Improving the Detection Accuracy of Evolutionary Coupling

Manishankar Mondal Chanchal K. Roy Kevin A. Schneider
Department of Computer Science, University of Saskatchewan, Canada
{mshankar.mondal, chanchal.roy, kevin.schneider}@usask.ca

***Abstract*—If two or more program entities (e.g., files, classes, methods) co-change frequently during software evolution, these entities are said to have evolutionary coupling. The entities that frequently co-change (i.e., exhibit evolutionary coupling) are likely to have logical coupling (or dependencies) among them. Association rules and two related measurements, *Support* and *Confidence*, have been used to predict whether two or more co-changing entities are logically coupled. In this paper, we propose and investigate a new measurement, *Significance*, that has the potential to improve the detection accuracy of association rule mining techniques. Our preliminary investigation on four open-source subject systems implies that our proposed measurement is capable of extracting coupling relationships even from infrequently co-changed entity sets that might seem insignificant while considering only *Support* and *Confidence*. Our proposed measurement, *Significance* (in association with *Support* and *Confidence*), has the potential to predict logical coupling with higher precision and recall.**

***Index Terms*—Association Rules, Confidence and Support of Rules, Rule Significance.**

I. Introduction

Awareness of logical dependencies, that is logical coupling, among program entities is important from the perspective of software maintenance. A particular entity might have logical dependencies with many other entities. If we are not aware of all these dependencies, changes to that particular entity might introduce inconsistencies to the other related entities.

A widely studied approach of discovering the logical coupling among program entities is to analyze how they have co-changed during system evolution. Co-changeability of program entities is also termed as 'evolutionary coupling' [3]. The underlying idea is that if two or more entities co-change frequently, then it is very likely that there exists a logical coupling among them. Association rule mining technique [1] has been widely used to identify frequently co-changed entities that are likely to be logically coupled. In the existing studies, two measurements, *Support* and *Confidence*, have been used to rank the rules according to the quality of the rules. Obviously, higher values of these measurements can retrieve better quality rules and indicate higher likeliness of logical coupling. However, from our manual investigation we found that there are many rules that might appear as less promising or ignorable (because they have lower ranks) while considering these two measurements (*Support*, and *Confidence*), but, these rules can discover important coupling relationship among program artifacts. Thus, if we consider only *Support* and *Confidence*, there is a possibility of ignoring significant coupling information.

In order to minimize this drawback of the existing measurements, we propose a new measurement, *Significance*, and conduct preliminary investigation on it with promising early evidence. *Significance* quantifies the importance of the co-change of a set of co-changing entities by determining how much focused they were in those commits where they co-changed. Our idea of introducing *Significance* is new in the sense that it has the potential to retrieve coupling relationship from the infrequently co-changed entity sets. There is no previous study on how we can retrieve coupling relationship from infrequently co-changed entity sets. The research work conducted by Zimmermann et al. [7] is mostly related to our work. They used association rules with *Support* and *Confidence* to represent the co-change relationships among different program artifacts such as files, classes, methods, and variables. We have a related study [6] that investigates the effect of higher connectivity among co-changed method groups during software evolution.

According to our preliminary investigation, our proposed measurement has the potential to improve the precision and recall of association rule mining techniques. We observe that there are some rules that do not seem to be promising (that is, these rules are assigned lower ranks) considering the existing measurements (*Support*, and *Confidence*). However, these rules are true positives and our proposed measurement assigns higher ranks to these rules. While *Significance* can no way be a replacement of the existing measurements, it can complement them to retrieve better results.

The rest of the paper is organized as follows. Section II defines association rule and the related measurements, *Support* and *Confidence*. Section III describes our motivation behind introducing the new measurement, *Significance*. Section IV defines and describes *Significance*. Section V demonstrates the experimental steps and Section VI discusses our preliminary experimental results. Section VII presents the related work and Section VIII concludes the paper describing our future work.

