entities identified by the other technique. A similar scenario is where one technique performs inadequately and returns very few correct entities. Both cases are captured by our metric $correct_{mi\setminus mj}$. Our results contain cases where conceptual couplings are capable of identifying a large portion of correct entities not identified by evolutionary couplings, and vice versa. In case of *KOffice 2.0.1* both techniques are capable of capturing a similar portion of correct entities. These findings support our premise that combining conceptual and evolutionary couplings could identify a larger set of correct entities.

Based on our datasets, conceptual and semantic couplings identify correct entities orthogonally. With this knowledge we direct our attention to our second step towards demonstrating the benefits of combining the couplings.

Table V. Precision (P) and recall (R) percentages results of conceptual coupling (Conc), evolutionary coupling (Evol), disjunctive (Disj), and conjunctive (Conj) approaches to impact analysis for all systems using various cut points. ImpC and ImpE show the improvement obtained by the disjunctive approach compared to conceptual and evolutionary couplings respectively. The disjunctive approach outperforms with statistical significance.

		5		10		20		30		40		50			5		10		20		30		40		50	
		P	R	P	R	P	R	P	R	P	R	P	R		P	R	P	R	P	R	P	R	P	R	P	R
Conc	Apache	15	28	11	38	7	49	6	58	5	63	4	67	KOffice 2.0.91	13	27	9	35	6	46	5	53	4	56	3	59
Evol		18	38	11	43	6	48	4	51	3	53	3	54		9	19	6	22	4	24	3	26	2	28	2	28
Disj		21	43	14	54	9	64	6	70	5	73	4	78		17	34	12	44	8	55	6	60	5	63	4	65
Conj		16	34	10	40	6	47	4	47	3	48	2	48		8	16	5	21	3	22	2	22	2	23	1	23
ImpC		6	15	3	16	2	15	0	12	0	10	0	11		4	7	3	9	2	9	1	7	1	7	1	6
ImpE		3	5	3	11	3	16	2	19	2	20	1	24		8	15	6	22	4	31	3	34	3	35	2	37
Conc	ArgoUML	11	17	8	22	5	27	4	32	4	35	3	38	KOffice 2.0.1	10	19	7	26	4	30	3	33	3	35	2	37
Evol		6	10	5	15	4	20	3	24	3	27	2	29		13	26	9	30	5	35	4	36	3	37	2	37
Disj		11	19	9	27	6	33	5	38	4	41	4	44		16	34	11	41	7	49	5	53	4	55	4	57
Conj		10	16	7	18	4	21	3	25	3	25	2	25		8	15	5	17	3	18	2	18	2	18	1	18
ImpC		0	2	1	5	1	6	1	6	0	6	1	6		6	15	4	15	3	19	2	20	1	20	2	20
ImpE		5	9	4	12	2	13	2	14	1	14	2	15		3	8	2	11	2	14	1	17	1	18	2	20
Conc	iBatis	17	27	13	37	10	56	7	59	6	61	5	63	KOffice 2.0.1*	5	4	3	4	2	5	2	6	1	7	1	7
Evol		7	11	6	17	3	19	2	21	2	24	2	24		11	7	10	12	7	17	6	17	5	18	4	19
Disj		18	27	14	40	10	60	8	66	6	68	5	68		14	10	12	15	8	21	6	22	5	24	4	25
Conj		8	12	5	13	3	15	2	15	1	15	1	15		2	1	1	1	1	1	1	1	1	1	1	1
ImpC		1	0	1	3	0	4	1	7	0	7	0	5		9	6	9	11	6	16	4	16	4	17	3	18
ImpE		11	16	8	23	7	41	6	45	4	44	3	44		3	3	2	3	1	4	0	5	0	6	0	6

Table VI. Results of Wilcoxon signed-rank test ($\mu = 30$). The p values indicate that the disjunctive approach provided improvement is not by chance.

System	$H_{0\,CP}$	$H_{0\,CR}$	$H_{0\,EP}$	$H_{0\,ER}$	Null Hypothesis
Apache(httpd)	0.0002	0.0003	0.0001	0.0003	Rejected
ArgoUML	0.0050	0.0039	< 0.0001	< 0.0001	Rejected
iBatis	0.0126	0.0126	0.0001	0.0002	Rejected
KOffice 2.0.91	< 0.0001	< 0.0001	< 0.0001	< 0.0001	Rejected
KOffice 2.0.1	< 0.0001	< 0.0001	< 0.0001	< 0.0001	Rejected
KOffice 2.0.1*	< 0.0001	< 0.0001	< 0.0001	< 0.0001	Rejected

Table V provides precision and recall results for the subject systems under study. These results are obtained by using the various couplings for IA. Only a subset of the cut points (μ) we considered are shown in Table V. The cut points represent the sizes of the impact set considered with our combinations. For example, a cut point of 5 indicate that the estimated impact set with our approach contained 5 entities.

