Detecting Evolutionary Coupling Using Transitive Association Rules

Md. Anaytul Islam
Computer Science and Engg. Discipline
Khulna University, Bangladesh
anaytularpon@gmail.com

Md. Moksedul Islam
Computer Science and Engg. Discipline
Khulna University, Bangladesh
moksedulislammishuk@gmail.com

Manishankar Mondal
Department of Computer Science
University of Saskatchewan, Canada
mshankar.mondal@usask.ca

Banani Roy
Department of Computer Science
University of Saskatchewan, Canada
banani.roy@usask.ca

Chanchal K. Roy
Department of Computer Science
University of Saskatchewan, Canada
chanchal.roy@usask.ca

Kevin A. Schneider
Department of Computer Science
University of Saskatchewan, Canada
kevin.schneider@usask.ca

***Abstract*—If two or more program entities (such as files, classes, methods) co-change (i.e., change together) frequently during software evolution, then it is likely that these two entities are coupled (i.e., the entities are related). Such a coupling is termed as evolutionary coupling in the literature. The concept of traditional evolutionary coupling restricts us to assume coupling among only those entities that changed together in the past. The entities that did not co-change in the past might also have coupling. However, such couplings can not be retrieved using the current concept of detecting evolutionary coupling in the literature. In this paper, we investigate whether we can detect such couplings by applying transitive rules on the evolutionary couplings detected using the traditional mechanism. We call these couplings that we detect using our proposed mechanism as *transitive evolutionary couplings*. According to our research on thousands of revisions of four subject systems, *transitive evolutionary couplings* combined with the traditional ones provide us with 13.96% higher recall and 5.56% higher precision in detecting future co-change candidates when compared with a state-of-the-art technique.**

I. Introduction

A software system evolves by experiencing changes in its constituent entities (such as files, classes, methods). Making changes in a software entity can often be challenging, because changes to that entity might require corresponding changes to some other related entities in order to ensure consistency of the software system. Researchers have investigated to discover such change correspondences, also known as change couplings, among software entities through analyzing their evolutionary history. These change couplings are called evolutionary coupling or logical coupling in the literature [38].

If two or more program entities change together in a commit operation, we call that the entities have co-changed. Frequent co-change of a set of program entities during software evolution is a meaningful phenomenon, because it indicates the existence of change coupling or evolutionary coupling among the co-changing or co-evolving entities. If two or more entities frequently co-changed in the past, then it is likely that the entities are related and a change in one of the entities in future will require corresponding changes to the remaining entities as well. In other words, it is likely that the entities will again co-change in a future commit operation. While the primary reason behind detecting evolutionary coupling is to predict future co-change candidates for program entities, such coupling has also been used for bug prediction [2], [32], detecting cross cutting concerns [11], and finding important clones for refactoring and tracking [28], [29]. We investigate predicting future co-change candidates in our research. Better prediction of evolutionary coupling can assist us in better prediction of co-change candidates.

A number of studies [31], [21], [22], [38], [15], [13], [18] have investigated predicting future co-change candidates for program entities through analyzing evolutionary coupling of entities. Evolutionary coupling has been realized using association rules [38]. An association rule consisting of a number of program entities has two measures: *Support* and *Confidence*. These two measures are based on the co-change frequency of the constituent program entities and indicate the strength of coupling among the entities. Higher values of *Support* and *Confidence* indicate stronger coupling among entities. However, the main drawback of association rule mining technique dealing with support and confidence is that it can only help us realize evolutionary coupling among entities that changed together (co-changed) in the past. If two entities did not co-change in the past, we cannot form any association rule from them and thus, cannot realize coupling between them. However, such entities (the entities that did not co-change before) might also have coupling and changing one entity in future might require corresponding changes to the other entity. Focusing on this drawback of the classical association rules, we propose a technique for detecting coupling among program entities that did not co-change in the past.

Our proposed technique is also dependent on association rule mining technique. We first detect association rules by automatically examining the evolutionary history of our subject systems and then apply transitivity on these rules to derive new rules which we call *Transitive Association Rules*. Transitive association rules can realize coupling among program entities which did not co-change before. We call such couplings the *Transitive Evolutionary Coupling*. Fig. 1 explains our concept of transitive evolutionary coupling with a simple example.

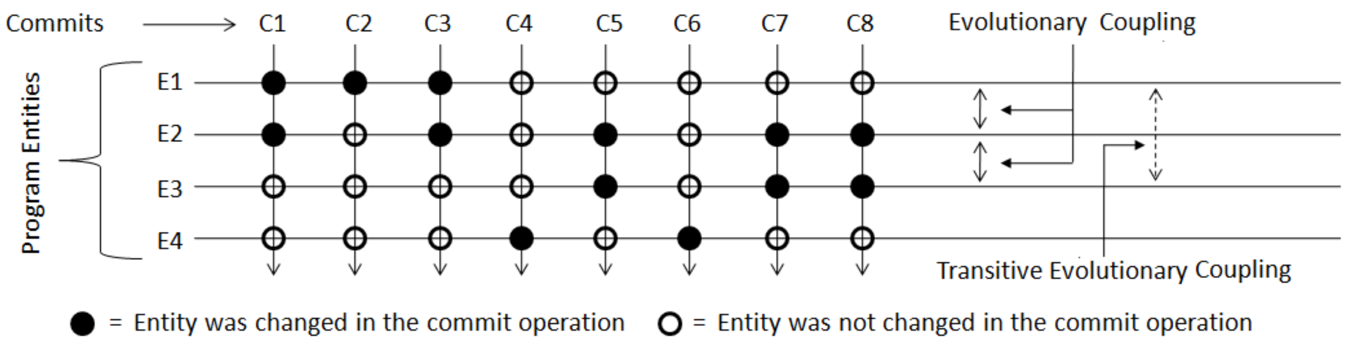
Commits
C1
C2
C3
C4
C5
C6
C7
C8
Evolutionary Coupling
Program Entities
E1
E2
E3
E4
Transitive Evolutionary Coupling
● = Entity was changed in the commit operation
○ = Entity was not changed in the commit operation

Fig. 1. Explaining transitive evolutionary coupling

Fig. 1 shows the evolution of four program entities E1 to E4 of a subject system through eight commit operations C1 to C8. From the evolution history it is clear that the entities E1 and E2 have evolutionary coupling, because they co-changed (i.e., changed together) in two commit operations: C1 and C3. The entities E2 and E3 also exhibit evolutionary coupling, because they co-changed in the commit operations: C5, C7, and C8. However, the entities E1 and E3 never co-changed and thus, according to the current consideration of evolutionary coupling, E1 and E3 do not have such a coupling between them. We cannot derive any association rules considering these two entities (E1 and E3). However, our idea is to realize a coupling even between E1 and E3, because both of them have coupling with a single entity, E2. This coupling that we propose to realize between E1 and E3 has been denoted as *transitive evolutionary coupling* in this paper. Section III defines *transitive association rules* and describes how we use it for realizing as well as quantifying such transitive evolutionary couplings. From Fig. 1 it is clear that the entity E4 does not exhibit evolutionary coupling with any of the other entities.

