Integrating Conceptual and Logical Couplings for Change Impact Analysis in Software

Huzefa Kagdi[1], Malcom Gethers[2*], and Denys Poshyvanyk[3]

[1]*Wichita State University*

[2]*University of Maryland Baltimore County*

[3]*The College of William and Mary*

*Malcom Gethers completed this work during his Ph.D. at College of William and Mary

kagdi@cs.wichita.edu, mgethers@umbc.edu, denys@cs.wm.edu

http://www.cs.wm.edu/semeru/
http://serl.cs.wichita.edu/

Abstract The paper presents an approach that combines conceptual and evolutionary techniques to support change impact analysis 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). 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 premise is that such combined methods provide improvements to the accuracy of impact sets compared to the two individual approaches. A rigorous empirical assessment on the changes of the open source systems Apache httpd, ArgoUML, iBatis, KOffice, and jEdit is also reported. The impact sets are evaluated at the file and method levels of granularity for all the software systems considered in the empirical evaluation. The results show that a combination of conceptual and evolutionary techniques, across several cut-off points and periods of history, provides statistically significant improvements in accuracy over either of the two techniques used independently. Improvements in F-measure values of up to 14% (from 3% to 17%) over the conceptual technique in ArgoUML at the method granularity, and up to 21% over the evolutionary technique in iBatis (from 9% to 30%) at the file granularity were reported.

Keywords *Change impact analysis, Information Retrieval, conceptual and logical coupling, mining software repositories, open-source software, software evolution and maintenance*

1 Introduction

Software maintenance and evolution is a particularly complex phenomenon in case of long-lived, large-scale systems (Lehman and Belady 1985; Rajlich and Bennett 2000). It is not uncommon for such systems to progress through years of development history, a number of developers, and a multitude of software artifacts including millions of lines of code. Therefore, realizing even a slight change may not be always straightforward. According to Bohner and Arnold (Bohner and Arnold 1996), 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. IA is a key task in software maintenance and evolution. 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, source code, and/or test cases.

Decades of research efforts have produced a wide spectrum of approaches, ranging from the traditional static and dynamic analysis techniques (Briand et al. 1999; Law and Rothermel 2003; Orso et al. 2004; Ren et al. 2004; Robillard 2005; Petrenko and Rajlich 2009) to the contemporary methods such as those based on Information Retrieval (IR) (Canfora and Cerulo 2005; Hill et al. 2007; Poshyvanyk et al. 2009; Gethers and Poshyvanyk 2010) and Mining Software Repositories (MSR) (Gall 1998; Ying et al. 2004; Fluri et al. 2005; Zimmermann et al. 2005; Canfora et al. 2010; Kagdi et al. 2010). 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 utilizing some of the existing solutions. Central to our approach are the information sources that are developer/human centric (*e.g.*, comments and identifiers, and commit practices), rather than (formal) language/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 (Antoniol et al. 2000; Poshyvanyk et al. 2009). 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 primary research hypothesis is that such combined methods provide improvements to the accuracy of impact sets.

An extensive empirical study on hundreds of changes from open source systems, such as *Apache httpd*, *ArgoUML*, *iBatis*, *KOffice,* and *jEdit*, 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-off 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 F-measure values of up to 14% (from 3% to 17%) over the conceptual technique in *ArgoUML* at the method granularity, and up to 21% over the evolutionary technique in *iBatis* (from 9% to 30%) at the file granularity. Also, we found that using larger history periods for computing evolutionary couplings improves impact analysis results for the combined technique. These results are encouraging considering that the combinations do not require an overly complex blending of two standalone approaches.

We significantly extends our previous work (Kagdi et al. 2010). In particular, we present detailed analysis results at the method level granularity for all the studies software systems: *Apache httpd*, *ArgoUML*, *iBatis*, and *KOffice*. These results were not available in (Kagdi et al. 2010). We added and analyzed data from another software system (*jEdit*) for the file and method granularity levels. Also, we extended the statistical tests to all the systems for both file and method levels of granularity. Finally, we investigated an additional research question (RQ3) in our empirical evaluation that studies the impact of history on the accuracy of our approach on *Apache httpd*, *ArgoUML*, *iBatis*, and *KOffice* software systems.

The rest of the paper is organized as follows. Section 2 provides a brief discussion of the related work, whereas section 3 presents our combined approach. The empirical assessment is presented in Section 4. We conclude in Section 5.

2 Background and Related Work

The paper addresses 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 these problems.

2.1 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 (Gallagher and Lyle 1991), hidden dependency analysis (Rajlich 1997; Chen and Rajlich 2001; Yu and Rajlich 2001), lightweight static analysis approaches (Moonen 2002; Petrenko and Rajlich 2009), concept analysis (Tonella 2003), dynamic analysis (Law and Rothermel 2003; Orso et al. 2004; Ren et al. 2004), hypertext systems, documentation systems, UML models (Briand et al. 2002), and Information Retrieval (Antoniol et al. 2000; Poshyvanyk et al. 2009) are already investigated in the literature. Queille et al. (Queille et al. 1994) 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 (Chen and Rajlich 2000).

Coupling measures have been also used to support impact analysis in Object Oriented systems (Briand et al. 1999; Wilkie and Kitchenham 2000; Poshyvanyk et al. 2009; Gethers and Poshyvanyk 2010). Wilkie and Kitchenham (Wilkie and Kitchenham 2000) 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. (Briand et al. 1999) 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 (Robillard 2005; Hill et al. 2007; Saul et al. 2007; Robillard 2008), where proposed tools could help navigate and prioritize system dependencies during various software maintenance tasks. The work in (Hill et al. 2007) relates to our approach only to the extent that 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 (Rountev et al. 2001; Kosara et al. 2003). A comparison of different impact analysis algorithms is provided in (Orso et al. 2004).

2.2 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 (Deissenboeck and Pizka 2005; Deissenboeck and Pizka 2006). These names often serve as a starting point in many program comprehension tasks (Caprile and Tonella 1999; Haiduc and Marcus 2008; Abebe et al. 2009; Arnaoudova et al. 2010). 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 (Antoniol et al. 2007; Binkley et al. 2009).

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 (Liu et al. 2007; Poshyvanyk et al. 2007; Poshyvanyk and Marcus 2007; Eaddy et al. 2008; Revelle and Poshyvanyk 2009; Revelle et al. 2010; Dit et al. 2012a; Dit et al. 2012b; Poshyvanyk et al. 2012), traceability link recovery (Antoniol et al. 2002; Hayes et al. 2006; De Lucia et al. 2007; Cleland-Huang et al. 2010; Oliveto et al. 2010; Gethers et al. 2011), and impact analysis (Antoniol et al. 2000; Canfora and Cerulo 2005; Poshyvanyk et al. 2009; Gethers and Poshyvanyk 2010;

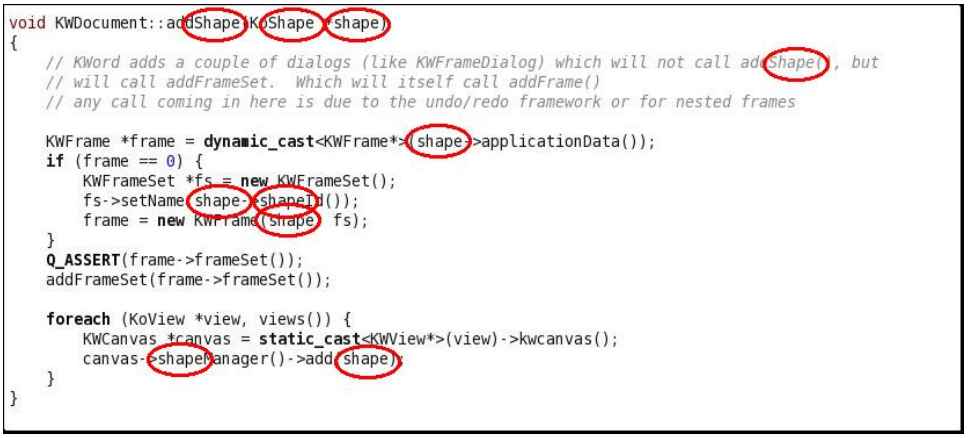


Figure 1 A method named *addShape()* from *KOffice* showing the conceptual information that is latent in (some of the) identifier names.

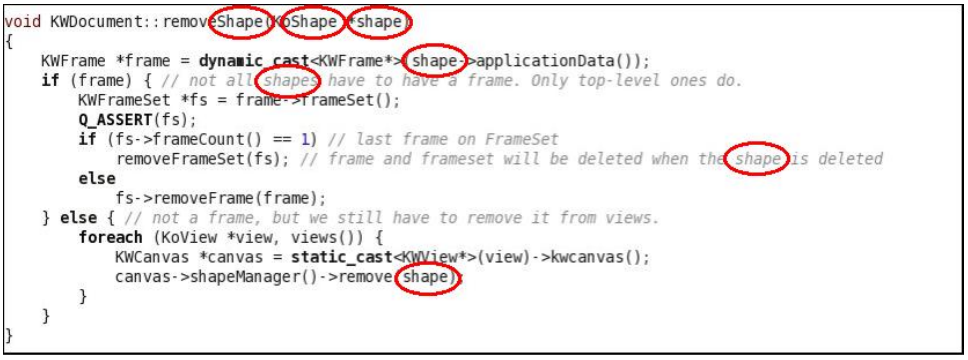


Figure 2 A method named *removeShape()* from *KOffice* showing the conceptual information that is latent in (some of the) identifier names.

Gethers et al. 2012). 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 (Binkley and Lawrie 2010a; Binkley and Lawrie 2010b) for such an overview.

2.3 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. (Zimmermann et al. 2005) used *CVS* logs for detecting evolutionary coupling (or logical couplings as defined by Gall et al.(Gall 2003)) among source code entities. Association rules based on *itemset* mining were formed from the change-sets and used for change-prediction. Canfora et al. (Canfora and Cerulo 2005) 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 (Anvik et al. 2006; Jeong et al. 2009; Kagdi and Poshyvanyk 2009; Kagdi et al. 2011; Kagdi et al. 2012), identifying duplicate bug reports (Runeson et al. 2007; Wang et al. 2008), estimating time to fix incoming bugs (Weiss et al. 2007) and classifying software maintenance requests (Di Lucca et al. 2002). Finally, we conducted a comprehensive literature survey on MSR approaches during the

prologue of this work (Kagdi et al. 2007). Xie's online bibliography and tutorial[1] on MSR is another well-maintained web resource.

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 conceptual couplings) of source code and data mining techniques are used on past commits of source code versions (*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 integrated technique.

3 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 (Zimmermann et al. 2005; Hill et al. 2007; Petrenko and Rajlich 2009; Poshyvanyk et al. 2009; Kagdi et al. 2010). 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 version 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 source code entity from Step 1.

Step 3: **Mine** commits from the source code repository and compute evolutionary couplings. Here, only the commits that occurred before the version 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.