We considered both disjunctive and conjunctive approaches to combining couplings. The disjunctive approach outperforms the conjunctive approach in all cases considered (see Table V). Additionally, the conjunctive approach is generally unable to provide improvement over either technique. This is somewhat expected because the two couplings appear complementary (see Table IV). The orthogonality between the sets of correct entities identified by the two couplings appears to contribute to the performance of the conjunctive approach. The utility of the conjunctive approach is probably better suited for scenarios where a pair of couplings identifies similar sets of correct entities, but varying sets of false positives. For such a scenario the conjunctive approach may serve as a useful filtering mechanism for false positives. The disjunctive approach better leverages the orthogonality between the couplings. The rest of the discussion about the combinations of two couplings refers to the disjunctive approach.

The prevailing pattern of our results demonstrates that the combination of conceptual and evolutionary couplings improves the performance over either standalone technique. Consider a case in Table V where $\mu = 30$ for *Apache httpd*. Conceptual and evolutionary couplings in this instance yield recall values of 58% and 51% respectively, while the combination of the two increases recall to 70%. Similar improvements are apparent throughout all the datasets considered in our evaluation. Another example is where $\mu = 50$ for *KOffice 2.0.1* (file-level granularity). In this case both conceptual and evolutionary couplings result in recall of 37% while their combination gives recall of 57%. Within our results a few cases surface that illustrate the importance of both techniques. For example, in the case where $\mu = 5$ for *iBatis* combining conceptual and evolutionary couplings does not improve accuracy. This can be partially attributed to the accuracy of the evolutionary coupling metric. In this case, the inadequate individual performance of a technique limits the gain acquired when they are combined.

Our results for combining conceptual and evolutionary couplings are promising. To further ascertain our conclusions on our initial dataset, we carried out a statistical test. We developed four testable null hypotheses:

$\mathbf{H}_{0\,CP}$: Combining conceptual and evolutionary couplings *does not significantly* improve precision results of impact analysis compared to conceptual couplings.

$\mathbf{H}_{0\,CR}$: Combining conceptual and evolutionary couplings *does not significantly* improve recall results of impact analysis compared to conceptual couplings.

$\mathbf{H}_{0\,EP}$: Combining conceptual and evolutionary couplings *does not significantly* improve precision results of impact analysis compared to evolutionary couplings.

$\mathbf{H}_{0\,ER}$: Combining conceptual and evolutionary couplings *does not significantly* improve recall results of impact analysis compared to evolutionary couplings.

We also developed alternative hypotheses for the cases where the null hypotheses can be rejected with relatively high confidence. For example:

$\mathbf{H}_{a\,CP}$: Combining conceptual and evolutionary couplings *significantly* improve precision results of impact analysis compared to conceptual couplings.

The remaining three alternative hypotheses are formulated in a similar manner and are left out for brevity.

To test for statistical significance we used the Wilcoxon signed-rank test, a non-parametric paired samples test. Our application of the test determines whether the improvement obtained using the combination of conceptual and evolutionary couplings compared to standalone approaches is statistically significant.

Table VI presents the results of performing the Wilcoxon signed-rank test. We performed the test for each of the four hypotheses for each system to determine whether the improvements for precision and recall when combining the techniques are statistically significant over the accuracy of standalone conceptual and evolutionary couplings. In all cases considered for our dataset we obtained a p-value less than 0.05, indicating that the improvement in accuracy obtained is not by chance.

*2) **RQ2**: Does the choice of granularity (i.e., file vs. method) impact standalone techniques and their combinations?*

Our second research question focuses on the impact of granularity on the accuracy of the standalone techniques, as well as their combinations. We examined the impact of different granularities on the accuracy of the couplings when they are used for IA. Here, we focused on the accuracy of the various couplings on the system *KOffice 2.0.1*. For this system we obtained results at both file and method levels of granularity. Accuracy results of the techniques for IA are shown in Table V. There is a noticeable decrease in accuracy when method level granularity is used. Conceptual coupling is affected by the difference in granularity more than evolutionary coupling. Regardless of the decrease in accuracy of the standalone techniques, when the two are

combined there exists a statistically significant improvement in accuracy. In certain cases the improvement achieved is 6%. Generally, only a small portion of correct methods identified by both techniques overlap, *i.e.*, they exhibit orthogonality. This allows their combination to provide an enriched set of correct methods.

