Using Multivariate Time Series and Association Rules to Detect Logical Change Coupling: an Empirical Study

Gerardo Canfora*, Michele Ceccarelli**, Luigi Cerulo**, Massimiliano Di Penta*
* Dept. of Engineering-RCOST, University of Sannio, Italy
** Dept. of Biological and Environmental Studies, University of Sannio, Italy
{canfora, ceccarelli, lcerulo, dipenta}@unisannio.it

***Abstract*—In recent years, techniques based on association rules discovery have been extensively used to determine change-coupling relations between artifacts that often changed together. Although association rules worked well in many cases, they fail to capture logical coupling relations between artifacts modified in subsequent change sets.**

To overcome such a limitation, we propose the use of multivariate time series analysis and forecasting, and in particular the use of Granger causality test, to determine whether a change occurred on a software artifact was consequentially related to changes occurred on some other artifacts.

Results of an empirical study performed on four Java and C open source systems show that Granger causality test is able to provide a set of change couplings complementary to association rules, and a hybrid recommender built combining recommendations from association rules and Granger causality is able to achieve a higher recall than the two single techniques.

Keywords: Change Coupling, Multivariate Time Series, Impact Analysis, Software Evolution, Empirical Study.

I. Introduction

When developers modify, during maintenance and evolution, a software artifact, it is very important to understand what other artifacts are likely to be changed immediately or in the near future. As pointed out by Zimmermann *et al.* [1], this has important consequences on software quality, but also on the ability of project managers to plan software maintenance and evolution activities.

As a matter of fact, a key application of change coupling detection is the identification of artifacts that have been—directly or indirectly—impacted by a change. To make an example of how this helps software maintenance, let us suppose that a change occurring to a class A belonging to the data model would impact classes B and C belonging to the business logic layer. If the developer, for example, commits change only to classes A and B, she may be warned about likely changes needed for class C. Such a suggestion has three positive consequences: (i) it prevents possible bugs due to leaving the system in an inconsistent state; (ii) it provides an idea of what is the impact of a change, that is if changes occurring to a class often impact a large number of other classes, then it is likely that the (sub-)system has to be refactored; and (iii) it allows the project manager to estimate the amount of changes to be done and thus plan and staff the maintenance tasks.

In the past, change impact analysis tasks mainly relied on static analysis [2] or dynamic analysis [3]. While these techniques identify actual (direct or indirect) dependencies between artifacts, they have some limitations. On the one hand, in many cases artifacts have many dependencies, meaning that a change could potentially impact a very large set of artifacts; this makes the recommendation quite complete, but not very practical. On the other hand, there may exist artifacts that have a strong change coupling without exhibiting a dependency: artifacts belonging to different stages of the software development life-cycle (e.g., requirements, design documents, user manuals, and source code), and components that exhibit hidden dependencies through the services of a publish-subscribe middleware for events/message passing are just some examples. In some cases, such a lack of dependency between related artifacts has been dealt by using a textual analysis [4].

In recent years, information available in software repositories, namely versioning systems, issue tracking systems and other repositories, has been largely used to perform various kinds of studies and to develop approaches aimed at supporting developers and managers in various tasks. In general, *historical analysis* of data from software repositories provides information complementary to static and dynamic analysis. In the context of change coupling detection, Ying *et al.* [5] and Zimmermann *et al.* [1], [6] proposed a technique to determine the impact of changes based on association rule discovery. Such technique learns change impact relations based on the co-changes of software artifacts. While such a technique has proven successful in capturing change relations among artifacts co-changing within a single change set [1], it may fail to capture change relations having a consequentiality spread over an interval of time.

In a previous paper [7] we have outlined the idea of using a Vector Auto–Regression (VAR) model, a generalization of univariate Auto–Regression (AR) model, to capture the evolution and the interdependencies between multiple time series. In particular, we proposed the use the bivariate Granger causality test [8] to identify if the changes of a software artifact are useful to forecasting the changes of

another software artifact. For a couple of software artifacts, *a* and *b*, an equation explains the change evolution of *a* based on its past changes and the past changes of *b*; then, a statistical test shows whether the past changes of *a* are useful for predicting changes to *b*. The bivariate Granger causality test has been successfully used in bioinformatics to identify gene regulatory relationships [9], while to the best of our knowledge it has not been applied yet to study software evolution phenomena where, up to now, univariate time series models have been used to predict software metrics and changes [10].

This paper presents an empirical comparison of change couplings identified by means of the two techniques performed on four open source systems—namely Mylyn, FreeBSD (i386), Rhino, and Squid—showing that, while association rules provides more precise results, Granger causality test is able to achieve a better recall, a better F-measure, and above all to recommend a higher number of true change couplings. More important, the set of true change couplings provided by the two techniques is mostly disjoint. Starting from such a result, the paper also proposes the use of a hybrid change coupling recommender obtained by combining the two techniques. The combined technique is able to achieve a precision in-between association rule discovery and Granger causality test, and an F-measure and a recall higher than the two single techniques.

The paper is organized as follows. Section II overviews association rule discovery and Granger causality test, explains how they can be used to identify change couplings, and outlines why and how the two approaches can be combined to build a better recommender. Section III defines the empirical study we performed, while results are reported and discussed in Section IV. Section V provides a discussion of the threats to validity. After a discussion of related work (Section VI), Section VII concludes the paper and outlines directions for future work.

II. Using Association Rules and Multivariate Time Series to identify Change Couplings

This section describes how we extract data for this study from the change logs of the analyzed systems. Then, it introduces the two techniques used to identify likely change couplings—namely association rule discovery and Granger causality test (in the following simply referred as "association rules" and "Granger"). Finally, it explains how it is possible to build a hybrid classifier by combining the two techniques.