II. Association Rule

Definition of Association Rule: An association rule is an expression of the form $X => Y$ where X is the antecedent and Y is the consequent. Each of X and Y is a set of one or more program entities. As we consider method level granularity, the sets X and Y consist of methods. Such a rule means that if X gets changed in a particular commit operation, Y also has

978-1-4673-3092-3/13 © 2013 IEEE

ICPC 2013, San Francisco, CA, USA

the tendency of getting changed in that commit. *Support*, and *Confidence* of a rule can be calculated in the following way.

Support: Support is the count of commits in which a method or a group of methods (together) appeared (got modified) during system evolution. Suppose, X is a set of methods that appeared together in C_X number of commit operations during evolution, then the support of this method set, *support(X)* = C_X. The support of an association rule, X => Y, is determined by the support of the union of the method sets, X and Y. We denote this according to the following equation.

$$support(X => Y) = support(XY) = C_{XY} \quad (1)$$

Here C_{XY} is the number of commits in which both X and Y (that means, the union of X and Y) appeared (got modified).

Confidence: Confidence of the association rule, X => Y, determines the probability that Y will appear in a commit operation provided that X appears in that commit operation. We calculate the confidence of X => Y by Eq. 2.

$$confidence(X => Y) = \frac{support(X => Y)}{support(X)} = \frac{C_{XY}}{C_X} \quad (2)$$

III. Motivation behind New Measurement

In general, if a group of program entities have higher support (i.e., the entities co-changed frequently), there is a higher likeliness of the existence of logical coupling among the entities. From this intuition, the existing association rule mining techniques emphasize on retrieving rules with higher support values. However, during manual investigation of the method co-change history of our candidate software systems we experienced that infrequently co-changed methods (i.e., the methods that co-changed with lower support value such as 1 or 2) sometimes exhibit logical coupling among them. From this we realize that if we only emphasize on the frequently co-changed entity sets excluding the infrequently co-changed entity sets from consideration, we might lose important coupling relationships that could be retrieved from the infrequently co-changed entity sets. We further investigated to identify those situations when a set of infrequently co-changed entities can exhibit logical coupling among them. We investigated considering method level granularity.

According to our observation, if too many methods (e.g., 20 or more) co-change in one commit operation, we rarely find a logical coupling among these methods. However, we found many examples where only two or three methods co-changed in only one commit operation during the observed range of evolution but these methods are logically coupled. The fact is that if a single commit operation affects too many entities, such a commit is likely to involve major structural changes and thus, is likely to affect methods from unrelated functionalities. Such commits are termed as atypical commits [5] in the literature. In the following two paragraphs we describe our observations with relevant examples from our candidate system, Ctags.

Observation regarding the commits that affect fewer methods: By manually investigating the method co-changes in Ctags, we found that the methods, *reportType* and *isGeneric*, in file *eiffel.c* co-changed in only one commit operation ($support$ = 1), the commit on revision 428, during the whole period of evolution consisting of 774 commits. No other method in the system was changed in this particular commit operation (i.e., on revision 428). In this commit, a call to method *stringListHasInsensitive* was replaced by another method-call to *stringListHas* in both of these methods (*reportType*, and *isGeneric*). The changes occurred to these methods imply that they are logically coupled. We found many other examples where two methods co-changed in only one or two commit operations during the evolution but they are logically coupled. The interesting matter is that the total number of methods co-changed in each of these commits (e.g., commit on revision 428, 130, 138, 337, 746, 730) is very low (two for most the cases). We term each of these commits as a *Commit Affecting Fewer Methods* (CAFM). We understand that as the number of methods co-changed in a *CAFM* is very low, these methods are more likely to take part in a common purpose and these are more likely to be logically coupled compared to the methods co-changed in a commit which affects many methods.

Observation regarding the atypical commits: We observe that the commit on revision 442 (of Ctags) affects 34 methods in total. We consider this commit operation as an atypical commit. We investigated the changes in these methods and found that many of these changes only affect the indentation of the statements and it is difficult to infer a logical dependency among the co-changed methods. For example, we mention two methods, *makeDefineTag* (in file *get.c*) and *newToken* (in file *fortran.c*), of these 34 methods that co-changed in this commit operation. The changes do not imply a logical relationship among these methods at all. We investigated several other atypical commits (e.g., the commit operation on revision 88, 242, 535) and experienced that a logical coupling among the co-changed methods can be rarely inferred.

