Software Quality Journal (2019) 27:1481–1503
https://doi.org/10.1007/s11219-019-09456-3

Pieces of contextual information suitable for predicting co-changes? An empirical study

Igor Scaliante Wiese[1] · Rodrigo Takashi Kuroda[2] · Igor Steinmacher[3] · Gustavo Ansaldi Oliva[4] · Reginaldo Ré[1] · Christoph Treude[5] · Marco Aurelio Gerosa[3]

Published online: 3 July 2019

Abstract
Models that predict software artifact co-changes have been proposed to assist developers in altering a software system and they often rely on coupling. However, developers have not yet widely adopted these approaches, presumably because of the high number of false recommendations. In this work, we conjecture that the contextual information related to software changes, which is collected from issues (e.g., issue type and reporter), developers' communication (e.g., number of issue comments, issue discussants and words in the discussion), and commit metadata (e.g., number of lines added, removed, and modified), improves the accuracy of co-change prediction. We built customized prediction models for each co-change and evaluated the approach on 129 releases from a curated set of 10 Apache Software Foundation projects. Comparing our approach with the widely used association rules as a baseline, we found that contextual information models and association rules provide a similar number of co-change recommendations, but our models achieved a significantly higher F-measure. In particular, we found that contextual information *significantly reduces the number of false recommendations* compared to the baseline model. We conclude that contextual information is an important source for supporting change prediction and may be used to warn developers when they are about to miss relevant artifacts while performing a software change.

Keywords Co-change prediction · Logical coupling · Change coupling · Change propagation · Change impact analysis · Social factors · Contextual information

1 Introduction

Changes are part of software development. Developers modify artifacts to fix defects, add new features, or improve existing source code. In order to make the necessary changes to fulfill a task (e.g., a change request), developers often execute manual and time-consuming tasks. Co-change prediction approaches have been proposed to support developers while they perform software

✉ Igor Scaliante Wiese
igor@utfpr.edu.br

Extended author information available on the last page of the article

changes (Bohner and Arnold 1996; Zimmermann et al. 2005; Hassan and Holt 2004). These approaches are based on the premise that if there is coupling between two files in a release, these files are prone to co-change in the consecutive release.

Predicting co-changes can be useful to avoid incomplete changes by notifying the developers about artifacts that are likely to change together (Hassan and Holt 2004), and to help newcomers complete their first contribution, especially when newcomers have little knowledge about the source code and the software architecture (Steinmacher et al. 2016).

Approaches that have been proposed to predict co-changes often rely on source code analysis, such as dynamic analysis (Orso et al. 2004), static analysis (Briand et al. 1999), frequent past changes and change coupling analysis (Gall et al. 1998; Ying et al. 2004; Zimmermann et al. 2005), and conceptual analysis (Gethers and Poshyvanyk 2010; Revelle et al. 2011). Other approaches combine these techniques into hybrid methods (Hassan and Holt 2004; Gethers et al. 2012; Kagdi et al. 2013; Dit et al. 2014). However, despite the advances in this area, the number of false recommendations is still high, presumably because the couplings do not adequately capture the situations in which the artifacts change together (Canfora et al. 2014; Oliva and Gerosa 2015a). Contextual information may help to characterize the changing context, improving the performance of the prediction models.

Developers change software artifacts for various reasons and the context involved in the changes may indicate the conditions in which two artifacts are prone to co-change. To investigate this hypothesis, we built prediction models at the file level for each pair of artifacts using contextual information from one release to predict if, in issues of the consecutive release, the two artifacts would change together. Our investigation sheds light on the possibility of using information about the context in which the software change occurred to reduce the number of false recommendations and to improve the effectiveness of co-change prediction. Contextual information is not considered by current approaches, yet it can be beneficial since software artifacts are changed for different reasons (Oliva et al. 2013; Canfora et al. 2014; Oliva and Gerosa 2015b).

In previous work (Wiese et al. 2015, 2016), we conducted an exploratory study with two projects by using random forest classifiers trained with contextual information from past changes to improve the co-change prediction. We relied on the concept of change coupling to select the pairs of files most likely to co-change and to compare our prediction model to an association rule model (Oliva and Gerosa 2015b). In this paper, we build upon our previous work to investigate the extent to which contextual information correctly predicts co-changes in a larger sample of projects and without limiting the analysis to the top 25 co-changes. We analyze whether our approach leads to predictions that are more accurate compared to a baseline model based on association rules (using different support and confidence thresholds). In addition, we contacted developers from the studied projects and asked them to inspect the results and discuss their perspective about adopting the proposed approach in practice.