A. Extracting change sets

The evolution of a software system, $SS = \{f_1, f_2, \ldots, f_M\}$, composed of M files and under the control of a versioning system can be viewed as a sequence of source code *Snapshots* (S_1, S_2,...,S_T) generated by a sequence of *Change Sets*, $\Delta_1, \Delta_2, \ldots, \Delta_T$. A change set, Δ_t, represents the changes performed by a developer on a set of source files, $\Delta_t \subseteq S$, at time t. Change sets can be extracted from a versioning (e.g., CVS or SVN) history log by using a time-windowing approach, that considers a change set as a sequence of file revisions that share the same author, branch, and commit notes, and such that the difference between the timestamps of two subsequent commits is less than or equal to 200 seconds [1]. Historical analysis of a software system consists of extracting useful information from change set data.

In this section we assume the following definition of change impact relation from source file f_i to f_j, with $i \neq j$:

> *f_j is impacted by a change in f_i ($f_i \rightarrowtail f_j$) $\Rightarrow$ a change performed in f_i will generate a change in f_j in at least $K \geq 0$ ahead change sets.*

B. Association Rule Discovery

Association rule discovery is an unsupervised learning technique used for local pattern detection showing attribute value conditions that occur together in a given dataset [11]. The application of association rules to change impact prediction assumes a dataset composed by a sequence of change sets, e.g., source files, that have been committed together within an interval of time into a versioning repository [1]. An association rule, $F_{left} \Rightarrow F_{right}$, between two disjoint source file sets means that if a change occurs in each $f_i \in F_{left}$, then another change should happen in each $f_j \in F_{right}$ within the same change set. The strength of an association rule is determined by its support and confidence [11], defined as:

$$\textit{Support} = \frac{|F_{left} \cup F_{right}|}{T}; \quad \textit{Confidence} = \frac{|F_{left} \cup F_{right}|}{|F_{left}|}$$

where T is the total number of change sets extracted from the versioning repository and $|F|$ is the number of change sets, Δ_t, with $F \subseteq \Delta_t$, i.e., the source files in F changed together. In this paper, we perform association rule discovery using a well-known algorithm named *Apriori* [11].

To find the pair of source files, f_i and f_j, such that $f_i \rightarrowtail f_j$ (i.e., f_j is impacted by a change in f_i), we compute the confidence and support for each $f_i, f_j \in S$, with $i \neq j$, and consider the top N of the list of source file pairs ranked by their confidence and support in a decreasing order. Starting from the snapshots identified as described in Section II-A, we estimate variables that can be used to mine association rules. Given $F \equiv \{f_1, \ldots f_m\}$ the set of all versioned artifacts for which we are interested to perform the prediction—i.e., the set of all source code files—association rules can be mined from a change matrix CH, where $ch_{i,j} = 1$ if a file f_j changes in the snapshot S_i, $ch_{i,j} = 0$ otherwise.

C. Granger Causality Test

Granger causality test is a technique for determining whether one time series is useful in forecasting another one [8].

In recent years, it has become popular in bioinformatics to discover gene and metabolic pathways of an organism [9]. The basic idea is that a cause cannot come after the effect, then if a variable *x* affects a variable *z*, the former variable should help improving the predictions of the latter one. We use this notion to capture whether changes performed in a source file could "*cause*", in a Granger sense, a change in another source file. Let $f_k(t)$, $t = 1, \ldots, T$ be the change time series of the source file f_k defined as:

$$f_k(t) = \begin{cases} 1, & f_k \in \Delta_t \\ 0, & f_k \notin \Delta_t \end{cases}$$

i.e., $f_k(t)$ is one if the file f_k changes in snapshot Δ_t, zero otherwise. The simplest bivariate Granger test between two time series $f_1(t)$ and $f_2(t)$ uses the autoregressive specification [12], which consists of the following bivariate and univariate autoregression, solved by estimating the ordinary least squares:

$$f_2(t) = c_1 + \alpha_1 f_1(t-1) + \alpha_2 f_1(t-2) + \ldots + \alpha_p f_1(t-p) + \\ + \beta_1 f_2(t-1) + \beta_2 f_2(t-2) + \ldots + \beta_p f_2(t-p) + u(t)$$

$$f_2(t) = c_1 + \gamma_1 f_1(t-1) + \gamma_2 f_1(t-2) + \ldots + \gamma_p f_1(t-p) + e(t)$$

where p is the lag length, which can be estimated with various criteria [12], $u(t)$ and $e(t)$ are independent and identically distributed random variables. To test whether $f_1(t)$ Granger-cause $f_2(t)$ the null hypothesis to reject (H_0: f_1 does not Granger-cause f_2) is defined as:

$$H_0 : \beta_1 = \beta_2 = \ldots = \beta_p = 0$$

which can be implemented by calculating the sum of squared residuals (RSS):

$$RSS_1 = \sum_{t=1}^{T} \widehat{u}(t)^2; \quad RSS_0 = \sum_{t=1}^{T} \widehat{e}(t)^2$$

The null hypothesis is rejected if

$$Sscore = \frac{(RSS_0 - RSS_1)/p}{RSS_1/(T-2p-1)}$$

is greater than the 5% critical value for an $F(p, T-2p-1)$ distribution. To find the pair of source files, f_i and f_j, such that $f_i \rightarrowtail f_j$ (f_j is impacted by a change in f_i), for each $f_i, f_j \in S$, with $i \neq j$, we consider the top N in the list of pair of source files ranked by $Sscore$ in decreasing order.

To train the multivariate time series model, we discretize the time in terms of snapshots, i.e., the independent variable $t = 1$ for the snapshot S_1 etc. The other independent variables are changes occurring to all files in the system being analyzed. In our preliminary work [7] we used exactly the same matrix of changes used for association rules to build the multivariate time series model. However, differently from association rules, Granger causality is able to better work on series of continuous values (reals) rather than on Boolean variables, and thus be able to keep into account the strength of a relation: if a file f_j always undergoes many changes after file $f_{j'}$ has been changed, again, several times, then the captured change coupling will be stronger than when a file $f_{j'}$ changes a few times after a few changes occurring to f_j. Thus, instead of using a Boolean variable, we populated the change matrix with change frequencies defined as:

$$ch_{i,j} = \frac{changes_{i,j}}{\sum_{k=1}^{m} changes_{i,k}}$$

where $changes_{i,j}$ indicates the number of changes occurred to file f_j in the snapshot S_i. We chosen such a frequency as from some experiments we did it outperformed the simple Boolean matrix when using Granger for change coupling prediction. We are considering, in our work-in-progress—further indicators of strength of a change, e.g., the number of source code lines changed.