From the above scenario we understand that *the higher the number of co-changed methods in a commit operation is, the lower the likeliness of the existence of logical coupling among the co-changed methods is*. We thus note the followings.

- Infrequent co-changes (e.g., support = 1) might indicate logical coupling if the co-changes take place in *CAFM*s.
- If two methods co-changed several times (support > 1) but only in atypical commits, these methods might not be logically coupled.

Focusing on these findings, a possible way of improving the accuracy of predicting co-changeable candidates (logically coupled candidates) could be to exclude the atypical commits from consideration. However, this is very difficult to detect the atypical commits. There is no empirically established threshold on the number of entities that should be changed in a particular commit to consider that commit as an atypical commit.

In this research work, we propose and investigate a new measurement, *Significance*, that has the potential to improve the predictability of co-changeable (logically coupled) candidates (methods in our study). We calculate significance (Section IV) in such a way that it has the tendency of excluding atypical commits by giving more emphasis on the *CAFM*s.

IV. Our Proposed Measurement

Suppose N_c ($N_c \geq 2$) methods changed together (or co-changed) in a particular commit operation c. These methods are, $m_{1_c}, m_{2_c}, m_{3_c}, ... m_{N_c}$. For any pair of these methods we calculate the significance according to the following equation.

$$S_c = \frac{1}{N_c - 1} \quad (3)$$

Here, S_c is the significance of the co-change of any method pair (m_{i_c}, m_{j_c}) considering the commit c. Fig. 1 shows the change history of five methods: m_1, m_2, m_3, m_4, and m_5 during six commits, $c1$ to $c6$. The significances for the commits: c2, c3, c4, and c6 are S_{c2}, S_{c3}, S_{c4}, and S_{c6} respectively. In each of the other commits only one method changed. We cannot calculate significance for these commits.

Suppose, N methods were changed (appeared in different commit operations) in total during the evolution of a software system. For every possible pair of these methods we calculate the significance considering the entire range of evolution.

A Method Pair: If two methods, m_i and m_j changed together (with or without being associated by other methods) in at least one commit operation during software evolution, we call them a method pair (m_i, m_j). Fig. 1 shows four possible method pairs: (m_1, m_2), (m_3, m_5), (m_1, m_3), and (m_2, m_3).

Suppose, the set of commit operations where a method pair, m_i and m_j ($1 \leq i \leq N$, $1 \leq j \leq N$), co-changed is $C_{m_i m_j}$. Then, the significance of co-change of this method pair during system evolution is the summation of the significance values of this method pair considering the commits in $C_{m_i m_j}$. We calculate this in the following way.

$$S(m_i, m_j) = \sum_{c \epsilon C_{m_i m_j}} S_c \quad (4)$$

Here, $S(m_i, m_j)$ is the significance of the co-change of the method pair, m_i and m_j, considering the entire range of evolution. Fig. 1 shows the calculation of the significance of the method pair, (m_2 m_3), that co-changed in commits $c4$ and $c6$. For every possible method pair (m_i, m_j) we can assume two rules, m_i => m_j and m_j => m_i. The significance, $S(m_i$ => $m_j)$, of a rule, m_i => m_j, is determined by the significance of the method pair (m_i, m_j).

$$S(m_i \text{ => } m_j) = S(m_i, m_j) \quad (5)$$

We can easily understand that the rules, m_i => m_j and m_j => m_i, have the same support and the same significance values. Confidences of these two rules might be different.