3.1 Conceptual Couplings

We use *conceptual similarity* as a primary mechanism for 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 or simply LSI):

$$CSE(e_k, e_j) = \frac{ve_k^T ve_j}{|ve_k|_2 \times |ve_j|_2} \qquad \text{Eq. I}$$

The value of $CSE(e_k, e_j) \in [-1, 1]$, as *CSE* is a cosine in the Vector Space Model (VSM). In order to comply with non-negativity property of coupling metrics (Briand et al. 1999), we redefine *CSE* as:

$$CSE(e_k, e_j) = \begin{cases} CSE(e_k, e_j) & if \quad CSE(e_k, e_j) \geq 0 \\ else \quad 0 \end{cases} \qquad \text{Eq. II}$$

For source code documents, the entities can be attributes, methods, classes, files, etc. In 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 given in Eq. I), while deriving method-class or class-class *CSE* (given in Eq. II) 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:

$$CSEMC(m_k, c_j) = \sum_{q=1}^{t} CSE(m_k, m_{jq}) / t, \qquad \text{Eq. III}$$

which is an average of the conceptual similarities between method m_k and all the methods from class c_j. Using *CSEMC* (given in Eq. III) 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:

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

Table I Excerpt of conceptual coupling values for the methods *addShape()* and *removeShape()* in *KOffice* software system.

	KWDocument::addShape	**KWDocument::removeShape**	**KWDocument::InsertPage**
KWDocument:: addShape	1	**0.78**	0.24
KWDocument:: removeShape	**0.78**	1	0.27
KWDocument:: InsertPage	0.24	0.27	1

$$CSEBC(c_k, c_j) = \sum_{l=1}^{r} CSEMC(m_{kl}, c_j)/r, \qquad \text{Eq. IV}$$

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 (Poshyvanyk and Marcus 2006; Poshyvanyk et al. 2009). Please note that conceptual couplings at class level granularity are computed using $CSE(e_k, e_j)$, where e_k and e_j are files implementing respective classes, however we present all the formulas for completeness purposes if a method level granularity is desired for computing conceptual couplings.

To analyze conceptual information in a given version 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* (Collard et al. 2003) 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.

Figure 1 and Figure 2 show the implementation of two methods named *addShape* and *removeShape()* from *KOffice*. To derive the conceptual similarity between these methods, we extract the identifier names and terms in comments from methods (*i.e.*, corpus creation). All terms in the method, with the exception of stop words (common English words and programming language keywords) are included in the corpus. The corpus is indexed using LSI and its real-valued vector subspace representation is created. The vector space is represented with a term-document matrix. Dimensionality reduction is performed in this step, capturing the important semantic information about identifiers and comments in the source code, and their relationships. In the resulting subspace, each document (method) has a corresponding vector. Similarities between each pair of documents (*i.e.*, methods) in the source code are computed using the cosine measure. The first two rows in Table I show the conceptual coupling values for the method pair *addShape* and *removeShape()*. The conceptual coupling value is between 0 and 1. The conceptual coupling between *addShape* and *removeShape()* is 0.78. Notice that it is a symmetric metric, *i.e.*, the values of (*addShape*, *removeShape()*) and (*removeShape(),addShape*) are the same. The terms such as *canvas*, *frameset*, and *shape* contribute to the conceptual similarity between these methods. For example, the occurrences of the term *shape* are highlighted in red in Figure 1 and Figure 2 to provide a visual emphasis of its contribution. For more details and examples, please refer to our prior work on conceptual coupling measures (Poshyvanyk and Marcus 2006; Poshyvanyk et al. 2009).

3.2 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 use *itemset* mining (Agrawal and Srikant 1995), 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 $\bigcup_{i=1}^{m} csi$ be the set of *m* entities, each of which was changed in at least one change-set during the time interval λ. 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, $ec_u = \{e_p, e_q, ..., e_o\}$ 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 initial entity 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 rule is of the form $lhs \Rightarrow rhs$. Multiple rules are possible for the same *lhs* entity (and also the *rhs* entity). 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*).

Figure 3 shows a log entry from the *Subversion* repository of *kdelibs* (a part of *KDE* repository.) A log entry corresponds to a single *commit* operation. This information can be readily obtained in an XML format by using the command–line client *svn log*. *Subversion*'s log entries include the dimensions *author*, *date*, and *paths* involved in a change-set. In this case, the changes in the files *khtml_part.cpp* and *loader.h* are committed together by the developer *kling* on the date/time *2005-07-25T17:46:20.434104Z*. The *revision* number *438663* is assigned to the entire change-set (and not to each file that is changed as is in the case with some version-control systems such as CVS). Additionally, a text message describing the change entered by the developer is also recorded. Note that the order in which the files appear in the log entry is not necessarily the order in which they were changed. In the rest of the section, we use the term change-sets for the log entries in *Subversion* repositories.

2) Convert to Fine-grained Change-sets

The differences in a file of a change-set could be readily obtained at the file-and-line level granularity (*e.g.,* with the *GNU diff* utility). These line differences in the files need to be mapped to the corresponding fine-grained differences in the syntactic constructs. The proposed approach employs a lightweight methodology for fine-grained differencing of files in a change-set. The previous and current versions of a source code file are processed using a word-differencing tool, namely *dwdiff* (http://os.ghalkes.nl/dwdiff.html). This differencing produces two source code files along with the changed locations. The first file is marked with the exact locations from where tokens, *i.e.*, words of a programming language, are deleted and the second file is marked with the exact locations where tokens are added. These markers are appropriately labeled with "specialized" source code comments.

Both files produced from the word differencing are converted to the *srcML* representation (Collard et al. 2003). The format preserves all the original source code contents including comments, white space, and preprocessor directives. Finally, both *srcML* files are processed with

```
<?xml version="1.0" encoding="utf-8"?>
<log>
  <log entry revision="438663">
    <author>kling</author>
    <date>2005-07-25T17:46:20.434104Z</date>
    <paths>
      <path action="M">khtml_part.cpp</path>
      <path action="M">loader.h</path>
    </paths>
    <msg>
       Do pixmap notifications when
       running ad filters.
    </msg>
  </log entry>
</log>
```

Figure 3 A Snippet of *kdelibs* Subversion Log

the standard XML processing tools to give a list of added and deleted constructs in a hierarchical manner up to the granularity of an identifier. The entire process is realized in the form of a fine-grained differencing tool, namely *codediff*. The *codediff* approach has a very close similarity to Collard's *srcDiff* representation (Maletic and Collard 2004) that achieves fine-grained differencing using line differencing (i.e., *diff*) and *srcML*. The important distinction between the two is that *codediff* achieves much finer levels of difference granularity than the *srcDiff* toolset and avoids situations of a line change cross-cutting multiple constructs. The tool *codediff* is used to process all the files in every change-set for source code differences at a fine-grained syntactic level. Alternatively, heavyweight approaches such as AST based and semantic comparisons are not practically feasible due to a very high computational cost involved in processing a number of versions (Mens 2002; Raghavan et al. 2004). Additionally, they typically require a system-wide parsing and as such may need additional files that are outside a given change-set to the extent of the entire system. Our *codediff* approach requires only processing of commits and nothing beyond.

3) Mine Evolutionary Couplings

A mining tool, namely *sqminer* (Kagdi et al. 2006), was previously developed to uncover evolutionary couplings from the set of commits (processed at fine-granularity levels with *codediff* 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 is based on the Sequential Pattern Discovery Algorithm (SPADE) (Zaki 2001) which utilizes an efficient enumeration of ordered patterns based on common-prefix subsequences and division of search space using equivalence classes. Additionally, it utilizes a vertical input-transaction format (i.e., a set of transactions for each file vs. a set of transactions consisting of files) for efficiency.

To help prune the number of candidate patterns produced by the mining techniques, patterns with redundant information are eliminated. A pattern that is frequent means that all possible patterns formed from the subsets of its files are also frequent. The support of a pattern is always less than or equal to the subset patterns. A common pruning mechanism used in frequent-pattern mining is to eliminate all the subset patterns that have the same support of the corresponding larger pattern. Such subset patterns are only used with other larger patterns and not in isolation by themselves. Therefore, they give redundant information that may be of very little meaning. As a result, only disjoint patterns (i.e., patterns with no common files) that subsume all subsets of patterns with the same or higher support are retained. Such patterns are known as *closed* patterns. Our tool produces only closed patterns.

Frequent-pattern mining algorithms typically report the support of a pattern but not the transactions in which it occurs. Our tool records the transactions in which a pattern is found. For uncovering both unordered and ordered change patterns, we use the same underlying mining algorithm. The tool *sqminer* can also be used for frequent itemset mining. In this case the transactions are formed with no ordering information of items. The configuration parameters of *sqminer* include support, maximum number of items in a pattern, mining of sequence (association) rules, and output in both a flat-file and XML format. For further detail on the XML output format of the ordered patterns and rules, we refer to (Kagdi et al. 2006).

sqminer has already been applied previously to mine co-changes at the file level (Kagdi et al. 2006), uncover/discover traceability links (Kagdi et al. 2007), and mine evolutionary couplings of localized documents (Kagdi and Maletic 2007).

For example, consider a method named *getType* in *ArgoUML*. The evolutionary coupling

{argouml/model/mdr/FacadeMDRImpl.java/getType,
argouml/model/mdr/FacadeMDRImpl.java/isAStereotype}

is mined from the commit history between versions 0.24 and 0.26.2 of *ArgoUML*. This coupling is supported by three commits with ID's 13341, 12784, and 12810. In these three commits, both *getType()* and *isAStereotype()* are found to co-change. Based on this evolutionary coupling, the association rule

{argouml/model/mdr/FacadeMDRImpl.java/getType} $\Rightarrow$
{argouml/model/mdr/FacadeMDRImpl.java/isAStereotype}

is computed. This rule has a confidence value of 1.0 (100%) and support of three (i.e., it appears in three commits). It suggests that should the method *getType()* be changed, the method *isASteretype()* is also likely to be a part of the same change with a conditional probability of 100%.

3.3 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

```
# 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 4 The procedures for computing conceptual and evolutionary couplings

```
# 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 5 The procedures for disjunctive and conjunctive impact analysis

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 5). 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-off point for deriving an estimated impact set. For a given cut-off point, μ, provided by the user, we compute the impact set of the disjunctive method $E(e_s)$ 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, when an odd number is specified as μ) and the cardinality of their union equals to 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 to μ, if possible. The procedures used to compute the underlying conceptual and evolutionary couplings are shown in Figure 4. They use typical sets of parameters needed for LSI and itemset mining algorithms.

3.4 Motivating 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*

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

Table II 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	/server/core_filters.c	/modules/proxy/mod_proxy_ftp.c	/modules/http/byterange_filter.c
4	/modules/debugging/mod_bucketeer.c	/server/core.c	**/modules/http/http_protocol.c**
5	/modules/http/byterange_filter.c	/include/ap_mmn.h	/server/core_filters.c

does not receive a transmitted 300 or 400 response prior to sending its body." In this revision, three source code files were 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 that also require modification. Using conceptual and evolutionary couplings for impact analysis, we obtain the results in Table II. As standalone techniques neither conceptual nor evolutionary coupling is capable of establishing 100% recall when the developer specifies an impact set size $\mu = 5$.

Conceptual coupling ranks */server/protocol.c* the first in the ranked list, but ranks /modules/http/http_protocol.c as 91st.