We analyzed 10 open source projects and 129 releases. Overall, we found that contextual information extracted from issues, developers' communication, and commit metadata enables a highly accurate prediction of co-changes, correctly predicting 19,746 out of 26,189 co-changes (75%). Association rules covered 33% of all possible co-changes, while contextual information models covered 25%. However, association rules issued more wrong recommendations than contextual information models (111 k vs 16 k). We also found that contextual information models based on numeric metrics can predict many more co-change instances and can be used to evaluate more instances (137 k vs 19 k), but that categorical information can improve the accuracy of the prediction models by an average of 12% of recall and 23% of precision. These results suggest that our model can be leveraged for the development of novel co-change prediction tools to support software evolution and maintenance.

2 Study design

In this section, we present our research questions and their rationale (Section 2.1), followed by an overview of our approach, including the data collection steps, the way we selected and tagged each co-change to build the prediction models, and the evaluation method (Section 2.2). Finally, we list and describe the studied systems (Section 2.3).

2.1 Research questions

Previous work has shown that prediction models can be built to predict co-change occurrences (Zimmermann et al. 2005). We conjecture that it is possible to improve these models by using contextual information collected from issues, developers' communication, and commit metadata. We aim to investigate how many co-change instances can be correctly predicted by contextual information models compared to a baseline model built with association rules, which is widely used in the literature (Ball et al. 1997; Ying et al. 2004; Zimmermann et al. 2005; Gethers et al. 2012; Kagdi et al. 2013). Hence, we formulate our first research question as follows:

> (RQ1) How does co-change prediction based on contextual information models compare to association rules in terms of accuracy and coverage?

To determine the accuracy of the prediction models, we check whether they *only* suggest co-changes that indeed occurred in a specific commit (precision) and whether *all* co-changes that occurred in a commit are suggested (recall). More specifically, we compare the F-measure (harmonic mean between precision and recall) achieved by the two models under different experimental settings. We also compare the models in terms of their *coverage*, which we calculate as the ratio of co-changes that we can correctly predict using each approach compared with the number of co-changes that occurred in each release.

Knowing how well a given model performs is not the only criterion governing its adoption. From a practical perspective, it is important to reason about the cost of collecting the data required to build the model. This aspect is particularly important in the domain of this study since pieces of contextual information might come from various sources (e.g., version control and issue tracking systems) or might involve intensive computation to be obtained (e.g., building a communication network from development discussion threads). Therefore, discovering which key features (metrics) enable building models with significantly less effort is important in practice. Determining key features may also serve as an input to drive new theories about the reasons behind change coupling. This reflection leads to our second research question:

> (RQ2) What are the most influential kinds of contextual information when predicting co-changes?

2.2 Approach overview

Figure 1 shows an overview of our approach. The approach is split into two main parts: (i) compute association rules from a release to predict co-changes in the consecutive release, and (ii) extract contextual information metrics and build a classifier for each antecedent file found by association rules to predict co-changes in the consecutive release. Our dataset and scripts are available on Zenodo.[1]

As an illustrative example, let us assume that we want to predict co-changes between two files, namely JMSConduit.java (File A) and JMSOldConfigHolder.java (File B) from the Apache CXF project. Calculating the frequency of changes during release 2.0, we find that File A changed 33 times, including 15 times when it changed together (co-changed) with File B. Based on this historical information, it seems reasonable to infer that both files are prone to change together in the consecutive release (2.1). Indeed, the frequency of past changes would correctly predict co-changes in 19 commits. However, in other 26 commits, File A changed, but without a corresponding change in File B. In this case, the frequency-of-past-changes approach would yield 26 false recommendations.

In our approach, we collect contextual information for each commit that includes File A in the release 2.0 and builds a prediction model for the pair File A and B because they frequently changed together in the past. This model indicates, for release 2.1, 17 co-changes between both files, and 19 cases in which File A changed without File B. Considering this example, our approach wrongly predicts 7 co-changes between both files (false positives) and 2 cases in which File A changed without File B (false negatives), but it correctly predicts 36 commits. Comparing our approach to the association rules approach in this example, contextual information reduced the number of false recommendations by 65%. For this example, the most influential contextual information used by our model was the number of lines changed, the number of words used to describe an issue, and who reported the issue.

In the following, we describe each step in more detail.

Step (1) Collecting issues and commits

We used two data sources: Version Control Systems (VCS) and Issue Tracking Systems (ITS). Issues (e.g., change requests and bug reports) are often logged in an Issue Tracking System (ITS), such as Bugzilla or JIRA and have a unique identifier (ID). This ID helps identify the commits in the version control system associated with an issue. We extracted data from the issues and commits of the studied software projects. We used Bicho[2] to parse and collect all issues from JIRA ITS. To collect data from VCS archives, we used the CVSAnalY[3] tool.