While most of the existing studies investigated evolutionary coupling considering file level granularity, we conduct our study considering a finer granularity, method level granularity. Investigation considering a finer granularity is important, because it can help us understand which fine grained entities in the code-base are responsible for frequent co-change of files. We answer two research questions listed in Table I through our experiments. Our research on thousands of revisions from four subject systems written in two different programming languages show that:

- Transitive association rules combined with the regular association rules outperform regular rules in predicting future co-change of program entities (methods in our experiment).
- A combination of regular and transitive association rules outperforms a state of the art technique called TARMAQ [31] in predicting co-change candidates for methods. Our prediction mechanism dealing with both regular and transitive rules achieves overall 13.96% higher recall and 5.56% higher precision when compared with TARMAQ that deals only with regular association rules.

The rest of the paper is organized as follows. Section II defines association rule and its related measures, Section III discusses on transitive association rules, Section IV describes the experiment setup and steps, Sections V and VI elaborate on our experiments and discuss our findings, Section VII discusses the threats to validity, Section VIII describes the related research, and finally, Section IX concludes the paper by mentioning future research possibilities.

TABLE I. RESEARCH QUESTIONS

Serial	Research Questions (RQs)
RQ1	Can transitive association rules help us in better prediction of co-change compared to regular rules?
RQ2	Can transitive association rules help us in better prediction of future co-change candidates for methods?

II. ASSOCIATION RULES

Association rule was first introduced by Agrawal et al. [1] for the purpose of identifying frequent item sets in large transactions. This concept was later used in the field of software engineering for identifying coupling among program entities. The coupling that we realize among program entities by mining association rules is termed as evolutionary coupling or logical coupling [9], [38]. An association rule has two measures: *Support* and *Confidence*. In the following paragraphs we define association rules and the two related measures.

A. Association Rule

An association rule [1] is formally defined as an expression of the form $X => Y$. Here, X is known as the antecedent and Y is the consequent. Each of X and Y is a set of one or more program entities. In the context of software engineering, such an association rule implies that if X gets changed in a particular commit operation, Y also has the tendency of getting changed in that commit operation.

B. Support and confidence of association rule

According to Zimmermann et al. [38], *support is the number of commits in which an entity or a group of entities changed together*. Let us consider an example of two program entities E_1 and E_2. These entities can be files, classes, or methods. If E_1 and E_2 have ever changed together (co-changed), we can assume two association rules, $E_1 => E_2$ and $E_2 => E_1$ from these. Suppose, E_1 was changed in four commit operation: 2, 5, 6, and 10 and E_2 was changed in six commits: 4, 6, 7, 8, 10, and 13. Thus, according to the definition, *support(E_1) = 4* and *support(E_2) = 6*. However, *support(E_1, E_2) = 2*, because E_1 and E_2 co-changed (changed together) in two commits: 6 and 10. Support of a rule is determined in the following way.

$$support(X => Y) = support(X, Y) \quad (1)$$

Here, (X, Y) is the union of X and Y, and so $support(X => Y) = support(Y => X)$. For the above example of two entities, $support(E_1 => E_2) = support(E_2 => E_1) = support(E_1, E_2) = 2$.

Confidence of an association rule, $X => Y$, determines the conditional probability that Y will change in a commit operation provided that X changed in that commit operation. We determine the confidence of the association rule, $X => Y$, according to the following equation.

$$confidence(X => Y) = support(X, Y)/support(X) \quad (2)$$

If we again consider the same example of two entities E1 and E2, then *confidence (E_1 => E_2) = support(E_1, E_2) / support(E_1) = 2 / 4 = 0.5* and *confidence(E_2 => E_1) = 2 / 6 = 0.33*. In our experiment we consider those association rules where each of X and Y consist of a single method. Such an association rule can be expressed as $x => y$ where x and y are two different methods.

III. Transitive Association Rules

We explain transitive association rules and their strengths by considering Fig. 1 as our example. According to the evolution history of the program entities E1, E2, and E3 in Fig. 1, E1 and E2 have evolutionary coupling. The entities E2 and E3 exhibit evolutionary coupling as well. The entities E1 and E3 do not exhibit evolutionary coupling according to the current concept. We propose to realize a coupling between E1 and E3, because both are coupled with the same entity E2. We call this coupling between E1 and E3 the *Transitive Evolutionary Coupling*. Before explaining how we derive *Transitive Association Rules* between E1 and E3, we show and quantify the regular association rules between E1 and E2, and also, between E2 and E3.

A. Regular association rules

We will use the symbol, =>, to denote regular association rules among program entities. From the evolution of two entities E1 and E2 in Fig. 1, we can derive two association rules, E1 => E2 and E2 => E1. Similarly, from the two entities E2 and E3 we can form two regular association rules, E2 => E3 and E3 => E2. According to the evolution in Fig. 1 and the definitions of *Support* and *Confidence* in Section II, we determine the supports and confidences of these association rules and present those in Table II.

TABLE II. Regular association rules from Fig. 1

Support and Confidence for **E1 => E2** and **E2 => E1**
Support (E1 => E2) = Support (E1, E2) = 2
Support (E2 => E1) = Support (E1, E2) = 2
Confidence (E1 => E2) = Support (E1 => E2) / Support (E1) = 2/3 = 0.66
Confidence (E2 => E1) = Support (E2 => E1) / Support (E2) = 2/5 = 0.4
Support and Confidence for **E2 => E3** and **E3 => E2**
Support (E2 => E3) = Support (E2, E3) = 3
Support (E3 => E2) = Support (E2, E3) = 3
Confidence (E2 => E3) = Support (E2 => E3) / Support (E2) = 3/5 = 0.6
Confidence (E3 => E2) = Support (E3 => E2) / Support (E3) = 3/3 = 1

B. Transitive association rules

We will use the symbol, ==>, to denote transitive association rules. For realizing the transitive evolutionary coupling between the entities E1 and E3, we define two transitive association rules: E1 ==> E3 and E3 ==> E1. We cannot calculate the support values for such rules, because the constituent entities never co-changed. However, we calculate their confidence values by using confidences of regular association rules in the following way. Confidence (E1 ==> E3) = Confidence (E1 => E2) × Confidence(E2 => E3). Similarly, Confidence (E3 ==> E1) = Confidence (E3 => E2) × Confidence(E2 => E1). Thus,

Confidence (E1 ==> E3) = 0.66 × 0.6 = 0.396

Confidence (E3 ==> E1) = 1.00 × 0.4 = 0.4

We can easily understand that the confidence value of a transitive association rule will be at least as small as the lowest value of its constituent confidences. It can never be greater than any of its constituent confidence values.