D. Building the hybrid change coupling recommender

In addition to empirically assessing the performances of association rules and Granger, in this paper we aim at investigating the performances achieved when combining the two techniques. Since both techniques provide a ranked list of recommendations, we combine them using an approach inspired from ranking fusion techniques [13], that consists of aggregating, for each possible recommendation, the scores provided by the two techniques. Specifically, let us consider each possible pair of files (f_i, f_j) for which association rules provided a confidence $Confidence_{i,j}$, and Granger provided a $Sscore_{i,j}$. It should be noted that, since association rules provide a confidence between two sets, we consider the $Confidence_{i,j}$ of a change coupling between f_i and f_j when f_i is in the left-hand-side of the rule and f_j is in the right-hand side. We assume $Confidence_{i,j} = 0$ if no rule involving f_i and f_j was discovered. Then, we proceed as follows:

1) Normalize both $Confidence$ and $Sscore$ in intervals $[0 \ldots 1]$, where 0 indicates absence of recommendation and 1 indicates the maximum score provided by the technique. This is done as follows:

$$ConfidenceN_{i,j} = \frac{Confidence_{i,j} - MinConf}{MaxConf - MinConf}$$

where $ConfidenceN$ is the normalized confidence obtained, $MinConf$ and $MaxConf$ are the minimum and maximum confidences across all pairs of files respectively. The normalized $Sscore$ ($SscoreN_{i,j}$) is computed similarly.

2) Since we noticed that the $Sscore$ has, differently from the confidence, a skewed distribution, we applied a logarithmic transformation $SscoreNlog_{i,j} = 100 \cdot log_{10}(SscoreN_{i,j})$, where the logarithm is multiplied

Table I
CHARACTERISTICS OF THE ANALYZED SYSTEMS

SYSTEM	LANGUAGE	FILES (OVERALL)	AVERAGE CHANGE SET SIZE (K=5)	VERSION HISTORY		TEST SET	
				PERIOD	# SNAPS	PERIOD	# SNAPS
Mylyn	Java	248	54 (± 3)	2005-06-08 – 2007-10-17	472	2007-06-08 – 2007-10-17	40
FreeBSD (i386)	C/C++	496	14 (± 1)	1998-03-20 – 2000-04-24	1715	2000-03-20 – 2000-04-24	40
Rhino	Java	557	13 (± 2)	2002-06-08 – 2004-07-28	706	2004-06-08 – 2004-07-28	40
Squid	C	538	140 (± 9)	2001-01-17 – 2003-02-05	762	2003-01-17 – 2003-02-05	40

by 100 to avoid that the association rule factor would weight too much when combining the techniques.

3) Aggregate $ConfidenceN_{i,j}$ and $SscoreNlog_{i,j}$ into a new score $comb_{i,j}$ as follows:

$$comb_{i,j} = w_1 \cdot ConfidenceN_{i,j} + w_2 \cdot SscoreNlog_{i,j}$$

where w_1 and w_2 weight the importance of the two techniques. In this paper we performed experiments using the following weights: (i) $w_1 = 0.5, w_2 = 0.5$ (equal importance to both techniques), (ii) $w_1 = 0.75, w_2 = 0.25$ (higher importance to association rules), and (iii) $w_1 = 0.25, w_2 = 0.75$ (higher importance to Granger).

4) Finally, rank the file pairs according to $comb_{i,j}$.

III. EMPIRICAL STUDY

The *goal* of this study is to investigate the use of multivariate time series analysis—and specifically Granger causality test—as a method to identify logical change couplings among files, and to predict changes on some files as a consequence of other changes. The *quality focus* is the accuracy and completeness of the identified change couplings, but also the complementariness of the Granger causality test with alternative techniques used to identify change couplings, namely association rule discovery. The *perspective* is mainly of researchers interested to investigate on novel approaches able to identify change couplings. The *context* consists of four open source software systems, belonging to different domains and developed with different programming languages, namely two C/C++ systems—Squid and FreeBSD (i386)—and two Java systems, Mylyn and Rhino. *Mylyn*[1] a task-manager for Eclipse. *FreeBSD*[2] is a Unix operating system kernel written in C/C++. In this paper, we limited the analysis to the *i386* subsystem. *Rhino*[3] Rhino an ECMA/Javascript interpreter developed for the Mozilla/Firefox browser. *Squid*[4] is a Web caching proxy, written in ANSI C, supporting HTTP, HTTPS, and FTP. Characteristics of the analyzed systems are reported in Table I.

A. Research Questions

This study aims at addressing the following research questions:

- **RQ1:** *What are the performances, in terms of accuracy and completeness of the identified change coupling, of association rules and Granger?* The aim of this research question is to evaluate and compare, in terms of accuracy and completeness, the two techniques we are investigating, i.e., association rules and Granger.
- **RQ2:** *To what extent do the change couplings identified by Granger overlap with those identified by association rules?* This research question aims at comparing the set of true positive links provided by the two techniques, to understand whether the two techniques provide the same (useful) information, or whether, instead, their recommendation are disjoint, suggesting a possible opportunity of combining the two techniques.
- **RQ3:** *Can association rules and Granger be combined to devise a better change coupling recommender?* This research question analyzes the performances of a hybrid change coupling recommender obtained by combining the two techniques is able to obtain better performances with respect to the association rule or Granger alone.

B. Analysis Method

The analyses reported in Section IV have been performed using the R statistical environment[5]. In particular, we used the *arules* package for mining association rules, and the *msbvar* package for Granger. For association rules, we considered valid rules those achieving a minimum support of 0.01 and a minimum confidence of 0.01, which allows to obtain a pretty comprehensive set of rules.