Since an issue might be resolved after several commits, to avoid missing cases of co-changes related to an issue, we grouped commits that addressed the same issue. To link issues and commits, we searched the commit messages for the expression “project name” + “issue_number” (e.g., Hadoop-1000), since this pattern is often used by Apache projects. We also checked if the commits were made while the issue was holding the status open and if the status changed to “fixed” afterward.

[1] Dataset and scripts are available at https://zenodo.org/record/2635857
[2] https://github.com/MetricsGrimoire/Bicho
[3] https://github.com/MetricsGrimoire/CVSAnalY

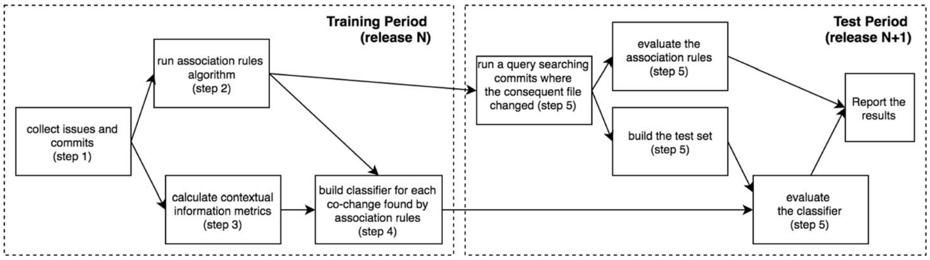
Training Period (release N)
Test Period (release N+1)
collect issues and commits (step 1)
run association rules algorithm (step 2)
calculate contextual information metrics (step 3)
build classifier for each co-change found by association rules (step 4)
run a query searching commits where the consequent file changed (step 5)
evaluate the association rules (step 5)
build the test set (step 5)
evaluate the classifier (step 5)
Report the results

Fig. 1 Approach overview

Step (2) Applying the association rules algorithm

An association rule is an implication of the form I ⇒ J, where I and J are two disjoint sets of items (a.k.a., item sets). A relevant rule I ⇒ J means that when I occurs, J is likely to co-occur. In this study, a rule I ⇒ J means that J is change-coupled to I. We also consider that I and J are file sets composed by one single file, where I = {fi} and J = {fj} and fi ≠ fj. The relevance of association rules can be measured according to several metrics. In this study, we employ the metrics of support and confidence, which have been extensively used in previous Software Engineering research studies (Zimmermann et al. 2005; Moonen et al. 2016; Rolfsnes et al. 2016).

For each release, we calculated all possible rules involving pairs of files. Since we wanted to compare our results to the association rules approach, we filtered the co-changes by values of support and confidence. We collected co-changes with support 2, 3, 4, 5, 6, 7, 8, and higher than 8. For each support value, we applied confidence thresholds ranging from 50 to 70%, 71–90%, and 91–100%. We used these thresholds based on previous work and after analyzing the distribution of support and confidence values identified in each project. We also filtered out co-changes with support of less than two because a unitary weight does not reflect how often two classes usually change together (Beyer and Noack 2005).

Step (3) Calculating contextual information metrics

To build the prediction models, we used metrics calculated from contextual information of issue reports, developers' communication, and commit metadata, as described in the following.

Issue context We hypothesize that some co-changes are more likely to happen when fixing a bug, while others appear when implementing new features. The assignee works on issues related to specific parts of the software, and an issue reported by the same reporter might involve the same files since the reporter might be interested in some specific requirements. The metrics defined for this dimension are the following: was the issue reopened? (categorical), issue type (categorical), issue assignee (categorical), and issue reporter (categorical).

Communication context Discussion characteristics can indicate how proneness files co-change. For example, some co-changes can happen in issues with more messages or more words (wordiness), either because the issue is difficult to understand or because the files necessary to fix this issue are complex. The metrics defined for the communication context are number of issue comments, number of issue discussants, number of words in the discussion, and number of distinct developers.

Developer's role in communication Developers involved in a discussion have different values of betweenness and closeness. Previous work has shown the importance of these metrics in other software engineering problems (Bird et al. 2009). We calculated the closeness and betweenness centrality for this dimension based on Wasserman and Faust (1994).

Structural hole of communication Structural hole metrics denote gaps between nodes in a social network and represent that people on either side of the hole have access to different flows of information, indicating that there is a diversity of information flow in the network. In previous work, we successfully used structural holes to identify recurrent change couplings (Wiese et al. 2014b). In this sense, these metrics represent a way to analyze the communication network revolving around software co-changes. The metrics are constraint, hierarchy, effective size, and efficiency. We calculated these metrics based on Wassermann and Faust (1994).