C. Rationale behind the way of calculating confidence for transitive association rules

Let us first consider the transitive association rule E1 ==> E3. The regular association rule E1 => E2 with a confidence of 0.66 indicates that if E1 gets changed, then E2 might also be changed with a probability of 0.66. Similarly, the regular association rule E2 => E3 with confidence 0.6 implies that if E2 is changed, then E3 might also be changed with a probability of 0.6. Thus, according to probability theory, changes in E1 might also trigger changes in E3 and the probability of occurrence of this phenomenon is the multiplication of the two probabilities 0.66 and 0.6. Thus, our calculation for determining confidence for transitive association rules is reasonable.

In our example, we have derived a transitive association rule from two regular association rules. However, it is also possible to derive a new transitive association rule from two existing transitive rules or from one transitive and one regular rule. In our research we detect transitive association rules in order to realize couplings among entities that did not co-change in the past. We use these transitive rules with regular rules and investigate whether transitive association rules can help us in better prediction of co-change candidates for program entities.

D. A real-world example where transitive association rules could help us in better prediction of co-change candidates

Table III shows a real-world example from our subject system called MonoOSC where we could use transitive evolutionary coupling for better prediction of co-change candidates. From Table III we see that the two methods *GetKey* and *InitializeComponent* changed together in the third commit operation (i.e., C3). The method ids: M6 and M13 were

TABLE III. Real Life Example of Transitivity

Commit	C3		C5		C6	
Method ID	**M6**	**M13**	**M13**	**M28**	**M6**	**M28**
Method Name	GetKey	InitializeComponent	InitializeComponent	PutProjectMeta	GetKey	PutProjectMeta
Signature	StringBuilderGetKey()	voidInitializeCompone-nt()	voidInitializeCompone-nt()	StringBuilderPutProject Meta(bool,string,string)	StringBuilderGetKey()	StringBuilderPutProject Meta(bool,string,string)
File path	MonoOSC/ MonoOSCFrame-work/Class/Functions/ Sources/GetPubkey.cs	MonoOSC/ MonoOSC/Forms/Main Form.Designer.cs	MonoOSC/ MonoOSC/Forms/Main Form.Designer.cs	MonoOSC/ MonoOSCFrame-work/Class/Functions/ Sources/ PutSource-ProjectMeta.cs	MonoOSC/ MonoOSCFrame-work/Class/Functions/ Sources/GetPubkey.cs	MonoOSC/ MonoOSCFrame-work/Class/Functions/ Sources/ PutSource-ProjectMeta.cs
Package name	Resource Manager	Resource Manager	Resource Manager	Resource Manager	Resource Manager	Resource Manager
Class name	MonoOSCFramework. Functions.Sources. GetPubkey	MonoOSC.MainForm	MonoOSC.MainForm	MonoOSCFramework. Functions.Sources. PutSourceProjec...	MonoOSCFramework. Functions.Sources. GetPubkey	MonoOSCFramework. Functions.Sources. PutSourceProjec...
Start line	30	29	29	32	42	48
End line	33	186	187	38	47	54

TABLE IV. Subject Systems

System	Lang.	Domain	LOC	Revisions
Ctags	C	Code Def. Generator	33,270	774
MonoOSC	C#	Formats and Protocols	18,991	355
BRL-CAD	C	Solid Modeling CAD	52,313	735
Camellia	C	Multimedia	85,015	207

generated by our prototype tool. The table also shows that the method M13 also co-changed with another method M28 (*PutProjectMeta*) in the fifth commit operation (C5). From commit C3 we can realize a regular evolutionary coupling between methods: M6 and M13. Similarly, from C5 we can again realize evolutionary coupling between methods M13 and M28. Now, according to our proposed idea, we can realize a transitive evolutionary coupling between methods M6 and M28. We see that these two methods have actually co-changed in commit operation C6. This example makes us understand that after commit C5, when a programmer attempted to change any one of these two methods (M6 and M28), we could suggest the other one as the possible co-change candidate. None of the existing techniques [22], [38], [15], [13], [18] including the state of the art [31] can provide such suggestions. Table III shows each method with it's method name, signature, file path, package name, class name, start line, and end line. These were automatically generated by our prototype tool that we implemented for this research.

We also looked at the changes that occurred in the methods M6 (*GetKey*) and M28 (*PutProjectMeta*) in commit C6 in order to check whether the changes are related. We found that the changes are actually related. In each of the methods, a variable called 'FullUserName' was replaced by another variable named 'VarGlobal.PrefixUserName'.

IV. Experiment setup and steps

A. Experiment setup

We conduct our experiment on four subject systems listed in Table IV. We downloaded all the revisions (as mentioned in Table IV) of these systems from an on-line SVN repository called SourceForge [35]. We see that the systems are of different sizes. They belong to different application domains and their revision history lengths are also different. The systems are written in two different programming languages. We selected the systems in this way because we wanted to avoid possible bias related to subject system nature in our experiment. We also conduct our experiment considering method level granularity. The existing studies on evolutionary coupling have mostly considered file level granularity. Thus, our study was done considering a finer granularity compared to most of the existing studies on evolutionary coupling.

B. Experiment steps

We automatically perform the following experimental steps for each of our subject systems.

- We detect methods from each of the revisions of the subject system using CTAGS [14].
- We then detect method genealogies considering all the methods in all revisions using the approach followed by Lozano and Wermelinger [24]. A genealogy of a method consists of the snap-shots (instances) of that method from the revisions where it was alive. Each such revision contains one snap-shot (instance) of the method. By analyzing the genealogy of a method, we can determine how it was changed during system evolution.
- Detecting changes between every two consecutive revisions using UNIX diff.
- Mapping the changes to the already detected methods of each revision by using starting and ending line numbers of the methods and changes.
- Detecting regular as well as transitive association rules of methods by analyzing method co-change history.
- Analyzing the impact of using transitive association rules in predicting co-change candidates for methods.

In the following subsection we describe how we detect association rules of methods by automatically examining the entire evolution history of each of our subject systems.

C. Detecting regular association rules among methods

We detect and analyze binary association rules of methods in our experiment. As an example of a binary association rule we can consider the rule $m1 => m2$. We see that this rule consists of only two methods. One method (m1) is the antecedent and the other method (m2) is the consequent. We detect binary association rules, because these can help us predict co-change candidates for unseen queries. We explain this in Section VI. For detecting binary association rules we automatically examine the commit operations of a subject system starting from the very beginning one (as mentioned in Table IV). When examining a particular commit operation, we first identify which methods changed together in this commit. Let us assume that M >= 2 methods changed together in a commit operation. We make all possible pairs from these methods. For example, if 6 methods changed together in a commit operation, then we make 15 method pairs in total. Let us assume that (m1, m2) is such a pair. We make two association rules: m1 => m2, and m2 => m1 from such a pair. As we examine the commit operations sequentially, we keep track of which method changed how many times, and also, which methods making a pair co-changed how many times. Using the change as well as co-change counts of m1 and m2, we determine the supports and confidences of the two association rules: m1 => m2, and m2 => m1. Section II describes the way of calculating support and confidence for an association rule. In this way, we detect all possible binary association rules as well as their supports and confidences by analyzing the entire evolution history of each subject system.