To evaluate the performances of the two methods (**RQ1**), first, given a set of snapshots S_i, with $i = 1, \ldots, n$, we divide it into two sets $Train \equiv \{S_1 \ldots S_t\}$ and $Test \equiv \{S_{t+1} \ldots S_n\}$. The number of snapshots composing the training and test set for the four systems and the related period of time are reported in Table I. Then, we use the training set to build our models, i.e., to mine association rules and to build the multivariate time series model. After that, for each snapshot in the test set, we perform an evaluation similar to what Zimmermann *et al.* [1] did in their work, also to better compare association rules and Granger, and to investigate their complementariness. That is, we consider as a query Q an artifact that changes within a snapshot, and we are interested to know what other artifacts will change, in the same subsequent k. In this paper we consider $k = 0$, i.e., for each file that has been changed,

[1] *http://www.eclipse.org/mylyn*
[2] *http://www.freebsd.org*
[3] *http://www.mozilla.org/rhino*
[4] *http://www.squid-cache.org*
[5] http://www.r-project.org

we predict—as Zimmermann *et al.* [1]—the set of changes occurring in the same snapshot, and $k = 5$, to investigate the ability of the approach to predict changes occurring in subsequent snapshots also. For each snapshot $S_i \in Test$:

1) The changes occurring in the snapshot are used as queries, thus they are used to make the prediction with association rules or with Granger. For association rules, we select all rules having as left-hand-side the query and rank them by confidence. For Granger, we use the queries as independent variables in the multivariate time series model. Thus, for both association rules and Granger, we use queries to predict the set of predicted changed artifacts PC_i for the next snapshots $S_{i+1}, \ldots, S_{i+k}$
2) Given the set of predicted snapshots PC_i, and given the set of actual changes AC_i occurring in $S_{i+1}, \ldots, S_{i+k}$, we compute the capability of both approaches to produce *accurate* and *complete* sets of change couplings in terms of precision and recall, computed in the top N list of retrieved change couplings ordered by confidence (association rules) of by S-score (Granger):

$$Precision_i = \frac{|PC_i \cap AC_i|}{|PC_i|}$$

and

$$Recall_i = \frac{|PC_i \cap AC_i|}{|AC_i|}$$

Consistently with previous studies [1], when a query returns an empty set, the precision is considered to be 100%. Other than computing precision and recall, we also compute the F-measure, which is the harmonic mean of precision and recall and it is defined as follows:

$$F - \text{measure}_i = \frac{2 \cdot Precision_i \cdot Recall_i}{Precision_i + Recall_i}$$

the information provided by the F-measure is particularly useful in this context, especially because it would penalize cases where a high precision is achieved just because the recall is very low or even zero.

To address **RQ2**, for each snapshot $S_i \in Test$ we consider the correctly predicted changes computed with association rules $PCA_i \equiv PC_{i,arules} \cap AC_i$ and those predicted by Granger $PCG_i \equiv PC_{i,granger} \cap AC_i$, and consider the size of their union $|PCA_i \cup PCG_i|$ and intersection $|PCA_i \cap PCG_i|$, to determine if they help to correctly predict the same (or a similar) set of change couplings or if, instead, they predict different sets.

Finally, **RQ3** is addressed similarly to **RQ1**, i.e., by computing the precision, recall and F-measure of the change couplings recommended by the two techniques.

To analyze the presence of a statistically significant difference among different techniques for different $top-N$ rankings considered, we use the two-way Analysis of Variance (ANOVA), to test whether the precision, recall, F-measure, or the number of true links identified significantly differ (i) among techniques, (ii) for increasing ranking sets ($top-N$) considered, and (iii) if the two factors interact. To evaluate the practical effect of the differences detected we use the Cohen d effect size [14], which indicates the magnitude of the effect of a treatment on the dependent variables. The effect size is considered small for $0.2 \leq d < 0.5$, medium for $0.5 \leq d < 0.8$ and large for $d \geq 0.8$. For independent samples (to be used in the context of unpaired analyses, as in our case), it is defined as the difference between the means (M_1 and M_2), divided by the pooled standard deviation ($\sigma = \sqrt{(\sigma_1^2 + \sigma_2^2)/2}$) of both groups: $d = (M_1 - M_2)/\sigma$.

IV. Results

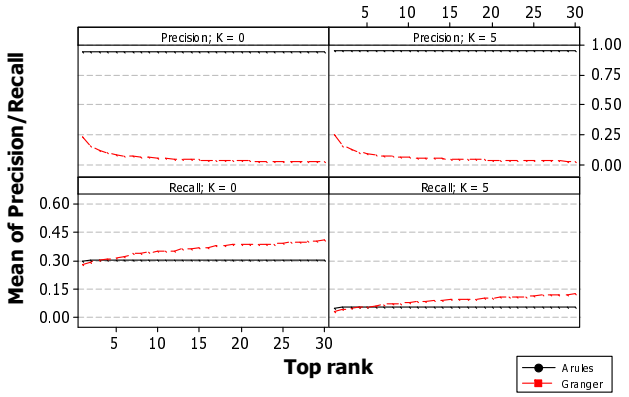
This section reports and discusses results of the empirical study defined in Section III. Raw data and working data sets are available for replication purposes[6].

A. RQ1: What are the performances, in terms of accuracy and completeness of the identified change coupling, of association rules and Granger?