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. The conceptual coupling would provide the following set of files

{/server/protocol.c, /modules/proxy/mod_proxy_http.c, /server/core_filters.c}

while the evolutionary coupling would provide

{/modules/http/byterange_filter.c, /modules/http/http_protocol.c}.

The result of the disjunctive combination is

{/server/protocol.c, /modules/proxy/mod_proxy_http.c, /modules/http/byterange_filter.c, /modules/http/http_protocol.c, /server/core_filters.c}.

Both evolutionary and conceptual coupling contribute approximately $\mu/2$ files. When μ is odd, a user-defined preference for either type of couplings could be set to contribute the remaining one extra entity. In this case $\mu=5$, the first two ranked files by conceptual couplings are included, then the first two ranked files by evolutionary couplings are included, and finally the third ranked file by conceptual couplings is included. This results in the set of entities that also appear in Table II. Here we can see that when combined, the couplings are capable of identifying all methods requiring modification within an impact set, *i.e.*, cut-off point, of five methods. In the case of the conjunctive approach, the set intersection is used as oppose to the set union. We use μ entities as opposed of $\mu/2$ entities. In this example, the following methods appear in the set intersection of the results of conceptual and evolutionary coupling:

{/modules/http/byterange_filter.c, /modules/proxy/mod_proxy_http.c}.

The algorithm would continue to iterate through the ranked list of both techniques until the intersection is of size μ. Note that our disjunctive and conjunctive approaches result in sets as opposed to ranked lists (*i.e.,* the entities are unordered). Therefore, the order of picking entities in the final impact set between the contributions of the two types of couplings is irrelevant. In our evaluation, see Section 4, we evaluate the final impact set (and not a ranked list).

4 Case Study

In this section we describe the empirical assessment of our approach. We describe our study following the Goal-Question-Metrics paradigm (Basili et al. 1994), 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 standalone IA techniques and their combination?
- **RQ3**: Does the amount of training historical data impact the accuracy of the proposed 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 research questions directly address the effectiveness and expressiveness of an IA solution. With regards to

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

System	Version	LOC	Files	Methods	Terms
Apache(httpd)	2.2.3	311K	782	3.9K	6,583
ArgoUML	0.28	367K	1,995	13.3K	9,384
iBatis	3.0.0-216	70K	774	5.5K	3,772
KOffice 2.0.91	2.0.91	231K	6.5K	104.6K	48,513
KOffice 2.0.1	2.0.1	257K	6.7K	68.4K	32,212
jEdit	4.3	109K	503	6.4K	13,904

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. Note that we give more weight to recall. In the case of impact analysis, it is important to consider all possible source code entities that are impacted. Overlooking a single entity could possibly introduce new issues in a software system. 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, combination weights, and periods of training history used to derive evolutionary coupling data.

4.1 Evaluated Subject Software Systems

The *context* of our study is characterized by five open source software systems, namely *Apache httpd*, *ArgoUML*, *iBatis*, *KOffice* and *jEdit*. These systems represent different primary implementation languages (*e.g.,* C/C++ and Java), sizes, development environments, and application domains. *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/modeling 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. *jEdit* is a popular open-source text editor. Specifics of various system characteristics appear in Table III.

4.2 Accuracy Metrics – Precision, Recall, and F-Measure

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 two metrics are often collectively represented by *F-measure* values. 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 precision is defined by Eq. V. The recall of conceptual couplings, R_{EI}, is the mean percentage of correctly estimated changed entities over the total correctly changed entities. The recall is given by Eq. VI. F-measure considers both precision and recall, and can be interpreted as a weighted average of the precision and recall, where an $F\beta_{EI}$ score (for positive real β=2) reaches its best value at 1 and worst score at 0. Note that we set β=2 to give more weight to recall. The F-measure is given by Eq. VII.

$$P_{EI} = \frac{1}{n}\sum_{i=1}^{n}\frac{|EI_i \cap R_i|}{|EI_i|}\times 100\% \qquad \textbf{Eq. V}$$

$$R_{EI} = \frac{1}{n}\sum_{i=1}^{n}\frac{|EI_i \cap R_i|}{|R_i|}\times 100\% \qquad \textbf{Eq. VI}$$

$$F2_{EI} = (1+2^2)\frac{P_{EI}\cdot R_{EI}}{(2^2\cdot P_{EI})+R_{EI}} \qquad \textbf{Eq. VII}$$

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 indicate the couplings from the disjunctive and conjunctive methods respectively.

4.3 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:

Table IV Evolutionary training and (testing) datasets used for the empirical evaluation. The numbers within the brackets are for the testing set and those without the brackets are for the training set.

System	History				File Level		Method Level	
	Training		Testing		# of Commits	# of Files	# of Commits	# of Methods
	Releases	Days	Releases	Days				
Apache(httpd)	2.2.9-2.3.3	373	2.3.3-2.3.5	48	1,736 (287)	2,086 (982)	318 (19)	1,468 (107)
ArgoUML	0.24-0.26.2	519	0.26.2-0.28	103	3,375 (773)	4,217 (621)	1,157 (227)	7,294 (1580)
iBatis	3.0.0-190_b1 - 3.0.0-216_b7	33	3.0.0-216_b7 - 3.0.0-240_b10	11	108 (40)	461 (118)	18 (21)	94 (154)
KOffice 2.0.91	2.0.0-2.0.91	154	2.0.91-2.1.0	28	2,749 (522)	5,580 (1,072)	1,082 (29)	6,344 (106)
KOffice 2.0.1	2.0.0-2.0.1	27	2.0.1-2.0.2	12	763 (255)	1,233 (533)	577 (192)	5,530 (1438)
jEdit	4.2.0-4.3.0	719	4.3.0-4.3.2	60	2,525 (160)	584 (125)	2,344 (142)	6,344 (106)

1. Compute conceptual couplings on one particular version (*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 version in Step 1 – *Evolutionary Training Set*.
3. Select a set of commits in a history period after the selected version 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 versions.

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

4.3.1 Conceptual training sets - Corpora

We generated two sets of corpora from the subject systems corresponding to *documents* at the *file* and *method* levels of granularity. 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 3). Details of the corpora, including the versions indexed, are provided in Table III. The associated computing time was consistent with the previous uses (Antoniol et al. 2000; Poshyvanyk et al. 2009).

4.3.2 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 2.3.3. The commit history from versions 2.2.9 to 2.3.3 was considered for the evolutionary training set. These commits spanned across 373 total days. Days on which no commits were submitted were not included in the total count. We discarded all non-source code files from the commits, as our approach here focuses on source code. Commits with more than ten source files were also 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 (Zimmermann et al. 2005; Kagdi et al. 2007; Alali et al. 2008). The training set consisted of 1,736 commits with 2,086 files that were changed. In this set, 318 commits contained 1,468 function changes. Note that not all the commits with the file changes contributed to changes in functions, as not all the changed files contained changes in functions. We considered two different release periods of *KOffice*. The release history 2.0.0-2.0.91 contains the development versions that were involved toward the next major release 2.1.0 from the major release

Table V Orthogonality check for various cut-off points of conceptual (Conc), evolutionary (Evol), and their combination. The results show that conceptual and evolutionary couplings provide orthogonal information, and support a strong case for combining them.

		File Level						Method Level					
		10	**20**	**30**	**40**	**50**			**10**	**20**	**30**	**40**	**50**
Apache	Conc\Evol	33	35	35	35	37	**Apache**	Conc\Evol	48	49	51	53	61
	Evol\Conc	36	28	23	20	17		Evol\Conc	29	26	24	22	18
	Conc∩Evol	32	37	42	45	46		Conc∩Evol	23	25	25	25	21
ArgoUML	Conc\Evol	51	44	41	41	40	**ArgoUML**	Conc\Evol	24	22	22	23	25
	Evol\Conc	26	28	25	24	22		Evol\Conc	68	66	64	61	57
	Conc∩Evol	23	29	34	35	38		Conc∩Evol	8	12	14	16	18
iBatis	Conc\Evol	65	69	70	70	70	**iBatis**	Conc\Evol	86	86	87	87	87
	Evol\Conc	21	14	14	13	12		Evol\Conc	13	12	11	11	11
	Conc∩Evol	13	16	16	17	18		Conc∩Evol	1	2	2	2	2
KOffice 2.0.91	Conc\Evol	62	64	64	63	64	**KOffice 2.0.91**	Conc\Evol	57	63	63	63	63
	Evol\Conc	22	16	13	12	11		Evol\Conc	40	35	35	35	35
	Conc∩Evol	16	20	23	24	26		Conc∩Evol	3	2	2	2	2
KOffice 2.0.1	Conc\Evol	41	40	42	43	44	**KOffice 2.0.1**	Conc\Evol	48	46	47	46	46
	Evol\Conc	38	36	35	33	32		Evol\Conc	51	52	50	51	51
	Conc∩Evol	21	23	23	23	24		Conc∩Evol	1	2	3	3	3
jEdit	Conc\Evol	24	20	23	24	26	**jEdit**	Conc\Evol	12	8	6	9	9
	Evol\Conc	44	39	40	43	41		Evol\Conc	85	85	86	84	83
	Conc∩Evol	32	41	37	33	33		Conc∩Evol	3	7	8	7	8

2.0.0. The release history 2.0.0-2.0.2 includes the maintenance release for the major release 2.0.0. These two history periods allow us to investigate development and maintenance periods.

The tool *sqminer* was employed to mine evolutionary couplings (and association rules) in the *itemset mining* mode with the minimum support values of 1, 2, 4, and 8. Also, we considered all the possible (binary) 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.

4.3.3 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 version 2.3.3 and up to the version 2.3.5 was considered for the testing set. The testing set consisted of 287 commits with 982 files that were changed. In this set, 19 commits contained 107 function changes. Similar to the training set, not all the commits with the file changes contributed to changes in functions in the testing set, as not all the changed files contained changes in functions. 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 (Zimmermann et al. 2005; Kagdi et al. 2007).

Table IV shows the details of the evolutionary training and testing sets considered at the file and method levels. The file level granularity is on the left-hand side of the table, whereas method level granularity is on the right-hand side. They include a range of versions 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 IV) periods before and after the *Versions* (Table III) 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). We used *codediff* to process the commits at the method-level granularity level here, whereas, *srcDiff* was used in our previous work (Kagdi et al. 2010). This choice of a different tool helps address the differencing-specific tool bias in the processing of the commits at the fine-grained method level.

Table VI *File level granularity*: F-measure percentage (%) results of conceptual coupling (Conc), evolutionary coupling (Evol), and disjunctive (Disj) approaches to impact analysis for all systems using various cut-off points. The disjunctive approach outperforms standalone techniques with statistical significance (highlighted in bold).