In Section III, we described how we detect transitive association rules from regular association rules. In the following sections we describe our experiment on transitive association rules, and answer the research questions in Table I by analyzing experiment results.

V. Predicting co-change among program entities using transitive association rules

We apply our implementation on each of our subject systems and answer the two research questions listed in Table I through analyzing our experiment results. This section answer the first research question RQ 1.

RQ 1: *Can transitive association rules help us in better prediction of co-change compared to regular rules?*

Rationale behind answering RQ 1. The fundamental concept of evolutionary coupling is that *if two entities co-changed (changed together) in the past, they might again co-change in future*. Association rules and the related measures *support* and *confidence* help us determine this future co-change possibility. We extend this idea of evolutionary coupling by proposing that *two entities that did not co-change in the past can also have a possibility of co-changing in future*. By introducing the concept of *transitive association rules* we identify such entities (i.e., the entities that did not co-change at past but have a possibility of co-changing in future) and determine their probabilities of co-changing in future. Our experiment regarding RQ 1 determines whether transitive association rules can really help us identify such entities. Identification of such entities will add a new dimension to the concept of evolutionary coupling. We conduct our experiment in the following way.

Investigation procedure. For each of our subject systems, we divide the entire history of evolution into two halves. For example, if a system's evolution history consists of C commit operations in total, then we divide these commits into two halves where each half contains $C/2$ consecutive commit operations. Now, by considering the first half of the commits, we determine the regular as well as the transitive association rules. As we discussed in Section IV, our association rules (both regular and transitive) are binary ones. From these association rules, we determine pairs of methods where the two entities in the pair have evolutionary coupling. Let us assume that we have obtained a method pair, (M1, M2), from the first half of the commits of a subject system. We now analyze the second half of the commits of the system and determine pairs of methods such that the two methods making a pair co-changed in at least one of the commit operations in the second half. If the method pair (M1, M2) that we obtained from the first half of the commits also appear in the set of method pairs obtained from the second half, then we understand that we could correctly predict a coupling between M1 and M2 by analyzing the commits in the first half. We determine the following three sets by examining the method pairs obtained from the two halves of commits of a subject system.

TABLE V. Statistics regarding regular and transitive association rules

	Ctags	MonoOSC	BRL-CAD	Camellia
Number of method pairs in the set RS (Regular Set)	1804	3806	11379	26323
Number of method pairs in the set TS (Transitive Set)	11407	22990	21758	4058
Number of method pairs in the set CS (Check Set)	2331	5915	9556	452
$CS \cap RS$	77	457	1062	73
$CS \cap TS$	136	849	1270	1
$CS \cap (TS \cup RS)$	213	1306	2332	74

- **RS** (Regular Set): This set contains method pairs obtained by applying regular association rules on the first half of the commit operations. For each pair in this set, the regular association rules predict the presence of evolutionary coupling between the two constituent methods.
- **TS** (Transitive Set): This set contains method pairs obtained by applying transitive association rules on the first half of the commits. For each pair in this set, the transitive association rules predict the presence of a coupling (i.e., transitive evolutionary coupling) between the two constituent methods. The existing techniques dealing with evolutionary coupling cannot predict such couplings among program entities.
- **CS** (Check Set): This set contains method pairs that we obtained by examining the co-changes of methods in the second half of the commits. If a pair that we obtained from **RS** or **TS** is also present in **CS**, then we understand that the regular association rules or the transitive association rules could correctly predict a coupling between the constituent methods in the pair.

From the definitions of the two sets **RS** and **TS**, we understand that these are disjoint. We show the values of $|RS|$, $|TS|$, and $|CS|$ in Table V for each of our subject systems. By using the three sets (RS, TS, and CS) we determine the following three percentages.

- **P1**: We first determine what percentage of co-changes in the second half can be predicted by applying regular association rules in the first half. We call this percentage **P1** and calculate it in the following way.

$$P1 = \frac{|CS \cap RS| \times 100}{|CS|} \tag{3}$$

- **P2**: P2 is the percentage of co-changes in the second half that can be predicted by applying transitive association rules in the first half. We determine this percentage in the following way.

$$P2 = \frac{|CS \cap TS| \times 100}{|CS|} \tag{4}$$

- **P3**: P3 is the percentage of co-changes in the second half that can be predicted by applying both regular and transitive association rules in the first half. We determine this percentage in the following way.

$$P3 = \frac{|CS \cap (TS \cup RS)| \times 100}{|CS|} \tag{5}$$

We determine these three percentages for each of our subject systems and plot the percentages in the graph of Fig. 2. From Fig. 2 this is clear that the percentage P2 is higher than P1 for all subject systems except Camellia. For Camellia, the intersection of the two sets CS and TS provides only one method pair (c.f., Table V), and it makes the value of P2 to be near zero. P3 is always the highest. In other words, the possibility that a co-change in the second half of commits will be predicted by the regular association rules in the first half is most of the time lower than the possibility that a co-change in the second half will be predicted by the transitive association rules in the first half. When we combine regular association rules with the transitive ones, we achieve the highest possibility of predicting co-changes in the second half of commit operations.

Answer to RQ 1: According to our investigation results and analysis, *transitive association rules can often help us in better prediction of co-change compared to regular association rules*. Moreover, a combination of regular and transitive association rules perform even better in predicting co-changes.

VI. Predicting future co-change candidates using transitive association rules

This section answers our second research question (RQ 2).

RQ 2: *Can transitive association rules help us in better prediction of future co-change candidates for methods?*

Rationale behind answering RQ 2. The primary purpose of detecting evolutionary coupling is to predict co-change candidates for a program entity that a programmer is going to change. In this section we will investigate how accurately we can predict co-change candidates for methods using a combination of regular and transitive association rules. We also compare our prediction accuracy with that of a recently introduced technique called TARMAQ [31] that deals with the regular association rules only. For the purpose of our comparison, we have implemented TARMAQ according to the algorithm proposed by Rolfsnes et al. [31]. We could use the existing implementation of TARMAQ, however, the existing implementation suggests co-change candidates considering file level granularity. In our research, we investigate a finer granularity, method level granularity. Investigation considering method level granularity is tricky, because we need to detect