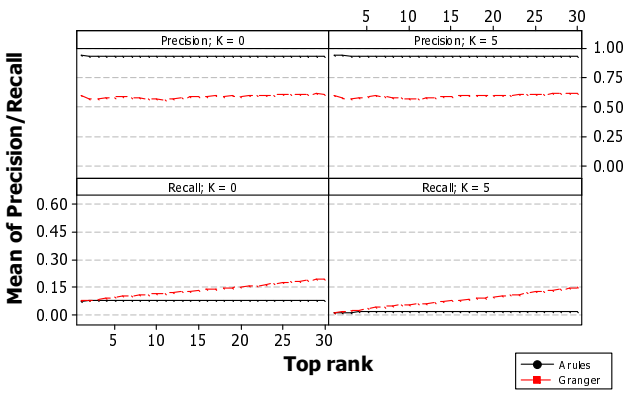
Figure 1 shows the precision and recall of the two methods—association rules and Granger causality—for the four systems. Precision and recall are shown in two lines on the y-axis, while the x-axis indicates that the precision/recall in correspondence of that point is computed across the top N recommendations, ranked by confidence for association rules and by S-score for the Granger causality test. For each system we show results predicting, for each file changed in a snapshot change couplings within the same snapshot (k=0) and within the next 5 snapshots (k=5). Results indicate that, in all cases, the precision is higher for association rules—in agreement with findings of previous works, it tends to be above 70%, and it is actually 100% for FreeBSD-i386 and Squid—than for Granger, where for FreeBSD-i386 it starts at 25% and then decreases, it is around 60% for Squid, is between 20% and 30% for Mylyn, and around 15% for Rhino. In all cases the difference is statistically significant and has an high effect size (p-value < 0.01, $d > 1$), for FreeBSD-i386 and Rhino the precision for both techniques significantly decreases when increasing the set of top N recommendations, and for all systems the differences between the two techniques significantly decrease when increasing the set of top N recommendations considered.

The recall achieved with Granger tends to increase between 10% and 20% for FreeBSD-i386 and Squid when increasing the number of recommendations considered, while for association rules it tends to remain constant, below Granger. In both cases, ANOVA indicates that Granger outperforms association rules (p-value < 0.01), that recall increases with top N, and that also the difference between the two techniques significantly increases when increasing

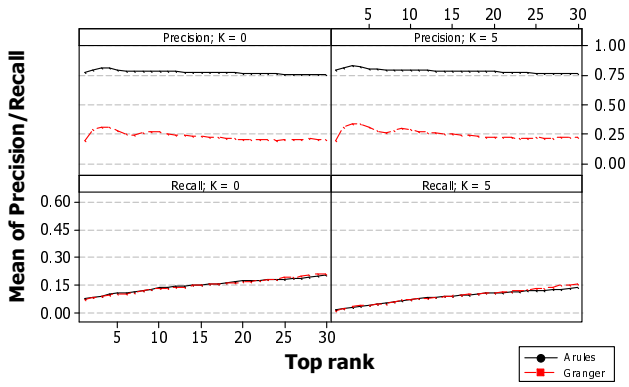
[6]http://www.rcost.unisannio.it/mdipenta/granger-data.tgz



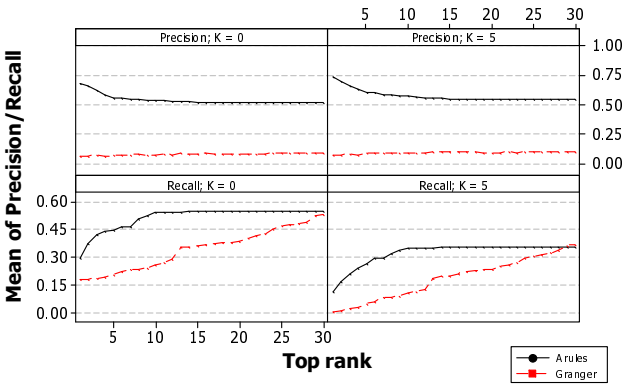
(a) FreeBSD-i386



(b) Squid

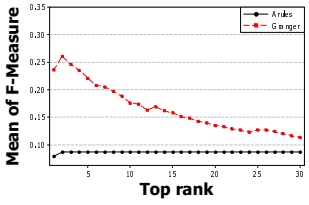


(c) Mylyn

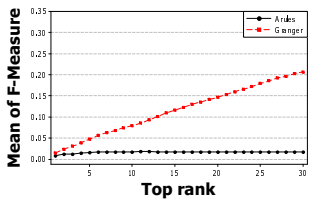


(d) Rhino

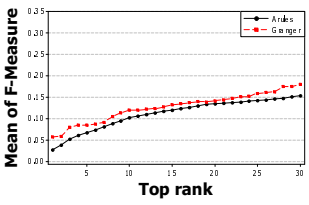
Figure 1. Precision and Recall for change couplings occurring in the same snapshot (k=0) and in the following five snapshots (k=5)



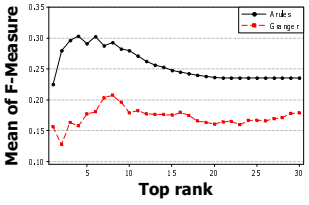
(a) FreeBSD-i386



(b) Squid



(c) Mylyn

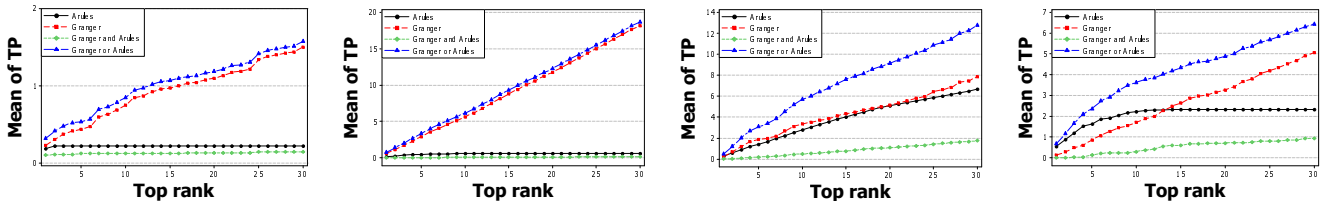


(d) Rhino

Figure 2. F-measure achieved by association rules and Granger causality

the size of the top N list, with an effect size that is medium ($d \sim 0.5$) for the top 3-10 recommendations, and high ($d > 0.8$) for ranked lists larger than 10 recommendations. No differences between the two techniques can be noticed for Mylyn (p-value=0.43), nor any variation of such a difference when varying the size of the recommendation list, while the recall for both techniques clearly increases with the size of the recommendation list for both techniques (p-value < 0.01). For Rhino, the recall is significantly better for association rules than for Granger (p-value<0.01). Also, for association rules the recall increases with few (~ 10) recommendations, and then it remains constant, while it increases slowly for Granger, and for 30 recommendations converges to the results provided by association rules. The effect size of such a difference is initially high ($d \sim 1$) then it decreases and becomes lower than 0.2 for about 20 recommendations.