		10	**20**	**30**	**40**	**50**		**10**	**20**	**30**	**40**	**50**
Conc	**Apache**	25	22	21	19	16	**KOffice 2.0.91**	22	20	18	16	12
Evol		27	20	15	12	12		14	12	10	8	8
Disj		**34**	**29**	**22**	**20**	**17**		**29**	**25**	**21**	**19**	**16**
Conc	**ArgoUML**	16	14	13	14	11	**KOffice 2.0.1**	17	13	11	11	8
Evol		11	11	10	10	8		20	16	14	11	8
Disj		**19**	**17**	**16**	14	**15**		**27**	**22**	**18**	**15**	**16**
Conc	**iBatis**	27	29	24	22	19	**jEdit**	11	10	8	8	4
Evol		12	9	7	8	8		13	11	8	8	8
Disj		**29**	**30**	**27**	22	19		**14**	11	8	8	8

4.4 Results

We present the findings from investigating our research questions below.

*4.4.1 **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.

Orthogonality of the Conceptual and Evolutionary Couplings. 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, as they would be two different ways to produce the same result. In order to measure the degree to which the techniques could potentially complement each other, we use the metrics given in Eq. VIII and Eq. IX:

$$correct_{m_i \cap m_j} = \frac{\left| correct_{m_i} \cap correct_{m_j} \right|}{\left| correct_{m_i} \cup correct_{m_j} \right|} \% \quad \text{Eq. VIII}$$

$$correct_{m_i \setminus m_j} = \frac{\left| correct_{m_i} \setminus correct_{m_j} \right|}{\left| correct_{m_i} \cup correct_{m_j} \right|} \% \quad \text{Eq. IX}$$

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 types of couplings for the various systems at both file and method levels are given in Table V. For example, consider the file level granularity results for *Apache* for the impact set size of 10, i.e., a cut-off point of 10. The *Conc\Evol* value is 33%, which is the percentage of the correctly contributed files by the conceptual coupling alone. The *Evol\Conc* value is 36%, which is the percentage of the correctly contributed files by the evolutionary coupling alone. The *Conc∩Evol* value is 32%, which is the percentage of the common contribution of both conceptual and evolutionary couplings. Thus, 71% of the correctly

Table VII *Method level granularity*: F-measure percentage (%) results of conceptual coupling (Conc), evolutionary coupling (Evol), and disjunctive (Disj) approaches to impact analysis for all systems using various cut-off points. The disjunctive approach outperforms standalone IA techniques with statistical significance (highlighted in bold).

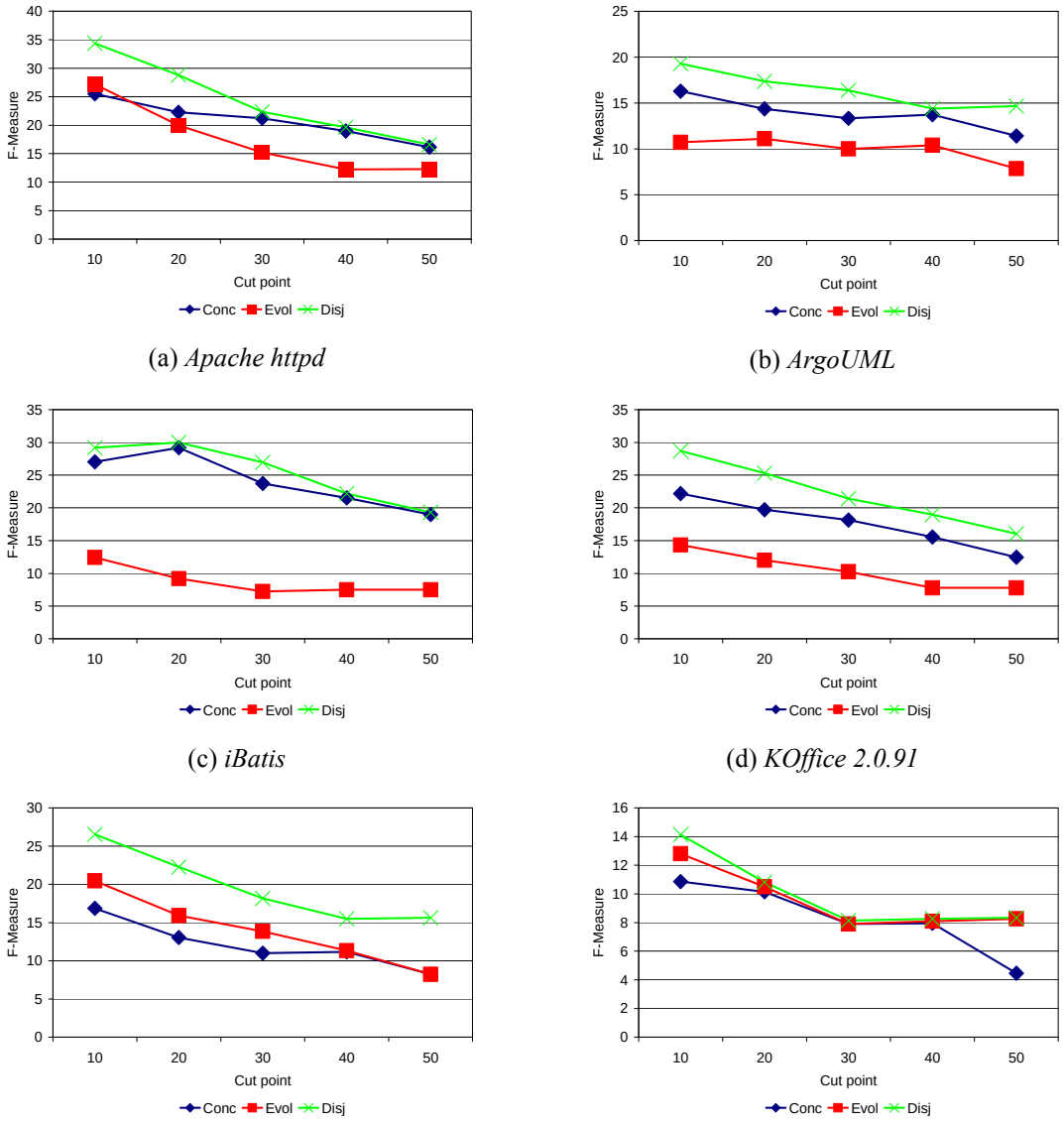
		10	**20**	**30**	**40**	**50**		**10**	**20**	**30**	**40**	**50**
Conc	**Apache**	18	17	14	14	11	**KOffice 2.0.91**	11	9	10	8	8
Evol		14	15	14	13	11		10	10	8	7	7
Disj		**23**	**23**	**22**	**21**	**20**		**19**	**18**	**16**	**15**	**12**
Conc	**ArgoUML**	4	3	4	4	4	**KOffice 2.0.1**	4	4	4	3	3
Evol		12	14	14	13	13		12	13	12	12	11
Disj		**16**	**17**	**17**	**16**	**15**		**14**	**16**	**14**	**14**	**12**
Conc	**iBatis**	15	15	13	11	11	**jEdit**	2	0	0	0	0
Evol		4	3	3	3	0		9	9	7	7	7
Disj		**20**	15	13	11	11		**10**	9	7	7	7

recommended files are individually contributed, indicating the orthogonally of these couplings. Based on our datasets, the overlap between the sets of correctly estimated IA entities by the two approaches did not exceed 46% and 21% at the file and method levels. A minimal overlap indicates potential orthogonality between the two techniques. One exception is the case where virtually all the correct entities identified by one technique makes 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 that is 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.

F-measure Performance of the Standalone and Combined Approaches. Based on our datasets, conceptual and evolutionary 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 VI and Table VII provide F-measure results for the subject systems under study at class and method level granularity respectively. These results are obtained by using the various couplings for IA. Only a subset of the cut-off points (μ) we considered are shown in Table VI and Table VII. The cut-off points represent the sizes of the impact set considered with our combinations. For example, a cut-off point of 10 indicates that the estimated impact set with our approach contained 10 entities.

We considered both disjunctive and conjunctive approaches to combining couplings. The disjunctive approach outperforms the conjunctive approach in all cases considered (see Table VI and Table VII). 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 V). The orthogonality between the sets of correct entities identified by the two couplings appears to contribute to the lackluster 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 two couplings. The rest of the discussion about the combinations of two couplings refers to the disjunctive approach.

The prevailing pattern in our finding is that the combination of conceptual and evolutionary couplings improves the performance over either standalone technique. Consider a case in Table VI where μ = 30 (i.e., the impact set size is 30 files) for *Apache httpd* at the file level granularity. Conceptual and evolutionary couplings yield F-measure values of 21% and 15% respectively, while the combination of the two increases F-measure to 22%. Similar improvements are apparent throughout all the datasets considered in our evaluation. Another example is where $\mu = 50$ for



(a) *Apache httpd* (b) *ArgoUML*

(c) *iBatis* (d) *KOffice 2.0.91*

(e) *KOffice 2.0.1* (f) *jEdit*

Figure 6 *File level granularity*: F-measure (F) percentages results of conceptual coupling (Conc), evolutionary coupling (Evol), and disjunctive (Disj) approaches to impact analysis for all systems using various cut-off points. ImpC and ImpE show the improvement obtained by the disjunctive approach compared to conceptual and evolutionary couplings respectively. The disjunctive approach outperforms standalone techniques with statistical significance.

KOffice 2.0.1 (file-level granularity). In this case, both couplings yield the F-measure value of 8%, while the combination of the two increases F-measure to 16%. 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.

Figure 6 and Figure 7 show the results of F-measures across cut-off points at the file-level granularity and method-level granularity respectively. The blue (diamond), red (square), and green (cross) curves show the performances of conceptual, evolutionary, and disjunctive approaches. We observe that the disjunctive approach generally outperforms both conceptual and evolutionary couplings across the cut-off points. Also, we notice that conceptual couplings tend to provide better accuracy (with the apparent exception of *jEdit*) than the evolutionary couplings. These trends are consistent regardless of the level of granularity. This indicates that the disjunctive approach provides an improvement despite the change in granularity.