Our results show that the level of granularity does impact the accuracy of both standalone techniques and their combinations. Although finer granularity decreases accuracy of all approaches, it does not prevent the combination of the two from outperforming the standalone techniques. That is, the gain acquired by combining conceptual and evolutionary coupling exists regardless of the granularity considered in this study. For both file-level and method-level granularity levels, combining conceptual and evolutionary information delivers accuracy superior to either standalone technique.

E. Threats to validity

We address some of the threats to validity that could have impacted our empirical study and results. The uses of LSI and itemset mining algorithms are sensitive to a set of user-defined parameters. It is a viable risk that the improvements gained by our approach are valid only for a particular set of these parameter values. To address this risk, we experimented with different parameter values. For example, the accuracy of evolutionary couplings decreases with an increase in the minimum support value; however, the trend of accuracy gains continued with our approach. We will continue our quest to obtain the optimal values with other studies in the future.

We measured the accuracy of IA with precision and recall metrics. It is possible that a different accuracy metric may produce a different result; however, both these metrics are widely used and accepted in the community, including for IA. We tried with F-measure, which is based on precision and recall, and also noticed statistically significant improvements with our disjunctive approach. We considered (later) commits as the gold standard for computing our accuracy metrics. It is reasonable to assume that not all the entities in a commit are related to a single change request, and a single commit may not capture all the entities related to a change request. Therefore, they may not be an accurate representation of the actual change-sets and could have compromised our accuracy basis. However, commits have been used as a basis for accuracy assessment previously (*e.g.*, see Zimmerman et al. [46]). We did some manual inspection and plan to conduct a user study with developer established actual impact sets in the future. We reported our findings at the granularity of file and method levels. A possible issue here could be how well our results hold for other granularity levels besides the two considered. We concur with previous studies [46] that file and method granularity levels provide a realistic balance of coarse and fine granularity levels for IA. The accuracies of the two standalone techniques, however low in certain cases to raise a practicality concern, are comparable to other previous results [34, 46]. Our work shows how to improve accuracy by forming effective combinations.

We evaluated on datasets from four open source systems that represent a wide spectrum of domains, programming languages (C/C++ and Java), sizes, and development processes. However, we do not claim that our combined approach would operate with equivalent improvement in accuracy on other systems, including closed source.

V. CONCLUSIONS AND FUTURE WORK

The empirical assessment on four open source systems provides support for our approach with several conclusions in the context of change impact analysis. Combining conceptual and evolutionary couplings improves accuracy. Our findings indicate that in certain cases an improvement of 20% in recall is achieved when conceptual and evolutionary coupling is combined. The overall improvement obtained when combining the two techniques is statistically significant for the dataset used in our evaluation. Although our combining methods of couplings may appear straightforward, it did provide promising improvements in accuracy. Our findings show that the disjunctive approach clearly outperforms the conjunctive approach in accuracy. We conjecture that the difference in performance is, in part, an attribute of the orthogonal nature of the correct entities revealed by the two couplings in our empirical analysis.

Varying granularity levels does impact accuracy; however, combining conceptual and evolutionary couplings maintains the accuracy gains. Based on our datasets, using finer granularity (*i.e.*, method-level) decreases the accuracy of all techniques considered. One important point to note is that, regardless of the decrease in individual accuracy, the combination of conceptual and evolutionary coupling consistently outperformed both standalone techniques. Thus, there is strong evidence showing the benefits of the combination of conceptual and evolutionary couplings at various levels of granularity.

We plan to devise and empirically validate other combinations of conceptual and evolutionary couplings (e.g., weighed contributions of entities from each coupling based on the amount of change history considered). Another key future direction includes the addition of static and dynamic analysis information, and application of IR on multi-version artifacts (e.g., commit messages and bug reports) in our approach, and extending our approach to provide IA support beginning from a high-level textual change request. We are also planning extensive comparative studies with other approaches (e.g., structural metrics). In a previous study [3, 34], it was reported that IR techniques performed as well as or better than those based on structural metrics for IA. This work will serve as a guideline for our future studies.

VI. ACKNOWLEDGEMENTS

This work is supported by NSF CCF-1016868 and NSF CCF-1016887 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

VII. REFERENCES

[1] Agrawal, R. and Srikant, R., "Mining Sequential Patterns", in Proc. of 11th ICDE, Taipei, Taiwan, March 1995.