Overall, results of precision and recall suggest that association rules tend to be more precise, and Granger only in some cases helps to achieve a better recall. The recall often decreases when looking k snapshots ahead instead of looking at changes in the same snapshot (k=0), although in



(a) FreeBSD-i386 (b) Squid (c) Mylyn (d) Rhino

Figure 3. Average number of true positives provided by association rules and Granger causality, their union and intersection

general results for $k = 0$ and $k = 5$ are rather consistent. In the following, we will only report results for $k = 5$.

Figure 2 shows the F-measure achieved for the four systems. For FreeBSD-i386 and Squid, as it can be noticed, the F-measure is significantly higher for Granger (p-value $<$ 0.01), and the difference between the two techniques significantly increases (p-value < 0.01) with the top N, reaching a medium effect size ($d \sim 0.5$) for top 5 recommendations and a high effect size ($d > 0.8$) after 10 recommendations. For Mylyn, no significant difference can be found between the two techniques (p-value=0.44) nor any significant increase/decrease of the difference when increasing the size of the recommendation list (p-value=0.71). For Rhino, the F-measure is higher for association rules (p-value < 0.01), and the difference significantly decreases when increasing the size of the recommendation list (p-value < 0.01), with an effect size initially high ($d \sim 1$) and below 0.2 after 11 recommendations. Overall, the F-measure indicates that in many cases association rules are not necessarily better than Granger causality. That is because, in agreement to what also done in previous studies [1], results of association rules are computed considering as true positive cases where, given a changing artifact, its predicted change coupling is empty, thus achieving a 0% recall where the actual set of change couplings is not empty.

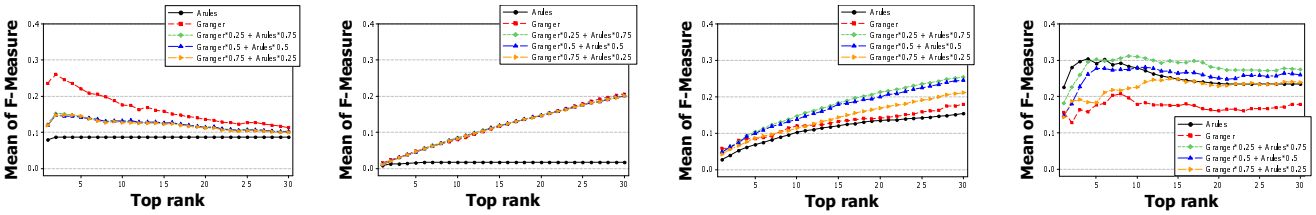
B. RQ2: To what extent do the change couplings identified by Granger overlap with those identified by association rules?

Figure 3 shows the average number of true positive recommendations provided, for each query and for increasing number of top N recommendations, by association rules, by Granger, by the union ("Granger or Arules") and by the intersection ("Granger and Arules") of the two sets. For all systems, we can immediately notice that the intersection is very low and does not increase when increasing the size of the ranked list, thus suggesting that **the two techniques provide complementary results**. Also, it can be noticed that for Squid and FreeBSD-i386 the average number of true positives provided by association rules is very low, always significantly lower than for Granger (p-value < 0.01 high effect size $d > 1$). For Mylyn, the difference is significant though it has a negligible effect size if favor of Granger ($d < 0.2$). For Rhino, association rules perform significantly better than Granger for the top 15 recommendations, with an effect size varying from $d \sim 1$ to $d < 0.2$, then it becomes significantly higher for Granger, with an effect size increasing up to $d = 1.6$.

The union of the two sets of recommendations is slightly higher than Granger for FreeBSD-i386 and Squid, and higher—i.e., able to provide two or three more useful couplings—for Mylyn and Rhino. In all cases the differences are statistically significant (p-value < 0.01 and the effect size is high, $d > 0.8$) except for the top 9 recommendations of FreeBSD-i386, for which the effect size ranges from small ($d = 0.3$) to medium ($d = 0.78$). It has to be noticed that the number of true positive recommendations—on average between zero and two—provided for FreeBSD-i386 is lower than for the other systems due to the lower average number of changes impacting such a FreeBSD subsystem (i386) in each commit (see the fourth column of Table I). In summary, results suggest that, although having a lower precision than association rules, **Granger causality is able to provide a higher number of useful couplings**, which has an important consequence when analyzing change impacts. In fact, for such a kind of tasks a developer could accept the overhead of receiving some additional recommendations, however she would like to obtain a set of change couplings as more complete as possible. Results also suggest that a method combining recommendations would possibly allow for performing a more complete change impact analysis.

C. RQ3: Can association rules and Granger be combined to devise a better change coupling recommender?

Figure 4 shows the F-measure achieved from the combination of the two techniques described in Section II-D—for different weights given to association rules and Granger—compared with the one of the single techniques. For FreeBSD-i386 the F-measure of the combined technique—regardless of the choice of the weights—is lower than the one achieved for Granger, and higher than association rules. Differences among techniques are statistically significant (p-value < 0.01) and significantly vary with top N (p-value$<$ 0.01). Although it might not be perceived from the figure, the effect size of the difference between Granger and the combined technique is small ($d < 0.5$, in many cases



(a) FreeBSD-i386 (b) Squid (c) Mylyn (d) Rhino

Figure 4. F-measure of association rules, Granger causality, and of the combination of the two techniques with different weights

$d < 0.2$), while when comparing the combined technique with association rules, the effect size is medium to high (i.e., d varying between 0.5 and 1 for different top N recommendation lists).