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. Although, we have summarized all the results with F-measure, we wanted to determine the individual statistical impact of both precision and recall metrics. We developed four testable null hypotheses (both at file and method level granularity):

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

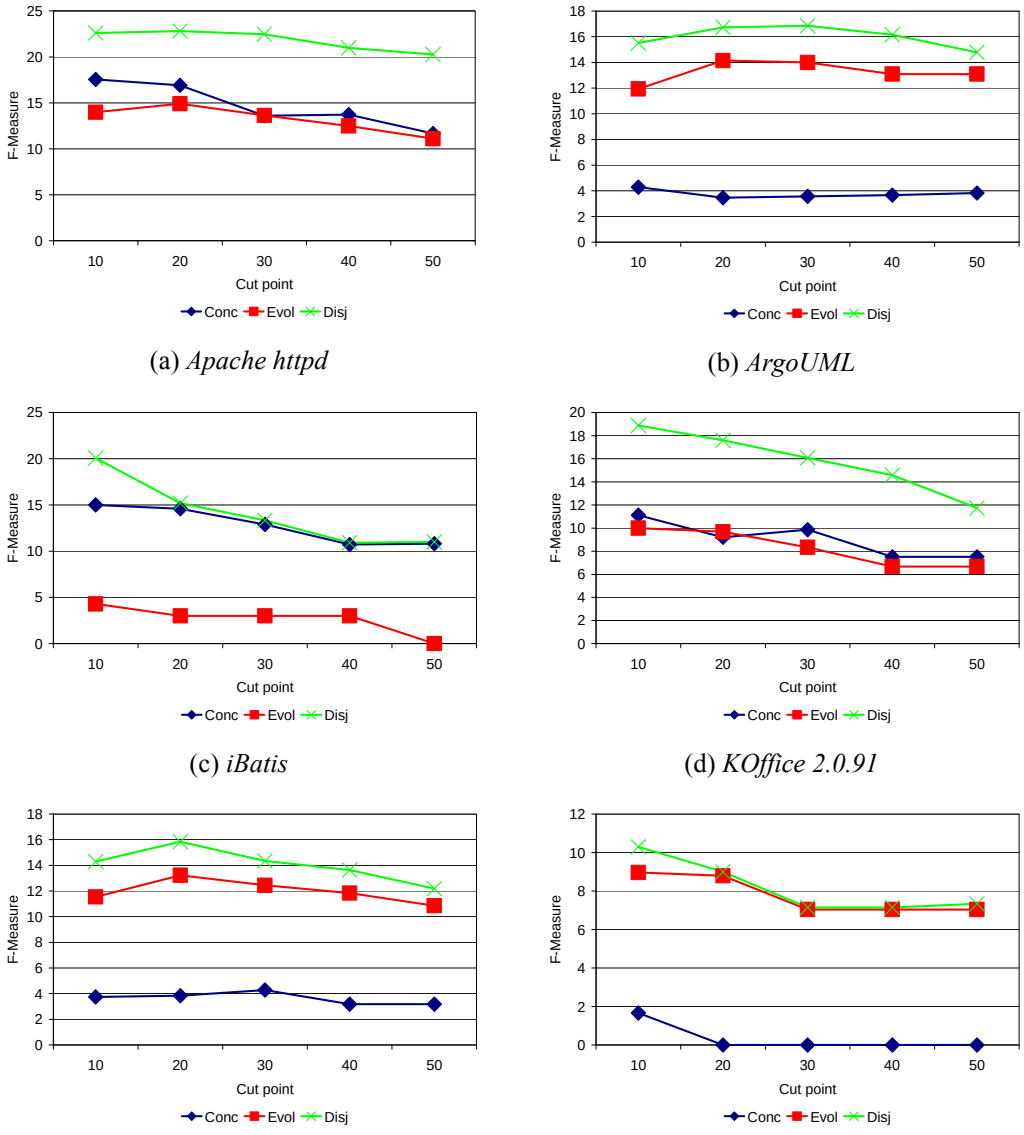
$H_{0\ CR}$: Combining conceptual and evolutionary couplings *does not significantly* improve recall results of impact analysis compared to conceptual couplings.

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

$H_{0\ ER}$: 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:

$H_{a\ CP}$: Combining conceptual and evolutionary couplings *significantly* improves precision results of impact analysis compared to conceptual couplings.



(a) *Apache httpd* (b) *ArgoUML*

(c) *iBatis* (d) *KOffice 2.0.91*

(e) *KOffice 2.0.1* (f) *jEdit*

Figure 7 *Method level granularity*: F-measure (F) percentages results of conceptual coupling (Conc), evolutionary coupling (Evol), and disjunctive (Disj) approaches to impact analysis. Disjunctive approach outperforms the two standalone approaches with statistical significance.

Table VIII Results of *Wilcoxon* signed-rank test (μ = 30). The *p*-values of less than 0.05 (marked in green) indicate that the disjunctive approach provided improvement is not by chance.

Granularity	System	$H_{0\,CP}$	$H_{0\,CR}$	$H_{0\,EP}$	$H_{0\,ER}$
File Level Granularity	Apache(httpd)	0.0002	0.0003	0.0001	0.0003
	ArgoUML	0.0050	0.0039	< 0.0001	< 0.0001
	iBatis	0.0126	0.0126	0.0001	0.0002
	KOffice 2.0.91	< 0.0001	< 0.0001	< 0.0001	< 0.0001
	KOffice 2.0.1	< 0.0001	< 0.0001	< 0.0001	< 0.0001
	jEdit	0.0300	0.0070	0.0180	0.0180
Method Level Granularity	Apache(httpd)	0.0160	0.0070	0.0180	0.0180
	ArgoUML	< 0.0001	< 0.0001	< 0.0001	< 0.0001
	iBatis	0.0010	0.0010	0.3100	0.3100
	KOffice 2.0.91	0.0200	0.0003	0.0100	0.0100
	KOffice 2.0.1	< 0.0001	< 0.0001	< 0.0001	< 0.0001
	jEdit	< 0.0001	< 0.0001	0.1000	0.1000

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. We use the testing data (see Table IV) as our sample. 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 VIII 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 at the file level granularity. With a few exceptions, we obtained a p-value of less than 0.0125 (applying Bonferroni correction to ensure the four null hypotheses are correct 95% of the time – 0.05 * 1/4), indicating that the improvement in accuracy obtained is not by chance. Thus, we rejected all our null hypotheses and accepted the alternate hypotheses.

*4.4.2 **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 systems *Apache, ArgoUML, iBatis, KOffice 2.0.1, KOffice 2.0.92,* and *jEdit*. For these systems we obtained results at both file and method levels of granularity. F-measure results of the techniques for IA are shown in Table VI and Table VII. There is a noticeable decrease in F-measure values when the method level granularity is used. Consider a case in Table VII where μ = 40 (i.e., the impact set size is set of 40 files) for *Apache httpd*. Conceptual and evolutionary couplings yield the F-measure value of 14% each, while the combination of the two increases it to 22%. Similar improvements are apparent throughout all the datasets across all the three metrics considered in our evaluation. Figure 7 shows the results of F-measures across cut-off points at the method-level granularity. The blue (diamond), red (square), and green (cross) curves show the performances of conceptual, evolutionary, and disjunctive approaches. We observe that the disjunctive approach generally outperforms both conceptual and evolutionary couplings across the cut-off points. Also, we notice that evolutionary couplings tend to provide better accuracy (with the apparent exception of *iBatis* and *KOffice 2.0.91*) than the conceptual couplings. The training set for *KOffice 2.0.91* contained only minor releases, which indicates the mined evolutionary couplings may not provide for accurate and complete recommendations. The training and testing sets of *iBatis* consisted of a relatively small number of commits and entities compared to other systems in our evaluation.

Overall, conceptual couplings are more affected by the difference in granularity than evolutionary couplings. For the conceptual couplings, going from the coarse granularity of files to the finer granularity of methods resulted in the reduction of the sizes of the documents. The documents were reduced in terms (and frequency). That is, a file is typically much "bigger" than a method. Previous works on the application of information retrieval techniques in software

Table IX The amounts of change history periods considered for the five systems.

System	Release Interval	Date Interval	Days with Changes	# Commits
Apache(httpd)	[2.2.8, 2.3.3]	[2008-01-10, 2009-11-11]	481	2,320
	[2.2.9, 2.3.3]	[2008-06-10, 2009-11-11]	373	1,788
	[2.3.0, 2.3.3]	[2008-12-07, 2009-11-11]	256	1,286
	[2.3.1, 2.3.3]	[2009-01-03, 2009-11-11]	231	1,112
	[2.3.2, 2.3.3]	[2009-03-23, 2009-11-11]	176	871
ArgoUML	[0.14, 0.26.2]	[2003-12-15, 2008-11-19]	1,549	9,713
	[0.16, 0.26.2]	[2004-07-19, 2008-11-19]	1,369	8,989
	[0.18, 0.26.2]	[2005-04-30, 2008-11-19]	1,101	7,207
	[0.20, 0.26.2]	[2006-02-09, 2008-11-19]	837	5,625
	[0.22, 0.26.2]	[2006-08-08, 2008-11-19]	676	4,380
	[0.24, 0.26.2]	[2007-02-12, 2008-11-19]	519	3,375
	[0.26, 0.26.2]	[2008-09-27, 2008-11-19]	41	244
iBatis	3.0.0-190_[b1, b7]	[2009-08-03, 2009-12-19]	33	96
	3.0.0-190_[b2, b7]	[2009-08-16, 2009-12-19]	32	94
	3.0.0-190_[b3, b7]	[2009-08-29, 2009-12-19]	30	90
	3.0.0-190_[b4, b7]	[2009-10-10, 2009-12-19]	17	42
	3.0.0-190_[b5, b7]	[2009-11-01, 2009-12-19]	11	28
	3.0.0-190_[b6, b7]	[2009-12-05, 2009-12-19]	5	16
KOffice 2.0.91	[2.0.81, 2.0.91]	[2009-08-23, 2009-10-23]	60	1,184
	[2.0.82, 2.0.91]	[2009-09-10, 2009-10-23]	43	815
	[2.0.83, 2.0.91]	[2009-10-03, 2009-10-23]	20	296
jEdit	[4.3-pre1, 4.3]	[2004-12-31, 2009-12-22]	719	2,051
	[4.3-pre2, 4.3]	[2005-03-12, 2009-12-22]	481	1,388
	[4.3-pre4, 4.3]	[2006-05-12, 2009-12-22]	477	1,379

engineering have observed a similar sensitivity trend with respect to the size of the document and the accuracy of the intended tasks (Revelle et al. 2011). For the evolutionary couplings, only the co-change information in commits was utilized for evolutionary couplings. The accuracy difference from the file level to the method level was due to the fact that there was no one-to-one mapping between file and method changes in commits. That is, not all the files that were involved in the commits had method-level changes. On the other hand, there were files with multiple method level changes. As a result, despite of a correct recommendation of an impact set at the file level, there was no corresponding recommendation at the method level. It is interesting to note that there was also an observation on the opposite end of the spectrum. That is, we observed a gain in accuracy at the method level over the file level (precision of 11% over 6% for Apache). The "false" couplings inferred at the file level granularity from the training set (e.g., *f1* → *f2*) were not repeated in the testing set (*f1* → *f3)*. The method-level couplings identified from a single file (e.g., *m1* → *m2* from the file *f1*) in the training set were repeated in the testing set (e.g., *m1* → *m2* in the file *f1*). Overall, the accuracy difference for evolutionary couplings across the granularly levels do not show a sharp change. Regardless of the decrease in accuracy of the standalone techniques, when the two are combined there exists a statistically significant improvement in accuracy. 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.

Table VI and Table VII present the results for impact analysis while using the file and method granularity levels respectively. In general, our results indicate that granularity indeed impacts accuracy. In many cases the decrease in accuracy is quite noticeable. For example, for $\mu = 30$ in for *iBatis*, conceptual couplings yield decreasing F-measure values from 24% to 15% and evolutionary couplings yield decreasing F-measure value from 7% to 3%. Similar trends occur for all the software systems considered. One point of interest is the fact that for the method level granularity, evolutionary couplings provided better accuracy results for *ArgoUML* while conceptual couplings provided generally better results at file level granularity.

Our results indicate 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.

Table VIII also presents the results of performing the *Wilcoxon* signed-rank test for the method level granularity. We performed the test for each of the four hypotheses, formulated similar to that

Table X F-measure percentage (%) results of conceptual coupling (Conc), evolutionary coupling (Evol), and disjunctive (Disj) approaches to impact analysis for KOffice 2.0.91 (different periods of history) using various cut-off points. The disjunctive approach outperforms standalone IA techniques with statistical significance (highlighted in bold).