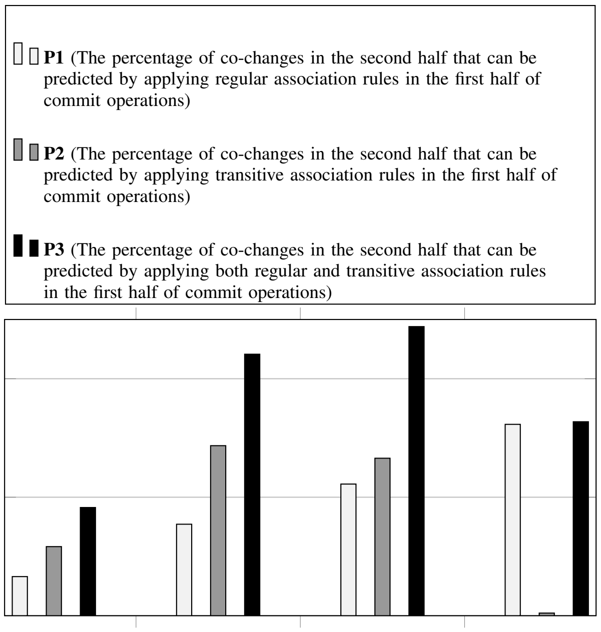

Fig. 2. Comparing the percentages of co-changes predicted by regular and transitive association rules

methods from each revision of a subject system and then detect method genealogies. We finally decided to implement TARMAQ considering method level granularity so that we can use it for comparing with our proposed mechanism. Two other techniques called ROSE [38] and SVD [36] analyze evolutionary coupling for suggesting co-change candidates by using regular association rules. However, Rolfsnes [31] showed that TARMAQ performs better than these two existing techniques in suggesting co-change candidates. This is the reason why we select TARMAQ for our comparison. In the following subsections we describe our investigation and analysis for answering RQ 2.

A. Prediction mechanism

Let us assume that a programmer is going to make changes to a particular method in the most recent revision (i.e., the working revision) of a subject system. We want to predict co-change candidates for this target method. For suggesting co-change candidates, we detect and analyze regular and transitive association rules from all the previous commit operations. We select those association rules where the target method is the antecedent. The union of the consequents of these rules is the set of suggested co-change candidates for the target method. We order these suggested co-change candidates on the basis of the confidence values of the selected association rules. Previous studies [26], [27] show that association rules with a very low support value (such as a support of 1 or 2) can often be important. This is the reason why we order the suggested co-change candidates using confidence. We can also predict co-change candidates for a set of two or more target methods. In this case, we consider each of the methods in the target set individually and select regular and transitive association rules from the previous commits such that each rule has one of these target methods as the antecedent. The union of the consequents of these selected rules ordered according to confidence values are the co-change candidates for the target method set. We should again note that we detect association rules as the binary ones where each of the antecedents and consequents is a single method. In the following subsection we will describe how we evaluate our prediction mechanism that uses regular and transitive association rules.

B. Handling unseen queries

TARMAQ [31] is capable of predicting co-change candidates for unseen queries. For an unseen query, TARMAQ finds co-change candidates for the seen parts using regular association rules, and provide these candidates as the suggestions for the whole query. Our prediction mechanism described in Section VI-A can also predict co-change candidates for unseen queries, because we work with binary association rules. For a query (i.e., a target set) that consists of two or more methods, we search binary association rules considering each of the methods in the query where the method is the antecedent in the rule. From these binary association rules, we determine the consequents. The union of these consequents is considered as the result for the query.

C. Mechanism for evaluating prediction accuracy

We automatically examine and analyze the entire evolution history of a subject system for the purpose of evaluation. Let us assume that we are now examining a particular commit operation c of a subject system. The set of methods that co-changed in this commit operation is SM (Set of Methods). We further assume that SM consists of five methods: m1, m2, m3, m4, and m5. We determine all possible subsets of these methods where a subset can have at most $|SM|-1$ methods (4 in this example). We consider each of these subsets as a target set. For example a target set can consist of the methods: m1 and m2. Our goal is to determine how accurately we can predict co-change candidates for such a target set. We call this target set TS. Section VI-A describes how we predict co-change candidates for a target set of methods by analyzing the regular and transitive association rules from the previous commit operations. Now, as we automatically examine the commit operation c, we realize that the actually co-changed candidates for the target set TS are m3, m4, and m5. We define a set called ACC (Actually Co-changed Candidates) consisting of the three methods: m3, m4, and m5. Let us assume that the set of predicted (suggested) co-change candidates for the target set TS is PCC (Predicted Co-change Candidates) as obtained by our prediction mechanism. Now, we can determine the precision and recall (in percentage) in suggesting co-change candidates for TS in the following way.

$$Precision = \frac{|ACC \cap PCC| \times 100}{|PCC|} \tag{6}$$

$$Recall = \frac{|ACC \cap PCC| \times 100}{|ACC|} \tag{7}$$

From the equations we realize that the set $ACC \cap PCC$ contains those methods that are correctly predicted by our prediction mechanism. In this way we determine the precision and recall for every possible target sets from each of the commit operations in the entire period of evolution of a subject system. We then determine the average precision and recall.

D. Comparing our prediction mechanism with TARMAQ

We also apply TARMAQ on each of the subject systems and determine precision and recall for every possible target set from all the revisions as we have described in Section VI-C. We then determine the average precision and recall for each system. We should note that TARMAQ only deals with the regular association rules. We did not use transitive association rules when predicting co-change candidates using TARMAQ.

We compare the precisions and recalls obtained by TARMAQ that only uses regular association rules and by our prediction mechanism that uses both regular and transitive association rules. The graphs in Fig. 3 to 10 show the comparisons. We show the precision and recall of our mechanism for different confidence thresholds of the transitive association rules. In each graph, we see a variation of the precision and recall values computed by our mechanism. This variation occurs because of considering different thresholds of confidences for transitive association rules. For a particular confidence threshold, we did not consider any transitive rule having a confidence which is less than this threshold for detecting co-change candidates. As TARMAQ does not consider transitive association rules, there is no variation in the precision and recall values computed by it.

Comparison regarding recall. If we consider the graphs (Fig. 3, 4, 5, and 6) showing comparison for recall, we realize that our prediction mechanism provides the highest recall when the confidence threshold for the transitive association rules is the lowest. As the threshold increases, the recall of our prediction mechanism decreases. We also see that the recall of our prediction mechanism is always higher than that of TARMAQ. It is easy to realize that transitive association rules provide a considerable amount of true positives and it makes our prediction mechanism to achieve high recall values. From Fig. 4 and Fig. 6 we realize that the recall of our prediction mechanism can be two times of the recall value of TARMAQ at the lowest confidence threshold of transitive association rules.

Comparison regarding precision. By looking at the graphs in Fig. 7, 8, 9, and 10 we realize that for the lowest confidence threshold of transitive association rules, our prediction mechanism provides the lowest precision except for Ctags. For Ctags, we get the highest precision at the lowest confidence threshold of transitive association rules. For the remaining three subject systems (BRL-CAD, Camellia, and MonoOSC), as we gradually increase the confidence threshold, the precision increases. At the confidence threshold of 0.7, our prediction mechanism provides better precisions for Camellia, MonoOSC, and Ctags and also better recalls for all the subject systems.