Justification and implications of Significance: Higher significance implies higher likeliness of the existence of a logical coupling among the co-changed methods. According to Eq. 3, the significance (S_c) regarding a particular commit

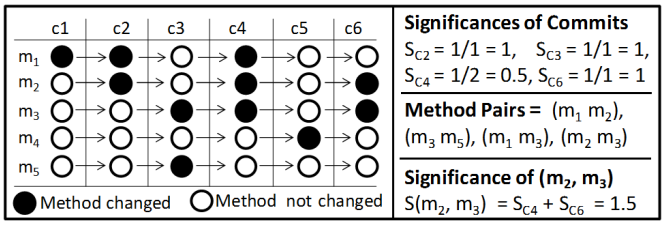

Fig. 1. An example of calculating significance

TABLE I
Subject Systems

Systems	Language	Domains	LOC	Revisions
Carol	Java	Game	25,092	1699
Ctags	C	Code Def. Generator	33,270	774
Camellia	C	Multimedia	85,015	207
GreenShot	C#	Multimedia	37,628	999

(c) is inversely proportional to the total number of methods (N_c) affected in that commit. S_c gets the highest value ('1'), if only two methods (N_c = 2) co-change in c. Thus, if only two methods co-change in a commit operation, Eq 3 assumes the highest likeliness of the existence of a coupling relationship among the co-changed methods (considering that commit operation). Also, S_c becomes negligible for an atypical commit. Thus, we believe that Eq. 3 reasonably considers our findings described in Section III.

Although the highest possible value of the significance (S_c) regarding a particular commit (c) is one, the significance of the co-change of a method pair for the whole period of evolution can be greater than one, because the method pair can co-change in more than one commit operation. For a particular method pair, the highest possible significance value equals the support value of the method pair.

V. Experimental Steps

For the purpose of our investigation we downloaded the candidate software systems listed in Table I from the open source SVN repository. We sequentially performed the following experimental steps for each of the software systems: (1) preprocessing of the source code of all the revisions, (2) determination and storage of methods from each of the revisions using CTAGS [2], and (3) detection of method genealogies. For the details of these steps we refer the interested readers to [6]. In the previous section we described how we determine method pairs (m_i, m_j), calculate the significance for a method pair (using Eq. 4) considering the whole range of evolution and assume association rules from each method pair.

VI. Experimental Results and Discussion

Our discussions in the previous sections indicate that rules with smaller supports such as 1, or 2 might also discover important coupling information if they have higher significance values. In the following subsections we provide and discuss our preliminary experimental results that support our findings and show the effectiveness of our proposed measurement in discovering significant association rules as well as coupling relationships even for the lowest support value.

We determined all possible method pairs, their supports, and significances for each of our candidate systems. We should note that a method pair (m_i, m_j) is a representative of the rules, m_i => m_j and m_j => m_i. We sorted these method pairs in two ways, according to the non-increasing order of the support values and significance values. We observed that many method pairs, that are generally considered as insignificant (or ignorable) because of their lower support values (e.g., support = 1 or 2), appear as significant ones because of their higher significance values.

TABLE II
PERCENTAGE OF TRUE POSITIVES FOR SUPPORT = 1 AND 2 (FOR CTAGS)

Support = 1			Support = 2		
SV / SR	CMP	PTP	SV / SR	CMP	PTP
S = 1	26	80.76	S = 2	1	100
S = 0.5	29	62.06	1.5 >= S >= 1	17	83.33
0.4 > S >= 0.3	42	25	0.75 >= S >= 0.5	5	80
S = 0.25	60	20	0.45 >= S >= 0.3	27	88.88
S = 0.2	52	10	0.3 > S >= 0.2	50	70
S < 0.2	1828	5	S < 0.2	92	40

S = Significance SV / SR = Significance Value / Significance Range
CMP = Count of Method Pairs PTP = Percentage of True Positives
* No method pair was found with the missing SV / SR

We performed a rigorous manual investigation on our candidate system, Ctags, to determine whether infrequently co-changed (e.g., support = 1 or 2) method pairs with higher significance values exhibit logical coupling among the constituent methods. For two support values, 1 (the smallest one) and 2, we separated the method pairs obtained from this subject system in difference ranges of significance values. For each range of significance we determined the percentages of true positives. A method pair is considered as true positive if the two constituent methods are logically coupled. In order to determine whether a method pair is true positive, we analyzed the changes occurred to the two methods in those commit operations where they co-changed.