[2] Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E., "Recovering Traceability Links between Code and Documentation", *IEEE TSE*, vol. 28, no. 10, October 2002, pp. 970 - 983.

[3] Antoniol, G., Canfora, G., Casazza, G., and Lucia, A., "Identifying the Starting Impact Set of a Maintenance Request: A Case Study", in Proc. of 4th CSMR'00, Zurich, Switzerland, pp. 227-231.

[4] Antoniol, G., Gueheneuc, Y.-G., Merlo, E., and Tonella, P., "Mining the Lexicon Used by Programmers during Software Evolution", in Proc. of 23rd ICSM'07, Paris, pp. 14-23.

[5] Anvik, J., Hiew, L., and Murphy, G. C., "Who should fix this bug?", in Proc. of 28th ICSE'06, pp. 361-370.

[6] Basili, V. R., Caldiera, G., and Rombach., D. H., *The Goal Question Metric Paradigm*, John W & S, 1994.

[7] Binkley, D. and Lawrie, D., "Information Retrieval Applications in Software Maintenance and Evolution", in *Encyclopedia of Software Engineering*: Taylor & Francis LLC, 2009.

[8] Bohner, S. and Arnold, R., *Software Change Impact Analysis*, Los Alamitos, CA, IEEE Computer Society, 1996.

[9] Briand, L., Labiche, Y., and Soccar, G., "Automating Impact Analysis and Regression Test Selection Based on UML Designs", in Proc. of ICSM'02, Montreal, pp. 252-261.

[10] Briand, L., Wust, J., and Louinis, H., "Using Coupling Measurement for Impact Analysis in Object-Oriented Systems", in Proc. of ICSM'99, pp. 475-482.

[11] Canfora, G. and Cerulo, L., "Impact Analysis by Mining Software and Change Request Repositories", 11th METRICS'05, pp. 20-29.

[12] Caprile, C. and Tonella, P., "Nomen Est Omen: Analyzing the Language of Function Identifiers", in Proc. of 6th WCRE'99, Atlanta, Georgia, USA, October 6-8, pp. 112-122.

[13] Chen, K. and Rajlich, V., "Case Study of Feature Location Using Dependence Graph", 8th IWPC'00, pp. 241-249.

[14] Chen, K. and Rajlich, V., "RIPPLES: Tool for Change in Legacy Software", in Proc. of ICSM'01, pp. 230-239.

[15] Collard, M. L., Kagdi, H. H., and Maletic, J. I., "An XML-Based Lightweight C++ Fact Extractor", in Proc. of 11th IWPC'03, Portland, OR, May 10-11 2003, pp. 134-143.

[16] De Lucia, A., Fasano, F., Oliveto, R., and Tortora, G., "Recovering Traceability Links in Software Artefact Management Systems", *ACM TOSEM*, vol. 16, no. 4, 2007.

[17] Deissenboeck, F. and Pizka, M., "Concise and Consistent Naming", in Proc. of 13th IWPC'05, 2005, pp. 97-106.

[18] Di Lucca, G. A., Di Penta, M., and Gradara, S., "An Approach to Classify Software Maintenance Requests", in Proc. of IEEE ICSM'02, Montréal, Québec, Canada, 2002, pp. 93-102.

[19] Gallagher, K. and Lyle, J., "Using Program Slicing in Software Maintenance", *IEEE TSE*, vol. 17, no. 8, August 1991 1991, pp. 751-762.

[20] Hill, E., Pollock, L., and Vijay-Shanker, K., "Exploring the Neighborhood with Dora to Expedite Software Maintenance", in Proc. of 22nd ASE'07, November 2007, pp. 14-23.

[21] Jeong, G., Kim, S., and Zimmermann, T., "Improving Bug Triage with Bug Tossing Graphs", in Proc. of ESEC/FSE'09.

[22] Kagdi, H., Collard, M. L., and Maletic, J. I., "A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution", *JSME*, vol. 19, no. 2, March/April 2007, pp. 77-131.

[23] Kagdi, H. and Maletic, J. I., "Mining Evolutionary Dependencies from Web-Localization Repositories", *JSME*, vol. 19, no. 5, 2007, pp. 315-337.

[24] Kagdi, H., Maletic, J. I., and Sharif, B., "Mining Software Repositories for Traceability Links", in Proc. of IEEE ICPC'07, Banff, Canada, June 26-29 2007, pp. 145-154.

[25] Kagdi, H. and Poshyvanyk, D., "Who Can Help Me with this Change Request?", in Proc. of 17th IEEE ICPC'09, Vancouver, British Columbia, Canada, 2009, pp. 273-277.