		File Level						Method Level				
		10	**20**	**30**	**40**	**50**		**10**	**20**	**30**	**40**	**50**
Conc	**KOffice 2.0.81**	22	20	18	16	12	**KOffice 2.0.81**	11	9	10	8	8
Evol		10	7	7	4	4		10	10	8	7	7
Disj		**27**	**23**	**19**	16	**16**		**19**	**18**	**16**	**15**	**12**
Conc	**KOffice 2.0.82**	22	20	18	16	12	**KOffice 2.0.82**	4	4	4	3	3
Evol		10	7	7	4	4		12	13	12	12	11
Disj		**27**	**23**	**19**	16	**16**		**14**	**16**	**14**	**14**	12
Conc	**KOffice 2.0.83**	22	20	18	16	12	**KOffice 2.0.83**	2	0	0	0	0
Evol		5	4	4	4	4		9	9	7	7	7
Disj		**24**	**22**	**19**	16	**16**		**10**	9	7	7	7

of the file level granularity, 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 at the method level granularity. Although, we have summarized all the results with F-measure, we wanted to determine the individual statistical impact of both precision and recall metrics. At the file level granularity, we obtained a p-value less than 0.05 for all the hypothesis, indicating that the improvement in accuracy obtained over the conceptual and evolutionary couplings is not by chance. In all cases considered at the method level, we obtained a p-value of less than 0.05 for the first two hypotheses, indicating that the improvement in accuracy obtained over the conceptual couplings is not by chance. Thus, we rejected these two null hypotheses and accepted the alternate hypotheses. For the third and fourth hypotheses, *Apache*, *ArgoUML*, and *KOffice* (the three largest systems in our dataset) showed p-values of less than 0.05, whereas *iBatis and jEdit* (the three smallest systems in our dataset) showed p-values of greater than 0.05. Therefore, we rejected these two null hypotheses and accepted the alternate hypotheses for *Apache*, *ArgoUML*, and *KOffice*, indicating that the improvement in accuracy obtained over the evolutionary couplings is not by chance. Our statistical analysis results show that the disjunctive approach outperforms both evolutionary and conceptual couplings at the file level of granularity across all the considered systems. Also, the disjunctive approach tends to outperform both evolutionary and conceptual couplings at the method level of granularity for larger systems in terms of number of files and methods.

*4.4.3 **RQ3**: Does the amount of training historical data impact the accuracy of the proposed combination?*

Our third research question examines the impact of history on our proposed approach. We considered various amounts of history periods for different systems. A wide spectrum of kinds of releases, including major, minor, development, prereleases, and beta, was considered. Table IX shows the specific release intervals, date intervals, number of days with changes, and number of commits for all the considered systems. Note that the values in the days with changes column may not align with the difference given by the corresponding release intervals. This mismatch is due to the fact that not all calendar days would have commits. For example, the date interval [2009-01-03, 2009-11-11] has only 231 days with commits.

We analyze *KOffice 2.0.91* in detail and summarize the F-measure performance of the disjunctive approach on all the considered systems. For *KOffice 2.0.91*, we obtain evolutionary couplings using three different periods of history from the versions 2.0.83, 2.0.82, and 2.0.81, each as a starting point to the version 2.0.91. The interval [2.0.81, 2.0.91] represents the largest training period, whereas the interval [2.0.83, 2.0.91] represents the shortest training period for evolutionary couplings. These three versions are the minor releases from the development of the major release 2.1.0 of *KOffice*.

Table X shows the results of using different periods of history in our combined approach. They demonstrate that using a larger period of history improves the F-measure of the results obtained for both evolutionary coupling, as well as the combination of evolutionary and conceptual couplings. For example, results for the file level granularity show that the F-measure value increases from 24%

to 27% whenever more history is used. More specifically, the larger period of history, starting from version 2.0.81 as oppose to 2.0.83, demonstrates a clear improvement in accuracy at cut-off point 10, for instance. Similarly, when using the method level granularity, F-measure value increases from 10% to 19% at the cut-off point of 10 when the history starts from version 2.0.81 instead of 2.0.83. The improvement of the standalone technique also carries over to the disjunctive combination, as attested by the results.

Figure 8 shows the F-measure performances of the disjunctive approach with the change in the amount of history for the considered five systems. Figure 8 (g) and Figure 8 (h) show the F-measure values of *KOffice 2.0.91* corresponding to the specific history periods across cut-off points for the file and method levels of granularity. The change in F-measure values at the file level does not show any noticeable difference with the increase in the amount of history (except a slight change for the cut-off point 10). This observation indicates that there may be an increase in either precision or recall values with an increase in the considered history period; however, there may not be any noticeable difference in combined accuracy with the F-measure. Another interesting observation is that the F-measure performance of the disjunctive approach shows a considerable change at the method level granularity. Therefore, our findings suggest that when the development versions of history are considered in the training set, the finer (method) granularity performance may improve with the increase in change history period.

Figure 8 also shows the F-measure performances of the other four systems across the cut-off values (shown with different marked lines) for the different commit duration. The x-axis shows the number of commits involved in each considered history duration. These versions and durations are given in Table IX. *ArgoUML* shows the same behavior as *KOffice 2.0.91*. The results for *ArgoUML* at the file and method levels of granularity are presented in Figure 8 (c) and Figure 8 (d) respectively. This similarity in performance could be attributed to the fact that development versions for major releases were considered for *ArgoUML*, which was also the case with *KOffice 2.0.91*. Also, these two systems were the largest systems considered in our evaluation. For the other systems *Apache httpd*, *jEdit*, and *iBatis*, the performance in F-measures do not show any noticeable difference with the change in the amount of history. These three systems had smaller numbers of commits and changed entities than *ArgoUML* and *KOffice*. Small differences in these systems were largely limited to smaller cut-off points, and were more prone at the method level than at the file level. For *iBatis*, only the beta (test) versions were considered. As such, the changes between consecutive releases did not contribute much to improving the predictions in the testing set. We considered all the major releases for *jEdit* from 2004 to 2007. The testing set considered in this case was much smaller than the training set (2,344 vs. 142 files). For *jEdit*, the performance of the testing had peaked for the initial training set (release 4.2 to 4.3) considered. We tried building training sets with pre-release change history for *jEdit*. The flat lines in Figure 8 (i) and Figure 8 (j) for the file and method levels of granularity indicate that when a relatively small testing set is considered only the most recent (release) history for the training set could be sufficient.

In summary, it was observed that increasing the training history with development versions across major releases (of larger systems) had an impact on the F-measure value, which was more noticeable at the method-level granularity than the file level granularity.

4.5 Threats to validity

We address some of the threats to validity, including internal and external, that could have impacted our empirical study and results.

Results are specific to only a certain parameter setting:

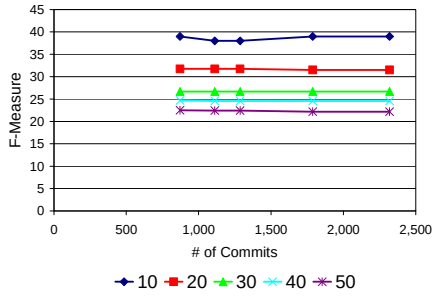
The uses of LSI and itemset mining algorithms are sensitive to a set of user-defined parameters. It is a viable risk that the improvements gained by our approach are valid only for a particular set of these parameter values. To address this risk, we experimented with different parameter values. For example, the accuracy of evolutionary couplings decreases with an increase in the minimum support value; however, the trend of accuracy gains continued with our approach. With respect to LSI, we set the dimensionality reduction parameter, k = 300. We will continue our quest to obtain the optimal values with other studies in the future.

The measures used for accuracy do not depict a true account:

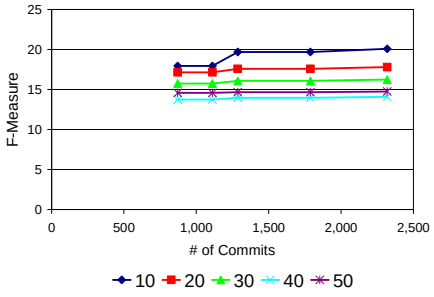
We measured the accuracy of IA with precision, recall, and F-measure metrics. It is possible that a different accuracy metric may produce a different result; however, both these metrics are widely used and accepted in the community, including for IA. We tried with F-measure, which is based on precision 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.

The gold standard used for evaluation is not realistic:

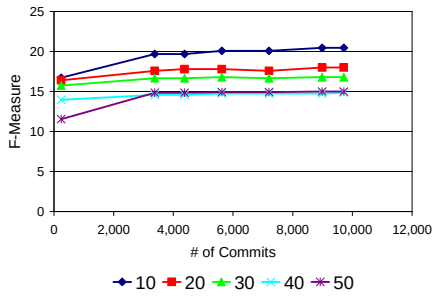
It is reasonable to assume that not all the entities in a commit are related to a single change request, and a single commit may not capture all the entities related to a change request. Therefore,



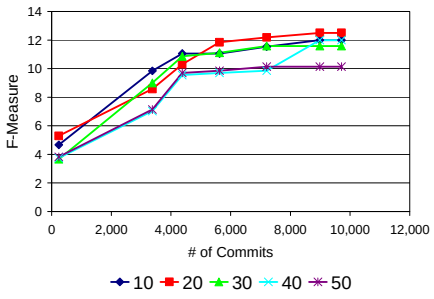
(a) *Apache httpd (File)*



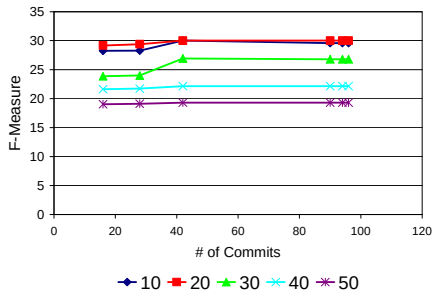
(b) Apache httpd (Method)



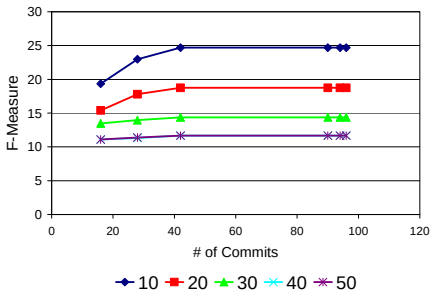
(c) *ArgoUML (File)*



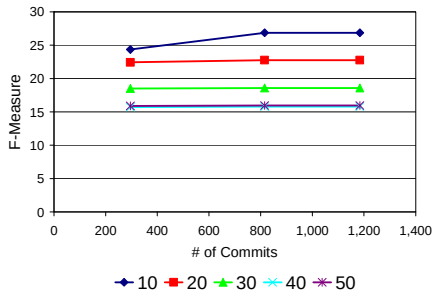
(d) *ArgoUML (Method)*



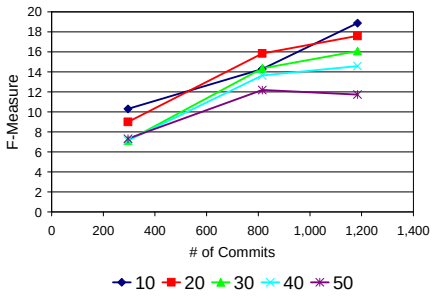
(e) *iBatis (File)*



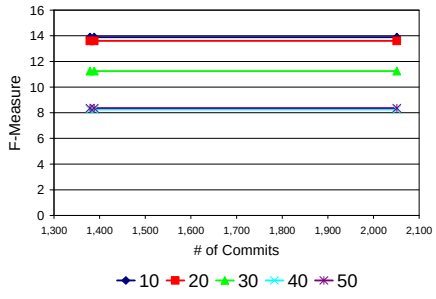
(f) *iBatis (Method)*



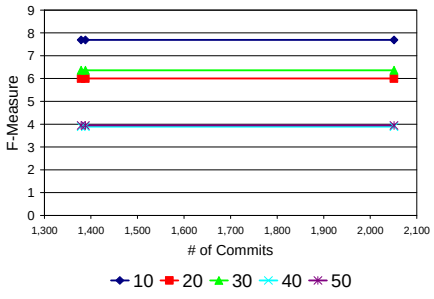
(g) *KOffice 2.0.91 (File)*



(h) *KOffice 2.0.91 (Method)*



(i) *jEdit (File)*



(j) *jEdit (Method)*

Figure 8 Impact of the amount of training-set history on the accuracy gains of the disjunctive approach on the five systems considered. F-measure was used for the accuracy. The charts on the left and right correspond to the file and method levels of granularity.