Table II contains the number of method pairs (CMP) and percentages of true positives (PTP) for different significance values / ranges (SV/SR) for support = 1 and 2. We did not find any method pair for the missing SV/SR. We should note that the highest possible value of the significance of a method pair is equal to its support value. For this reason, the highest significance values are 1 and 2 in case of support values of 1 and 2 respectively. In case of Support = 1 we manually investigated all the method pairs for the first two significance values (S = 1, S = 0.5) to determine how many of the method pairs are true positives. For each of the other four values / ranges we manually investigated the first 20 method pairs and determined the percentage of true positives (PTP) considering these 20 pairs. We followed the same approach (evaluation of the first 20 method pairs) for each of the two ranges, $0.3 > S \geq 0.2$ and $0.2 > S$, in case of Support = 2. For the remaining four values / ranges we manually investigated all method pairs.

Table II indicates that we get higher percentage of true positives (PTP) for higher significance values and PTP gradually decreases with the decrease of significance value (with only one exception for the significance range $0.45 >= S >= 0.3$ in case of Support = 2). From this we can say that *higher significance generally indicates higher likeliness of the existence of a logical coupling between the co-changed methods*. Moreover, *considering significance we can retrieve important association rules as well as coupling relationships from infrequently co-changed method sets which might be ignored (or regarded as insignificant) by the existing association rule mining techniques because of smaller support values (e.g., support = 1 or 2)*. Thus, *Significance* has the potential to improve the detection accuracy of association rule mining techniques.

VII. RELATED WORK

Association rules, introduced by Agarwal et al. [1] have been frequently used to find associated or co-changing program artifacts (also known as frequent itemsets). Gall et al. [3] introduced an approach for discovering logical dependencies analyzing the evolutionary coupling (or co-changing) of different program modules. Zimmermann et al. [7] used *association rules* with *support*, and *confidence* to represent the co-change relationships among different program artifacts. Jafar et al. [4] performed a comprehensive study on macro co-changes considering file level granularity and introduced the patterns, *MCC* (macro co-changes) and *DMCC* (diphase macro co-changes), that can help in retrieving file level coupling.

The existing studies related to association rules emphasized on identifying frequently co-changed entity sets. Also, most of these studies investigated only file level co-changes. Our study is different in the sense that we investigate on how to extract important association rules as well as coupling relationships from infrequently co-changed entity sets, because if we ignore infrequently co-changed entity sets we might lose significant coupling information. Moreover, we conduct our study considering a finer granularity, in the method level.

VIII. CONCLUSION

In this research work, we propose and investigate a new measurement, *Significance*, that has the potential to improve the precision and recall of the association rule mining techniques. According to our preliminary investigation, using this measurement we can discover logical coupling from infrequently co-changing candidates. Infrequently co-changed (e.g., support = 1) candidates are less emphasized by the existing association rule mining techniques because of their smaller support values. However, our investigation result suggests that consideration of *Significance* in association with *Support* and *Confidence* can improve the recall of these techniques. The proposed measurement has also the potential to improve the precision of such techniques by reasonably disregarding those candidates that have mainly co-changed in the atypical commits (that affect too many entities). As a future work, we plan to investigate different existing association rule mining techniques to see whether our proposed new measurement can improve their accuracy in terms of precision and recall.

REFERENCES

[1] R. Agrawal, T. Imieliski, A. Swami, "Mining association rules between sets of items in large databases". Proc. *ACM SIGMOD*, 1993, Vol. 22, Issue 2, pp. 207–216.
[2] Exuberant Ctags: http://ctags.sourceforge.net/
[3] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," Proc. *ICSM*, 1998, pp. 190–199.
[4] F. Jaafar, Y. Gueheneuc, S. Hamel, G. Antoniol, "An Exploratory Study of Macro Co-changes", Proc. *WCRE*, 2011, pp. 32–334.
[5] A. Lozano, M. Wermelinger, "Assessing the effect of clones on changeability," Proc. *ICSM*, 2008, pp. 227–236.
[6] M. Mondal, C. K. Roy, K. A. Schneider, "Connectivity of Co-changed Method Groups: A Case Study on Open Source Systems", Proc. *CASCON*, 2012, pp. 205 – 219.
[7] T. Zimmermann, P. Weisgerber, S. Diehl, A. Zeller, "Mining version histories to guide software changes," Proc. *ICSE*, 2004, pp. 563–572.