For Squid it can be noticed that the F-measure increases when increasing the size of the top N recommendation list for both Granger and the combination of the techniques (again regardless of the weights). Although ANOVA indicates a significant difference among techniques (p-value¡0.01), and an increase of the difference with the top N (p-value$<$ 0.01) even when comparing Granger with the combination of the techniques only, the effect size of the difference between Granger and the combination is small ($d \sim 0.2$ and below 0.2), while the effect size becomes large $d > 0.8$ for top $N > 4$ when comparing Granger or the combination of the techniques with association rules.

For Mylyn there are significant differences among different techniques and their combinations, as well as significant differences when increasing the set of top N recommendations (ANOVA reported a p-value < 0.01 for the effect of method, i.e., of the used technique, for varying the size of the top N, and for the interaction of the two factors). In this case, when increasing the top N, the combination exhibiting the best performances is the one giving a higher weight (0.75) to association rules and a lower weight (0.25) to Granger, with a medium effect size of the difference—with respect to Granger—(d varying between 0.5 and 0.7).

For Rhino, again there is a significant difference among different techniques and their combinations, as well as significant differences when increasing the set of top N recommendations (p-values < 0.01 as for Mylyn). In this case, we can notice that the combination of the two techniques—in particular again the combination obtained weighting 0.25 Granger and 0.75 association rules—outperforms the other techniques for top $N > 5$, although in this case the effect size of the difference with respect to association rules—which is the second technique performing better—is small ($d \sim 0.2$).

D. When Granger works while association rules do not

In this section we describe a couple of cases, occurring in the CVS commit notes of some systems, suggesting situations where Granger can be used to identify change couplings where association rules could not. For instance, in FreeBSD-i386 on Nov 26, 1999 both `linux_dummy.c` and `linux_sysent.c` (revisions 1.17 and 1.29) were changed by the user *marcel*, and the commit note (*"Implement fdatasync in terms of fsync. The regeneration of proto.h syscall.h and sysent.h was probably forgotten after the last changesyscalls.master."*) mentioned that some changes were probably forgotten. In FreeBSD-i386, many files need to be regenerated to account for new system event protocols. Such task is sometimes forgotten causing subsequent changes, e.g., in `ibcs2_xenix_sysent.c` and `ibcs2_isc_sysent.c`, that are successfully identified by Granger. Instead, association rules cannot detect them because the files do not change together.

In Rhino `IdFunction.java` underwent a change in its constructor signature causing a number of changes in 19 files, within a single change set, to adapt them to the new situation. However, such changes did not resolve all concerning issues, as some files (e.g., `IdScriptable.java`) were changed on month later fixing scope problems with `IdFunction.java` (commit note of user *igor%mir*, June 29, 2004: *"... To prevent null scope initialization problems in future IdFunction constructor now throws an exception if scope argument is null."*).

V. Threats to Validity

This section discusses the main threats to the validity of our study.

Construct validity threats concern the relationship between theory and observation. In this work, construct validity threats can be related to the imprecision of our measurements. In particular, we are aware that we identified changes at a coarse-grained level, i.e., at file level. In principle, changes occurring to two files can affect two unrelated functions/methods. It would be desirable, in future studies, to perform the study reported in this paper at a finer-grained level.

Conclusion validity concerns the relationship between the treatment and the outcome. We use ANOVA to test the presence of significant differences among different techniques and of the interaction between the treatment—i.e., the choice of the technique—and the chosen top N ranking. In addition, we estimate whether detected differences have a practical

meaning using the Cohen d effect size.

Threats to *internal validity* concern factors that can influence our observations. Although both association rule discovery and Granger causality test can statistically infer co-changes between files or temporally consequent changes—as in the case of Granger—this would not allow to claim anything about cause-effect relationships about changes occurring on a file and on those having a change-coupling relation with it.

Threats to *external validity* concern the generalization of our findings. Although we performed our analyses on four different systems, belonging to different domains and developed with different programming languages, we are aware that a further empirical validation on a larger set of systems would be beneficial to better support our findings.

VI. Related Work

As stated by Bohner and Arnold [15] *a major goal of impact analysis is to identify the software work products affected by proposed changes*.

Most of the existing change impact analysis techniques aim at exploiting the presence of dependencies in the source code, identified by means of static analysis [2], dynamic analysis [3], or specific techniques such as static and/or dynamic slicing [16]. Some impact analysis techniques cope with problems specific of particular kinds of artifacts—for example, UML models [17]. There is a large corpus of studies related to change impact analysis, however a complete survey of them is beyond the scope of this paper.

As mentioned in the introduction, one limitation of existing impact analysis techniques is that they work assuming the presence of dependencies between artifacts. Alternative approaches exist to overcome such a limitation. Some of them [4], [18] are based on information retrieval, i.e., they exploit the textual content of the artifacts, assuming that a change to a software artifact will impact other, textually similar artifacts. The weakness is that these approaches might fail to find pertinent links when the similarity is low—while artifacts are, indeed, related—or might find false positives when un-related artifacts are textually similar. Other approaches that do not rely on code dependencies are based on expert judgment and code inspection [19]: however, several studies have shown that expert predictions are frequently incorrect or at least biased by subjectiveness [20], and source code inspection can be prohibitively expensive [21].

The first studies aimed at identifying logical change couplings were performed by Gall *et al.* first on change releases of a telecommunication system [22] and then on commit histories extracted from CVS logs [23].

To overcome the limitations of the previous change impact analysis approaches, and above all to complement the recommendations that could be provided by traditional change impact analysis approaches, two approaches were developed in parallel by two different research groups, namely Ying *et al.* [5] and Zimmermann *et al.* [1], [6]. Both use association rules discovery, a well-known data-mining practice—that we summarized in Section II-B—to determine sets of files that were changed together frequently in the past from the change history of the code base. The hypothesis is that the change patterns inferred by means of association rules—i.e., files co-changing in the same change set—can be used to recommend potentially relevant source code to a developer performing a change. They found that in many cases the precision in the performed prediction is often above 70%, and in some cases higher than 90%, while the recall often lower than 25%, and in some cases below 10%.