We finally decide that for a confidence threshold of 0.7 for transitive association rules, our prediction mechanism provides better recalls (overall 13.96% higher recall according to our calculation considering all subject systems) and mostly better precisions (overall 5.56% higher precision according to all subject systems) when compared with TARMAQ. Fig. 11 and 12 show comparisons regarding recall and precision between our prediction mechanism and TARMAQ by considering a confidence threshold of 0.7 for transitive association rules. Fig. 11 shows that the recall of our prediction mechanism is always higher compared to TARMAQ. Fig. 12 implies

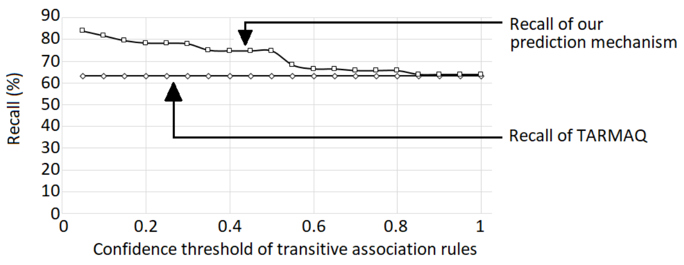

Fig. 3. Comparison of recalls for Brlcad

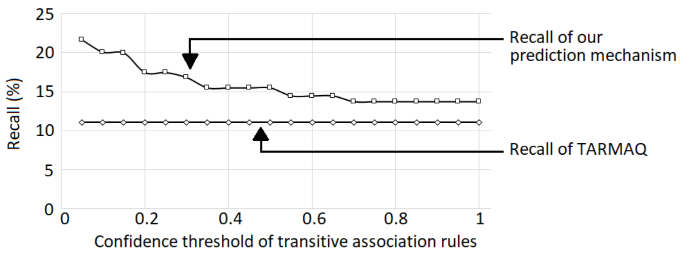

Fig. 4. Comparison of recalls for Ctags

that the precision of our prediction mechanism is higher than TARMAQ for three subject systems except BRL-CAD.

Answer to RQ 2. From our experiment and analysis we can state that *transitive association rules combined with regular rules can help us in better prediction of co-change candidates considering method level granularity*.

We should note that the precision and recall of both techniques (our proposed one and TARMAQ) are low for some subject systems such as Ctags and Camellia. The reason behind this is that we have investigated using method granularity. Kagdi et al. [21] showed that investigation on evolutionary coupling considering a finer granularity such as method granularity can result in lower precision and recall. Kagdi et al. [21] achieved at best 28% recall and 9% precision in their study. However, in our study we see that the precisions and recalls are considerable for some subject systems such as BRL-CAD and MonoOSC. The reason behind this is that we have used transitive association rules which have helped us achieve better precisions and recalls.

VII. Threats to Validity

We did not investigate enough subject systems in our study, and thus, our findings might not be generalized. However, we selected our systems focusing on the diversity of their application domains (four domains), sizes, revision history lengths, and implementation languages (C and C#) so that we can avoid subject system bias on our findings.

While the existing studies have investigated association rules consisting of more than two program entities, we investigate binary association rules each consisting of two entities (one entity is the antecedent and the other entity is the consequent). However, even after using binary association rules we detected co-change candidates for target method sets consisting of two or more entities. Section VI describes how we do this. When suggesting co-change candidates for a target method, we retrieve all binary association rules from the previous commits where the target method is the antecedent. The union of the