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. (Zimmermann et al. 2005)). We did some manual inspection. We examined randomly selected 5-10% of the commits in the testing set from each system. We were not able to find a strong evidence of files that were checked in together in a single commit, but were not related to each other. This observation could be an attributed of our sampled data, and may not be a general phenomenon. Therefore, we plan to conduct a user study with developers to establish actual impact sets in the future.

Instruments used in mining of evolutionary couplings may introduce errors:

The commits used in the training and testing sets were obtained using a *pysvn* API programmatically. Although, *pysvn* is robust and reliable, it is possible that we might have missed a few commits. We compared the results of *pysvn* with that of *Subversion*'s command line interface (i.e., *svn co*), and did not record any differences in the results. We processed the commits at the method level granularity using lightweight syntactic markup language (*srcML*) and differencing tools. It is possible that some of the methods may have been missed or incorrectly identified. For mining the patterns and rules for evolutionary couplings, we used standard itemset and association mining algorithms.

Results are specific to only a source code granularity level:

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 (Zimmermann et al. 2005) 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 (Zimmermann et al. 2005). Our work shows how to improve accuracy by forming effective combinations.

Results may hold for a specific training-set interval

Evolutionary couplings, one of the two major sources of information, used in our approach are mined from a chosen commit history period. This selection of training set introduces the risk that results may only hold for a specific period. To analyze the results carefully and mitigate this threat, we considered training sets from all minor, major, development, maintenance, and beta (test) versions of software systems. Additionally, we also assessed the performance of our approach across different sizes of training periods.

Results may not hold for other systems:

We evaluated on datasets from five 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.

5 Conclusions and future work

We presented an approach to software change impact analysis that combines two orthogonal sources of information: conceptual and evolutionary couplings. These couplings are derived from applying information retrieval and mining software repositories techniques. The empirical assessment on five open source systems provides support for our approach with several conclusions that combining conceptual and evolutionary couplings improves accuracy of impact analysis. Our findings indicate that in certain cases an improvement of 21% in F-measure is achieved when conceptual and evolutionary couplings are combined. The overall improvement obtained when combining the two techniques is statistically significant for most of the datasets used in our evaluation at the file level granularity and there exists a similar support at the method level granularity for larger datasets such as *Apache*, *ArgoUML*, and *KOffice*. Although our combining methods of couplings may appear straightforward, it did provide effective 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 the 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. The benefits of the approach, particularly in the larger systems are clearly evident with statistical significance at both the file and method level granularities, which is where improved support for impact analysis maybe much needed. Thus, there is strong evidence showing the value of the combination of conceptual and evolutionary couplings at various levels of granularity. Furthermore, we observe that larger

periods of history improve evolutionary couplings and subsequently its combination with the conceptual couplings.

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 previous studies (Antoniol et al. 2000; Poshyvanyk et al. 2009), 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.

Finally, the data used in producing the results in this paper is publicly available and other researchers are encouraged to reproduce or verify our results (http://www.cs.wm.edu/semeru/data/emse-impact-analysis).

Acknowledgements: We are grateful to the EMSE and WCRE 2010 reviewers, and WCRE 2010 conference participants for their pertinent comments that helped us in improving the quality of this extended paper. We would also like to thank Bogdan Dit for his help on improving the presentation of this paper. This work is supported by United States NSF CCF-1016868, NSF CCF-0916260, NSF CCF-1156401, and NSF CCF-1218129 grants. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

References

Abebe S. L., Haiduc S., Marcus A., Tonella P. and Antoniol G. (2009) Analyzing the Evolution of the Source Code Vocabulary. 13th IEEE European Conference on Software Maintenance and Reengineering (CSMR'09), Kaiserslautern, Germany, 189-198.

Agrawal R. and Srikant R. (1995) Mining Sequential Patterns. 11th International Conference on Data Engineering, Taipei, Taiwan, IEEE Computer Society: Los Alamitos CA.

Alali A., Kagdi H. and Maletic J. I. (2008) What's a Typical Commit? A Characterization of Open Source Software Repositories. 16th IEEE International Conference on Program Comprehension (ICPC'08), Amsterdam, The Netherlands.

Antoniol G., Canfora G., Casazza G., De Lucia A. and Merlo E. (2002) Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering **28**(10): 970 - 983.

Antoniol G., Canfora G., Casazza G. and Lucia A. (2000) Identifying the Starting Impact Set of a Maintenance Request: A Case Study. 4th European Conference on Software Maintenance and Reengineering (CSMR'00), Zurich, Switzerland, 227-231.

Antoniol G., Gueheneuc Y.-G., Merlo E. and Tonella P. (2007) Mining the Lexicon Used by Programmers during Software Evolution. 23rd IEEE International Conference on Software Maintenance (ICSM'07), Paris, France, IEEE Computer Society Press, 14-23.

Anvik J., Hiew L. and Murphy G. C. (2006) Who should fix this bug? 28th International Conference on Software Engineering (ICSE'06), 361-370.

Arnaoudova V., Eshkevari L., Oliveto R., Guéhéneuc Y.-G. and Antoniol G. (2010) Physical and conceptual identifier dispersion: Measures and relation to fault proneness. 26th IEEE International Conference on Software Maintenance (ICSM'10), Timisoara, Romania, 1-5.

Basili V. R., Caldiera G. and Rombach. D. H. (1994) The Goal Question Metric Paradigm, John W & S.

Binkley D., Davis M., Lawrie D. and Morrell C. (2009) To CamelCase or Under_score. 17th IEEE International Conference on Program Comprehension (ICPC'09), Vancouver, British Columbia, Canada, IEEE, 158-167.

Binkley D. and Lawrie D. (2010a). Information Retrieval Applications in Software Development. Encyclopedia of Software Engineering. P. Laplante, Taylor & Francis LLC.

Binkley D. and Lawrie D. (2010b). Information Retrieval Applications in Software Maintenance and Evolution. Encyclopedia of Software Engineering. P. Laplante, Taylor & Francis LLC.

Bohner S. and Arnold R. (1996) Software Change Impact Analysis. Los Alamitos, CA, IEEE Computer Society.

Briand L., Labiche Y. and Soccar G. (2002) Automating Impact Analysis and Regression Test Selection Based on UML Designs. International Conference on Software Maintenance (ICSM'02), Montreal, Quebec, Canada, 252-261.

Briand L., Wust J. and Louinis H. (1999) Using Coupling Measurement for Impact Analysis in Object-Oriented Systems. IEEE International Conference on Software Maintenance (ICSM'99), IEEE Computer Society Press, 475-482.

Briand L. C., Daly J. and Wüst J. (1999) A Unified Framework for Coupling Measurement in Object Oriented Systems. IEEE Transactions on Software Engineering **25**(1): 91-121.

Canfora G., Ceccarelli M., Cerulo L. and Di Penta M. (2010) Using Multivariate Time Series and Association Rules to Detect Logical Change Coupling: an Empirical Study. 26th IEEE International Conference on Software Maintenance, Timisoara, Romania.

Canfora G. and Cerulo L. (2005) Impact Analysis by Mining Software and Change Request Repositories. 11th IEEE International Symposium on Software Metrics (METRICS'05), 20-29.

Caprile C. and Tonella P. (1999) Nomen Est Omen: Analyzing the Language of Function Identifiers. 6th IEEE Working Conference on Reverse Engineering (WCRE'99), Atlanta, Georgia, USA, 112-122.

Chen K. and Rajlich V. (2000) Case Study of Feature Location Using Dependence Graph. 8th IEEE International Workshop on Program Comprehension (IWPC'00), Limerick, Ireland, 241-249.

Chen K. and Rajlich V. (2001) RIPPLES: Tool for Change in Legacy Software. International Conference on Software Maintenance (ICSM'01), Florence, Italy, 230-239.

Cleland-Huang J., Czauderna A., Gibiec M. and Emenecker J. (2010) A machine learning approach for tracing regulatory codes to product specific requirements. 32nd ACM/IEEE International Conference on Software Engineering (ICSE'10), Cape Town, South Africa 155-164.

Collard M. L., Kagdi H. H. and Maletic J. I. (2003) An XML-Based Lightweight C++ Fact Extractor. 11th IEEE International Workshop on Program Comprehension (IWPC'03), Portland, OR, IEEE-CS, 134-143.

De Lucia A., Fasano F., Oliveto R. and Tortora G. (2007) Recovering Traceability Links in Software Artefact Management Systems using Information Retrieval Methods. ACM Transactions on Software Engineering and Methodology (TOSEM) **16**(4).

Deissenboeck F. and Pizka M. (2005) Concise and Consistent Naming. 13th IEEE International Workshop on Program Comprehension (IWPC'05), St. Louis, Missouri, USA, 97-106.

Deissenboeck F. and Pizka M. (2006) Concise and Consistent Naming. Software Quality Journal **14**(3): 261-282

Di Lucca G. A., Di Penta M. and Gradara S. (2002) An Approach to Classify Software Maintenance Requests. IEEE International Conference on Software Maintenance (ICSM'02), Montréal, Québec, Canada, 93-102.

Dit B., Revelle M., Gethers M. and Poshyvanyk D. (2012a) Feature Location in Source Code: A Taxonomy and Survey. Journal of Software Maintenance and Evolution: Research and Practice (JSME) **doi: 10.1002/smr.567**.

Dit B., Revelle M. and Poshyvanyk D. (2012b) Integrating Information Retrieval, Execution and Link Analysis Algorithms to Improve Feature Location in Software. Empirical Software Engineering.

Eaddy M., Aho A. V., Antoniol G. and Guéhéneuc Y. G. (2008) CERBERUS: Tracing Requirements to Source Code Using Information Retrieval, Dynamic Analysis, and Program Analysis. 16th IEEE International Conference on Program Comprehension (ICPC'08), Amsterdam, The Netherlands, 53-62.