In a previous paper [7] we introduced the idea of using the multivariate time series for predicting the impact of a change. This paper continues the early work previously presented as follows:

- we present an empirical evaluation—through changes from four software systems—of Granger causality test, its comparison with association rule discovery, and the overlap of their results. The previous work only showed the applicability of the approach on a subsystem of the Samba[7] project (*sbmd*) composed of about 30 files only.
- to train the multivariate time series model in a way that provides the "strength" of the change coupling relation, we use file change frequencies, instead of Boolean variables indicating whether or not files changed.
- we define a hybrid approach that combines ranking of both association rules and Granger.

VII. Conclusion and Work-in-Progress

In recent years, Association rule discovery [11] has been successfully applied to predict change couplings among files by mining data from software repositories [1], [5]. This paper performs an empirical comparison of association rule discovery with a technique based on multivariate time series analysis, and specifically on the Granger causality test [8]. Results of an empirical study performed on change data extracted from CVS repositories of four different software systems—FreeBSD-i386,Mylyn, Squid, and Rhino—show that: (i) overall, association rule discovery exhibit a higher precision than Granger causality test, while the recall of Granger causality test is, in most cases, higher for Granger causality or at least comparable; and (ii) the number of true recommendations provided by Granger causality test is higher than for association rules, and above all, the two techniques provide a set of recommendations having a very low intersection.

The above results suggest the opportunity of combining the two techniques. A hybrid technique obtained by combining ranking scores provided by association rules and by Granger causality allow to obtain (i) a F-measure and a recall

[7] http://www.samba.org

higher than the two techniques alone, and (ii) a precision in-between the two. In summary, the performed study suggests the potential of multivariate time series analysis to suggest change couplings complementary to those provided by association rules, and the advantages of combining the two techniques.

Work-in-progress aims at (i) using enhanced ways of combining the two techniques, (ii) further validating the combined techniques through more case studies as well as by investigating how changes tend to be propagated in projects having a different organization, and (iii) better understanding the nature of change coupling inferred by Granger causality test as opposed to those inferred by mining association rules.

REFERENCES

[1] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, 2004, pp. 563–572.

[2] R. S. Arnold and S. A. Bohner, "Impact analysis - towards a framework for comparison." in *Proceedings of the Conference on Software Maintenance, ICSM 1993, Montréal, Quebec, Canada, September 1993*, 1993, pp. 292–301.

[3] J. Law and G. Rothermel, "Whole program path-based dynamic impact analysis," in *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*. IEEE Computer Society, 2003, pp. 308–318.

[4] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," in *11th IEEE International Symposium on Software Metrics (METRICS 2005), 19-22 September 2005, Como Italy*. IEEE Computer Society, 2005, p. 29.

[5] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining revision history," *IEEE Transactions on Software Engineering*, vol. 30, pp. 574–586, Sep. 2004.

[6] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Trans. Software Eng.*, vol. 31, no. 6, pp. 429–445, 2005.

[7] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta, "An eclectic approach for change impact analysis," in *Proceedings of the ACM/IEEE 32rd International Conference on Software Engineering (ICSE 1010) - New Ideas and Emerging Results (NIER) Track, 2-8 May 2010, Cape Town, South Africa (to appear)*. ACM Press, 2010, *http://www.rcost.unisannio.it/mdipenta/papers/nier2010.pdf*.

[8] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969.

[9] N. D. Mukhopadhyay and S. Chatterjee, "Causality and pathway search in microarray time series experiment," *Bioinformatics*, vol. 23, no. 4, pp. 442–449, 2007.

[10] A. Hindle, M. W. Godfrey, and R. C. Holt, "Mining recurrent activities: Fourier analysis of change events," in *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Companion Volume*, 2009, pp. 295–298.

[11] R. Agrawal, T. Imielinski, and A. N. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*. ACM Press, 1993, pp. 207–216.

[12] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, January 1994.

[13] J. H. Lee, "Combining multiple evidence from different properties of weighting schemes," in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM, 1995, pp. 180–188.

[14] D. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures (fourth edition)*. Chapman & All, 2007.

[15] R. Arnold and S. Bohner, *Software Change Impact Analysis*. Wiley-IEEE Computer Society, 1996.

[16] M. Kamkar, "An overview and comparative classification of program slicing techniques," *J. Syst. Softw.*, vol. 31, no. 3, pp. 197–214, 1995.

[17] L. C. Briand, Y. Labiche, L. O'Sullivan, and M. M. Sówka, "Automated impact analysis of UML models," *Journal of Systems and Software*, vol. 79, no. 3, pp. 339–352, 2006.

[18] A. Chen, E. Chou, J. Wong, A. Y. Yao, Q. Zhang, S. Zhang, and A. Michail, "CVSSearch: Searching through source code using CVS comments," in *ICSM '01: Proceedings of 17th IEEE International Conference on Software Maintenance*, 2001, p. 364.

[19] M. Lindvall and K. Sandahl, "Practical implications of traceability," *Software—Practice and Experience*, vol. 26, no. 10, pp. 1161–1180, Oct. 1996.

[20] ——, "How well do experienced software developers predict software change?" *J. Syst. Softw.*, vol. 43, no. 1, pp. 19–27, 1998.

[21] S. L. Pfleeger, *Software Engineering: Theory and Practice*. Upper Saddle River, NJ: Prentice-Hall, 1998.

[22] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings of the International Conference on Software Maintenance, ICSM 98*, 1998, pp. 190–197.

[23] H. Gall, M. Jazayeri, and J. Krajewski, "CVS release history data for detecting logical couplings," in *6th International Workshop on Principles of Software Evolution (IWPSE 2003), 1-2 September 2003, Helsinki, Finland*. IEEE Computer Society, 2003, pp. 13–23.