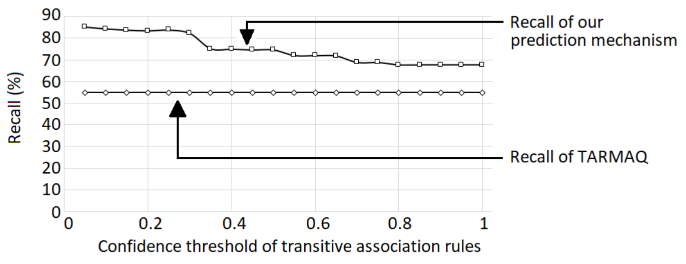

Fig. 5. Comparison of recalls for MonoOSC

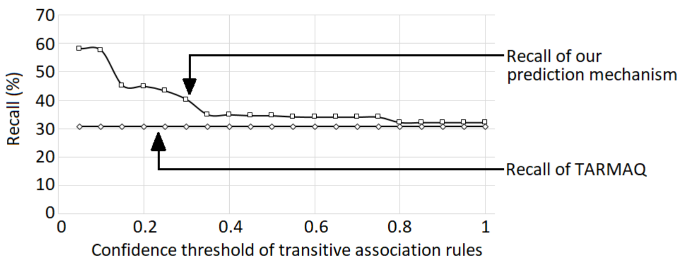

Fig. 6. Comparison of recalls for Camellia

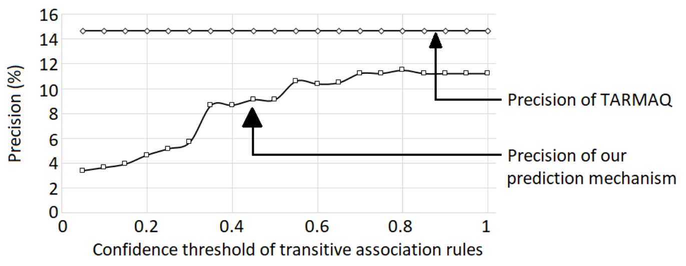

Fig. 7. Comparison of precisions for Brlcad

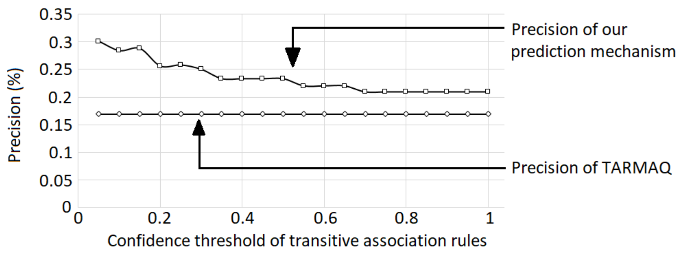

Fig. 8. Comparison of precisions for Ctags

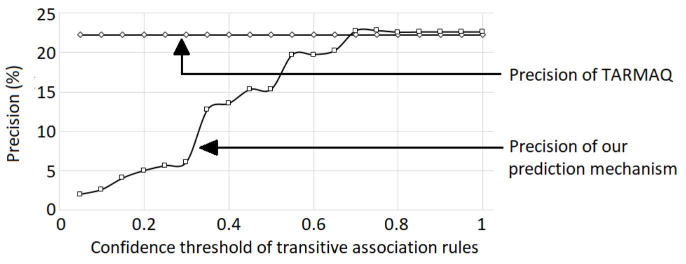

Fig. 9. Comparison of precisions for MonoOSC

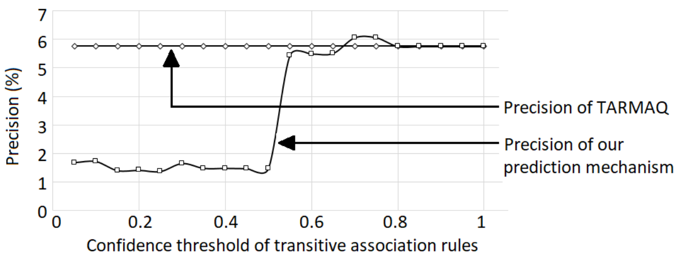

Fig. 10. Comparison of precisions for Camellia

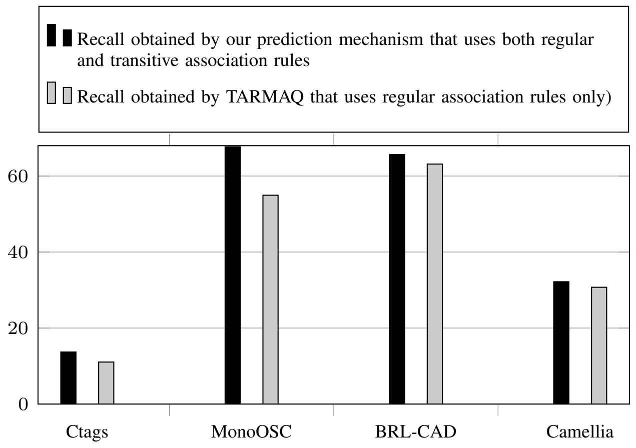

Fig. 11. Comparison regarding recall in predicting co-change candidates

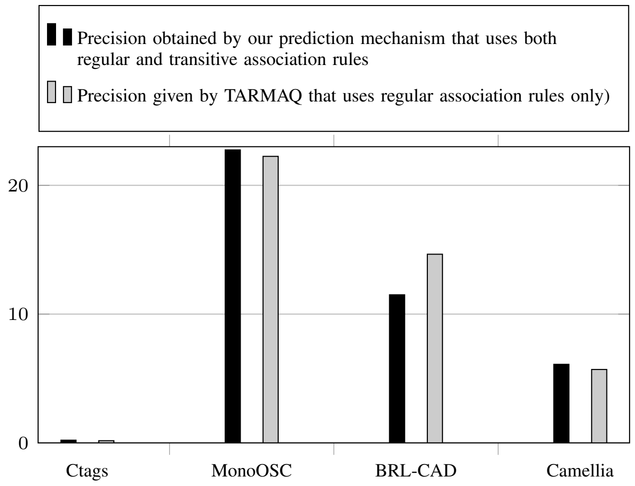

Fig. 12. Comparison regarding precision in predicting co-change candidates

consequents of these association rules is considered as the set of suggested co-change candidates for the target method. Thus, we do not miss any suggestions. Moreover, binary association rules makes us capable of handling unseen queries. We finally believe that our idea of working with binary association rules is reasonable.

Some of the existing studies [9], [37], [8], [23] have investigated evolutionary coupling by filtering out association rules with low support values. Such type of filtering is reasonable while dealing with file level evolutionary coupling. We investigate method level evolutionary coupling. Infrequent co-change of methods is often important while dealing with method level granularity [26], [27]. Thus, our decision of not discarding low support association rules is reasonable.

VIII. Related Work

The concept of *evolutionary coupling* originated from *association rules* introduced by Agrawal et al. [1]. They introduced an association rule to represent frequent item-sets in large databases. Later, this concept has been heavily used to represent change coupling (i.e., evolutionary coupling) among different program entities in a software system.

A great many studies [38], [16], [15], [19], [13], [18], [2], [32], [3], [11], [33], [31] have been conducted on the use of evolutionary coupling in software maintenance research and practice. In all of these studies evolutionary coupling has been realized by regular association rules. *Support* and *confidence* measures have been used to determine the likeliness of the presence of evolutionary coupling among the entities in a rule.

A number of studies [34], [9], [20], [26], [4], [7] investigated ways to improve the detection accuracy of evolutionary coupling by combining association rule mining with the Granger causality test [9], macro co-change and dephase macro co-change [20], and interactions [7]. There are some preliminary studies investigating some new measures such as *significance* [26], and pattern age and pattern distance [4]. These measures are dependent on the co-change frequency of entities. We see that each of these studies investigated ways to improve the detection accuracy of change coupling (i.e., evolutionary coupling), however, none of these deal with transitive association rules. Our study introduces transitive association rules and shows that these can provide better prediction of co-change candidates when combined with regular association rules.

Kagdi et al. [21] combined evolutionary coupling with conceptual coupling in order to identify the impact sets in change impact analysis. They identified evolutionary coupling using association rules and the related measures: *support* and *confidence*. However, we show that evolutionary coupling can be better identified when we consider regular association rules in combination with transitive association rules.

Rolfsnes et al. [31] investigated file level evolutionary coupling and determined co-change candidates for unseen queries by applying regular association rules on the previously seen parts of the query. We have implemented their mechanism called TARMAQ considering method level granularity and compared with our prediction mechanism that considers both regular and transitive association rules. TARMAQ works only with regular association rules. We compared our prediction mechanism with TARMAQ and found that our mechanism outperforms TARMAQ in predicting co-change candidates with 13.96% higher recall and 5.56% higher precision. The reason why our prediction mechanism performs better is that it considers transitive association rules that can predict coupling among methods which did not co-change previously. We did not compare our prediction mechanism with other existing techniques called ROSE [38] and SVD [36]. Rolfsnes et al. [31] compared TARMAQ with these techniques and showed that TARMAQ performs better than these techniques. Thus, comparing our technique with only TARMAQ is reasonable.

To the best of our knowledge, our study is the first one to introduce and investigate transitive association rules for better prediction of co-change candidates for methods. We find that transitive association rules combined with regular association rules can help us in better prediction of co-change candidates when compared with a state of the art tool called TARMAQ. Transitive association rules can assist regular association rules in all aspects where evolutionary coupling is used.

IX. Conclusion

In this paper, we introduce a novel a concept, *transitive association rules*, that helps us realize evolutionary coupling among program entities that did not co-change in the past. We call this coupling *transitive evolutionary coupling* in our research. The regular association rules cannot predict coupling among program entities that did not change together previously. For making transitive association rules, we apply transitivity on the regular association rules. Our experiment on thousands of commits from four diverse subject systems reveals that transitive association rules combined with regular ones can help us in better prediction of co-change among program entities (methods in our research). Our co-change prediction mechanism that uses both regular and transitive association rules outperforms a state of the art technique called TARMAQ with 13.96% higher recall and 5.56% higher precision. TARMAQ deals with regular association rules only. We believe that our proposed idea of transitive association rules will be beneficial in dealing with evolutionary coupling as well as change impact analysis. In future, we plan to apply this idea in different other aspects such as bug localization and finding cross-cutting concerns where evolutionary coupling has been used. Our implementation and data from our research are available on-line [30].