[26] Kagdi, H., Yusuf, S., and Maletic, J. I., "Mining Sequences of Changed-files from Version Histories", in Proc. of 3rd MSR'06 Shanghai, China, May 22-23 2006, pp. 47-53.

[27] Kosara, R., Healey, C. G., Interrante, V., Laidlaw, D. H., and Ware, C., "Visualization viewpoints", *Computer Graphics and Applications*, vol. 23, no. 4, July-August 2003, pp. 20-25.

[28] Law, J. and Rothermel, G., "Whole Program Path-Based Dynamic Impact Analysis", in Proc. of 25th ICSE, Portland, Oregon, May 03 - 10, 2003 2003, pp. 308-318.

[29] Moonen, L., "Lightweight Impact Analysis using Island Grammars", in Proc. of 10th IWPC'02, Paris, France, June 27 - 29, 2002 2002, pp. 219-228.

[30] Orso, A., Apiwattanapong, T., Law, J., Rothermel, G., and Harrold, M. J., "An empirical comparison of dynamic impact analysis algorithms", in Proc. of ICSE'04, pp. 776-786.

[31] Petrenko, M. and Rajlich, V., "Variable Granularity for Improving Precision of Impact Analysis", in Proc. of 17th ICPC'09, Vancouver, BC, Canada, 2009, pp. 10-19

[32] Poshyvanyk, D., Guéhéneuc, Y. G., Marcus, A., Antoniol, G., and Rajlich, V., "Feature Location using Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval", *IEEE TSE*, vol. 33, no. 6, June 2007, pp. 420-432.

[33] Poshyvanyk, D. and Marcus, A., "The Conceptual Coupling Metrics for Object-Oriented Systems", in Proc. of 22nd ICSM'06, Philadelphia, PA, Sept. 25-27 2006, pp. 469 - 478.

[34] Poshyvanyk, D., Marcus, A., Ferenc, R., and Gyimóthy, T., "Using Information Retrieval based Coupling Measures for Impact Analysis", *ESE*, vol. 14, no. 1, 2009, pp. 5-32.

[35] Poshyvanyk, D. and Marcus, D., "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code", in Proc. of 15th IEEE ICPC'07, Banff, Alberta, Canada, June 2007, pp. 37-48.

[36] Queille, J.-P., Voidrot, J.-F., Wilde, N., Munro, M., and "The Impact Analysis Task in Software Maintenance: A Model and a Case Study", in Proc. of ICSM'94, pp. 234 - 242.

[37] Ren, X., Shah, F., Tip, F., Ryder, B. G., and Chesley, O., "Chianti: a Tool for Change Impact Analysis of Java Programs", in Proc. of 19th ACM OOPSLA '04, Vancouver, BC, Canada, pp. 432-448.

[38] Robillard, M., "Automatic Generation of Suggestions for Program Investigation", in Proc. of ESEC/FSE'05, pp. 11 - 20

[39] Rountev, A., Milanova, A., and Ryder, B., G., "Points-to analysis for Java using annotated constraints", in Proc. of ACM OOPSLA'01, Tampa Bay, FL, USA, 2001, pp. 43-55.

[40] Tonella, P., "Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis", *IEEE TSE*, vol. 29, no. 6, June 2003 2003, pp. 495-509.

[41] Vaclav, R., "A Model for Change Propagation Based on Graph Rewriting", in Proc. of ICSM '97, Bari, Italy, 1997, pp. 84-91.

[42] Wang, X., Zhang, L., Xie, T., Anvik, J., and Sun, J., "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information", in Proc. of 30th ICSE'08, Leipzig, Germany, May 10 - 18 2008, pp. 461-470.

[43] Weiss, C., Premraj, R., Zimmermann, T., and Zeller, A., "How Long Will It Take to Fix This Bug?", in Proc. of 4th IEEE MSR'07, Minneapolis, MN, 2007, pp. 1-8.

[44] Wilkie, F. G. and Kitchenham, B. A., "Coupling measures and change ripples in C++ application software", *The Journal of Systems and Software*, vol. 52, 2000, pp. 157-164.

[45] Yu, Z. and Rajlich, V., "Hidden Dependencies in Program Comprehension and Change Propagation", in Proc. of 9th IEEE IWPC'01, Toronto, Canada, May 12 -13 2001, pp. 293-299.

[46] Zimmermann, T., Zeller, A., Weißgerber, P., and Diehl, S., "Mining Version Histories to Guide Software Changes", *IEEE TSE*, vol. 31, no. 6, 2005, pp. 429-445.