Fluri B., Gall H. and Pinzger M. (2005) Fine-Grained Analysis of Change Couplings. 5th International Workshop on Source Code Analysis and Manipulation (SCAM'05) Budapest, Hungary, IEEE Computer Society: Washington, DC, USA, 66-74.

Gall H., Hajek, K., Jazayeri, M. (1998) Detection of Logical Coupling Based on Product Release History. Proceedings of the International Conference on Software Maintenance 1998 (ICSM'98), 190 - 198.

Gall H., Jazayeri, M., Krajewski, J. (2003) CVS Release History Data for Detecting Logical Couplings. Sixth International Workshop on Principles of Software Evolution (IWPSE'03): 13 - 23.

Gallagher K. and Lyle J. (1991) Using Program Slicing in Software Maintenance. Transactions on Software Engineering **17**(8): 751-762.

Gethers M., Dit B., Kagdi H. and Poshyvanyk D. (2012) Integrated Impact Analysis for Managing Software Changes. 34th IEEE/ACM International Conference on Software Engineering (ICSE'12), Zurich, Switzerland, to appear 10 pages.

Gethers M., Oliveto R., Poshyvanyk D. and De Lucia A. (2011) On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Link Recovery. 27th IEEE

International Conference on Software Maintenance (ICSM'11), Williamsburg, Virginia, USA, 133-142.
Gethers M. and Poshyvanyk D. (2010) Using Relational Topic Models to Capture Coupling among Classes in Object-Oriented Software Systems. 26th IEEE International Conference on Software Maintenance (ICSM'10), Timişoara, Romania, 1-10.
Haiduc S. and Marcus A. (2008) On the Use of Domain Terms in Source Code. 16th IEEE International Conference on Program Comprehension (ICPC'08), Amsterdam, The Netherlands, 113-122.
Hayes J. H., Dekhtyar A. and Sundaram S. K. (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Transactions on Software Engineering **32**(1): 4-19.
Hill E., Pollock L. and Vijay-Shanker K. (2007) Exploring the Neighborhood with Dora to Expedite Software Maintenance. 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), 14-23.
Jeong G., Kim S. and Zimmermann T. (2009) Improving Bug Triage with Bug Tossing Graphs. 7th European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2009), Amsterdam, The Netherlands.
Kagdi H., Collard M. L. and Maletic J. I. (2007) A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. Journal of Software Maintenance and Evolution: Research and Practice (JSME) **19**(2): 77-131.
Kagdi H., Gethers M. and Poshyvanyk D. (2011) SE2 Model to Support Software Evolution. 27th IEEE International Conference on Software Maintenance (ICSM'11), Williamsburg, VA, 512-515.
Kagdi H., Gethers M., Poshyvanyk D. and Collard M. (2010) Blending Conceptual and Evolutionary Couplings to Support Change Impact Analysis in Source Code. 17th IEEE Working Conference on Reverse Engineering (WCRE'10), Beverly, Massachusetts, USA, 119-128.
Kagdi H., Gethers M., Poshyvanyk D. and Hammad M. (2012) Assigning Change Requests to Software Developers. Journal of Software Maintenance and Evolution: Research and Practice (JSME) **24**(1): 3–33.
Kagdi H. and Maletic J. I. (2007) Mining Evolutionary Dependencies from Web-Localization Repositories. Journal of Software Maintenance and Evolution: Research and Practice **19**(5): 315-337.
Kagdi H., Maletic J. I. and Sharif B. (2007) Mining Software Repositories for Traceability Links. 15th IEEE International Conference on Program Comprehension (ICPC'07), Banff, Alberta, Canada, 145-154.
Kagdi H. and Poshyvanyk D. (2009) Who Can Help Me with this Change Request? 17th IEEE International Conference on Program Comprehension (ICPC'09), Vancouver, British Columbia, Canada, 273-277.
Kagdi H., Yusuf S. and Maletic J. I. (2006) Mining Sequences of Changed-files from Version Histories. 3rd International Workshop on Mining Software Repositories (MSR'06) Shanghai, China, ACM Press: New York NY, 47-53.
Kosara R., Healey C. G., Interrante V., Laidlaw D. H. and Ware C. (2003) Visualization viewpoints. Computer Graphics and Applications **23**(4): 20-25.
Law J. and Rothermel G. (2003) Whole Program Path-Based Dynamic Impact Analysis. 25th International Conference on Software Engineering, Portland, Oregon, 308-318.
Lehman M. M. and Belady L. A. (1985) Program Evolution: Processes of Software Change, Academic Press Professional, Inc.
Liu D., Marcus A., Poshyvanyk D. and Rajlich V. (2007) Feature Location via Information Retrieval based Filtering of a Single Scenario Execution Trace. 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), Atlanta, Georgia, 234-243.
Maletic J. I. and Collard M. L. (2004) Supporting Source Code Difference Analysis. 20th IEEE International Conference on Software Maintenance (ICSM'04), Chicago, Illinois, IEEE Computer Society: Los Alamitos CA, 210-219.
Mens T. (2002) A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering **28**(5): 449-462.
Moonen L. (2002) Lightweight Impact Analysis using Island Grammars. 10th International Workshop on Program Comprehension (IWPC'02), Paris, France, 219-228.

Oliveto R., Gethers M., Poshyvanyk D. and De Lucia A. (2010) On the Equivalence of Information Retrieval Methods for Automated Traceability Link Recovery. 18th IEEE International Conference on Program Comprehension (ICPC'10), Braga, Portugal, 68-71.

Orso A., Apiwattanapong T., Law J., Rothermel G. and Harrold M. J. (2004) An empirical comparison of dynamic impact analysis algorithms. IEEE/ACM International Conference on Software Engineering (ICSE'04), 776-786.

Petrenko M. and Rajlich V. (2009) Variable Granularity for Improving Precision of Impact Analysis. 17th IEEE International Conference on Program Comprehension (ICPC'09), Vancouver, BC, Canada, 10-19

Poshyvanyk D., Gethers M. and Marcus A. (2012) Concept Location using Formal Concept Analysis and Information Retrieval. ACM Transactions on Software Engineering and Methodology.

Poshyvanyk D., Guéhéneuc Y. G., Marcus A., Antoniol G. and Rajlich V. (2007) Feature Location using Probabilistic Ranking of Methods based on Execution Scenarios and Information Retrieval. IEEE Transactions on Software Engineering **33**(6): 420-432.

Poshyvanyk D. and Marcus A. (2006) The Conceptual Coupling Metrics for Object-Oriented Systems. 22nd IEEE International Conference on Software Maintenance (ICSM'06), Philadelphia, PA, USA, 469 - 478.

Poshyvanyk D., Marcus A., Ferenc R. and Gyimóthy T. (2009) Using Information Retrieval based Coupling Measures for Impact Analysis. Empirical Software Engineering **14**(1): 5-32.

Poshyvanyk D. and Marcus D. (2007) Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code. 15th IEEE International Conference on Program Comprehension (ICPC'07), Banff, Alberta, Canada, 37-48.

Queille J.-P., Voidrot J.-F., Wilde N., Munro M. and (1994) The Impact Analysis Task in Software Maintenance: A Model and a Case Study. International Conference on Software Maintenance, 234 - 242.

Raghavan S., Rohana R., Leon D., Podgurski A. and Augustine V. (2004) Dex: A Semantic-Graph Differencing Tool for Studying Changes in Large Code Bases. 20th IEEE International Conference on Software Maintenance (ICSM'04), Chicago, Illinois, IEEE Computer Society: Los Alamitos CA, 188-197.

Rajlich V. and Bennett K. (2000) A Staged Model for the Software Lifecycle. Computer **33**(7): 66-71.

Ren X., Shah F., Tip F., Ryder B. G. and Chesley O. (2004) Chianti: a Tool for Change Impact Analysis of Java Programs. 19th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications(OOPSLA '04), Vancouver, BC, Canada, 432-448.

Revelle M., Dit B. and Poshyvanyk D. (2010) Using Data Fusion and Web Mining to Support Feature Location in Software. 18th IEEE International Conference on Program Comprehension (ICPC'10), Braga, Portugal, 14-23.

Revelle M., Gethers M. and Poshyvanyk D. (2011) Using Structural and Textual Information to Capture Feature Coupling in Object-Oriented Software. Empirical Software Engineering **16**(6): 773-811.

Revelle M. and Poshyvanyk D. (2009) An Exploratory Study on Assessing Feature Location Techniques. 17th IEEE International Conference on Program Comprehension (ICPC'09), Vancouver, British Columbia, Canada, 218-222.

Robillard M. (2005) Automatic Generation of Suggestions for Program Investigation. Joint European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, Lisbon, Portugal, 11 - 20

Robillard M. P. (2008) Topology Analysis of Software Dependencies. ACM Transactions on Software Engineering and Methodology **17**(4): 1-36.

Rountev A., Milanova A. and Ryder B., G. (2001) Points-to analysis for Java using annotated constraints. Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'01), Tampa Bay, FL, USA, 43-55.

Runeson P., Alexandersson M. and Nyholm O. (2007) Detection of Duplicate Defect Reports Using Natural Language Processing. 29th IEEE/ACM International Conference on Software Engineering (ICSE'07), Minneapolis, MN, 499-510.

Saul M. Z., Filkov V., Devanbu P. and Bird C. (2007) Recommending Random Walks. 11th European Software Engineering Conference held jointly with 15th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE'07), Dubrovnik, Croatia, 15-24.

Tonella P. (2003) Using a Concept Lattice of Decomposition Slices for Program Understanding and Impact Analysis. IEEE Transactions on Software Engineering **29**(6): 495-509.

Rajlich V. (1997) A Model for Change Propagation Based on Graph Rewriting. International Conference on Software Maintenance (ICSM '97), Bari, ITALY, IEEE, 84-91.

Wang X., Zhang L., Xie T., Anvik J. and Sun J. (2008) An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information. 30th International Conference on Software Engineering (ICSE'08), Leipzig, Germany, 461-470.

Weiss C., Premraj R., Zimmermann T. and Zeller A. (2007) How Long Will It Take to Fix This Bug? 4th IEEE International Workshop on Mining Software Repositories (MSR'07), Minneapolis, MN, 1-8.

Wilkie F. G. and Kitchenham B. A. (2000) Coupling measures and change ripples in C++ application software. The Journal of Systems and Software **52**: 157-164.

Ying A. T. T., Murphy G. C., Ng R. and Chu-Carroll M. C. (2004) Predicting Source Code Changes by Mining Change History. IEEE Transactions on Software Engineering **30**(9): 574-586.

Yu Z. and Rajlich V. (2001) Hidden Dependencies in Program Comprehension and Change Propagation. 9th IEEE International Workshop on Program Comprehension (IWPC'01), Toronto, Canada, 293-299.

Zaki M. J. (2001) SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning **42**(1-2): 31 - 60.

Zimmermann T., Zeller A., Weißgerber P. and Diehl S. (2005) Mining Version Histories to Guide Software Changes. IEEE Transactions on Software Engineering **31**(6): 429-445.