References

[1] R. Agrawal, T. Imieliski, A. Swami, "Mining association rules between sets of items in large databases", Proc. *ACM SIGMOD*, 1993, 22(2): 207 - 216.

[2] S. N. Ahsan, F. Wotawa, "Fault Prediction Capability of Program Files Logical-Coupling Metrics," Proc. *IWSM-MENSURA*, 2011, pp. 257 - 262.

[3] N. Ali, F. Jaafar, A.E. Hassan, "Leveraging Historical Co-change Information for Requirements Traceability", Proc. *WCRE*, 2013, pp. 361 - 370.

[4] A. Alali, B. Bartman, C. D. Newman, J. I. Maletic, "A Preliminary Investigation of Using Age and Distance Measures in the Detection of Evolutionary Couplings", Proc. *MSR*, 2013, pp. 169 - 172.

[5] R. S. Arnold, S. A. Bohner. "Impact analysis-towards a framework for comparison", Proc. *Conference on Software Maintenance*, 1993, pp. 292 - 301.

[6] T. Ball, J.-M. Kim, A. A. Porter, and H. P. Siy, "If your version control system could talk", Proc. *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, vol. 11, 1997, pp. 1 - 5.

[7] F. Bantelay, M. B. Zanjani, H. Kagdi, "Comparing and Combining Evolutionary Couplings from Interactions and Commits", Proc. *WCRE*, 2013, pp. 311 - 320.

[8] G. Bavota, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, A. D. Lucia, "An Empirical Study on the Developers' Perception of Software Coupling", Proc. *ICSE*, 2013, pp. 692 - 701.

[9] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, "Using multivariate time series and association rules to detect logical change coupling: An empirical study", Proc. *ICSM*, 2010, pp. 1 - 10.

[10] G. Canfora, and L. Cerulo. "Impact analysis by mining software and change request repositories", Proc. *Software Metrics*, 2005, pp. 9 - 29.

[11] G. Canfora, L. Cerulo, and M. D. Penta, "On the Use of Line Co-change for Identifying Crosscutting Concern Code", Proc. *ICSM*, 2006, pp. 213 - 222.

[12] M. DAmbros, M. Lanza, "Reverse Engineering with Logical Coupling", Proc. *WCRE*, 2006, pp. 189 - 198.

[13] M. D'Ambros, M. Lanza, M. Lungu, "The evolution radar: Visualizing integrated logical coupling information", Proc. *MSR*, 2006, pp. 26 - 32.

[14] Exuberant CTAGS: https://sourceforge.net/projects/ctags/?source=directory

[15] H. Gall, M. Jazayeri, J. Krajewski, "CVS Release History Data for Detecting Logical Couplings", Proc. *IWPSE*, 2003, pp. 13 - 23.

[16] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history", Proc. *ICSM*, 1998, pp. 190 - 199.

[17] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods", *Econometrica*, 1969, 37(3): 424 - 438.

[18] N. Hanakawa, "Visualization for software evolution based on logical coupling and module coupling",Proc. *APSEC*, 2007, pp. 214 - 221.

[19] J. Itkonen, M. Hillebrand, V. Lappalainen, "Application of Relation Analysis to a Small Java Software", Proc. *CSMR*, 2004, pp. 233 - 239.

[20] F. Jaafar, Y. Gueheneuc, S. Hamel, G. Antoniol, "An Exploratory Study of Macro Co-changes", Proc. *WCRE*, 2011, pp. 325 - 334.

[21] H. Kagdi, M. Gethers, D. Poshyvanyk, M. L. Collard,"Blending Conceptual and Evolutionary Couplings to Support Change Impact Analysis in Source Code", Proc. *WCRE*, 2010, pp. 119 - 128.

[22] H. Kagdi, M. Gethers, D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software", *Empirical Software Engineering*, 2013, 18(5):933 - 969.

[23] S. Kotsiantis , D. Kanellopoulos, "Association Rules Mining: A Recent Overview", *GESTS International Transactions on Computer Science and Engineering*, 2006, 32(1):71 - 82.

[24] A. Lozano and M. Wermelinger, "Tracking clones imprint", Proc. *IWSC*, 2010, pp. 65 - 72.

[25] M. M. Lehman, and J. F. Ramil. "Software evolutionbackground, theory, practice", *Information Processing Letters* 88.1 (2003): 33 - 44.

[26] M. Mondal, C. K. Roy, K. A. Schneider, "Improving the Detection Accuracy of Evolutionary Coupling", Proc. *ICPC*, 2013, pp. 223 - 226.

[27] M. Mondal, C. K. Roy, K. A. Schneider, "Improving the detection accuracy of evolutionary coupling by measuring change correspondence", Proc. *CSMR-WCRE*, 2014, pp. 358 - 362.

[28] M. Mondal, C. K. Roy, K. A. Schneider, "Automatic ranking of clones for refactoring through mining association rules", Proc. *CSMR-WCRE*, 2014, pp. 114 - 123.

[29] M. Mondal, C. K. Roy, K. A. Schneider, "Automatic Identification of Important Clones for Refactoring and Tracking", Proc. *SCAM*, 2014, pp. 11 - 20.

[30] M. A. Islam, M. M. Islam, M. Mondal, B. Roy, C. K. Roy, and K. A. Schneider, "Implementation and data from our investigation regarding transitive association rules", goo.gl/gVwcDj

[31] T. Rolfsnes, S. D. Alesio, R. Behjati, L. Moonen and D. W. Binkley, "Generalizing the Analysis of Evolutionary Coupling for Software Change Impact Analysis", Proc. *SANER*, 2016, pp. 201 - 212.

[32] C. Tantithamthavorn, A. Ihara, K. Matsumoto, "Using Co-Change Histories to Improve Bug Localization Performance", Proc. *ACIS*, 2013, pp. 543 - 548.

[33] S. Wenzel, H. Hutter, U. Kelter, "Tracing Model Elements", Proc. ICSM, 2007, pp. 104 - 113.

[34] R. Robbes, D. Pollet, and M. Lanza, "Logical coupling based on finegrained change information", Pro. *WCRE*, 2008, pp. 42 – 46.

[35] SourceForge: https://sourceforge.net/

[36] Singular Value Decomposition. https://en.wikipedia.org/wiki/Singular-value_decomposition

[37] A. T. T. Ying, G. C. Murphy, R. Ng, M. C. Chu-Carroll, "Predicting source code changes by mining change history," *TSE*, 2004, 30(9):574–586.

[38] T. Zimmermann, P. Weissgerber, S. Diehl, A. Zeller, "Mining Version Histories to Guide Software Changes", Proc. *ICSE*, 2005, pp. 563 - 572.