Communication network properties Network properties indicate aspects of how the social network is organized. Networks with more arcs indicate more message exchange intensity. Networks with more nodes indicate greater involvement of developers. The social network property is useful for predicting defects (Bird et al. 2009; Conway 1968). The metrics are size, ties, diameter, and density. We calculated these metrics based on Wassermann and Faust (1994).

Commit context Code churn or a specific operation (add or delete) on lines of codes can indicate specific aspects for different co-changes. The metrics are committer (categorical), # of lines of code added, # of lines of code deleted, code churn, and is the committer the file owner? (categorical).

Step (4) Building classifiers

Training/test set separation For the validation of our prediction models, we went through all releases of each project, building the training set in one release and using the changes that occurred in the consecutive release as a test set. Table 1 presents an example of the training set to predict co-changes between JMSConduit.java and JMSOldConfigHolder.java.

The column "Pair of files" indicates the co-change analyzed. In this example, File A (JMSConduit.java) is the antecedent and File B (JMSOldConfigHolder.java) is the descendent identified by the relevant association rule. All metrics are computed for File A. In this sense, the set of metrics was extracted from each commit and issue (indicated in the corresponding columns) in which file A (JMSConduit.java) was changed.

Tagging co-changes To tag each co-change, we looked at the descendent. Each commit containing File A (antecedent) was checked, and when the commit propagated changes to File B (descendent), we assigned the commit to class "1." Otherwise, we assigned it to class "0" to indicate that only File A was changed in the commit. Therefore, the value 1 indicates that the antecedent (File A) and the descendent (File B) were committed together. The first 4 lines indicate that file A had 4 commits in 4 distinct issues, but only commits 50 and 220 (2 and 4 line in Table 1) propagated changes to File B (class column = 1).

Table 1 An example of a training set built with metrics collected from JMSConduit. Java changes to predict when the co-change with JMSOldConfigHolder.java is likely to occur

Pair of files	# commit	# issue	Set of metrics from contextual information (issues, developers' communication, and commit)	Co-change
JMSConduit.java – JMSOldConfigHolder.java	1	1760	Issue Type = Bug, Issue Reopened = 0, Assignee = ffang, eporter = ffang, # of commenters = 3, # of dev commenters =2, wordiness =438 …	0
JMSConduit.java – JMSOldConfigHolder.java	50	1832	Issue Type = Improvement, Issue Reopened = 0, Assignee = chris@die-schneider.net, Reporter = chris@die-schneider.net, # of commenters = 3, # of dev commenters = 2, wordiness =764 …	1
JMSConduit.java – JMSOldConfigHolder.java	72	2207	Issue Type = Bug, Issue Reopened = 0, Assignee = njiang, Reporter = liucong, # of commenters = 2, # of dev commenters = 1, wordiness =311 …	0
JMSConduit.java – JMSOldConfigHolder.java	220	2316	Issue Type = Improvement, Issue Reopened = 0, Assignee = dkulp, Reporter = marat, # of commenters = 1, # of dev commenters = 0, wordiness =43 …	1
JMSConduit.java – JMSOldConfigHolder.java	…	…	…	...

Classifier construction For each release of each project analyzed, we generated a .csv file to use as a training or test set. The release version N is used as a training set and the release version N+1 is used as a test set. We ran the random forest technique to construct classifiers to predict the co-changes.

Random forest is frequently used in classification problems since the models can be used with large and small datasets, and it also can handle problems of missing data (Breiman 2001). Random forest has been used in several previous software engineering studies and tends to have good predictive power (Lessmann et al. 2008; McIntosh et al. 2014; Dias et al. 2015; Macho et al. 2016). The random forest technique builds a large number of decision trees at training time using a random subset of all the attributes. Using an aggregation of votes from all trees, the random forest technique decides whether the final score is higher than 0.5 to determine if a co-change will be predicted as true. We implemented our classifiers using the R package Caret (Kuhn 2008).

Step (5) Evaluating the co-change prediction performance

To evaluate our classifier, we used training and test sets generated for each release of each project under study. For each release, we found the relevant association rules, considering a broad range of support and confidence values.

Taking Table 1 as an example, we have an association rule JMSConduit ⇒ JMSOldConfigHolder, meaning that "when JMSConduit is changed, JMSOldConfigHolder is likely to change." This implies a change coupling from the right-hand side (RHS) to the left-hand side (LHS), i.e., changes in LHS often imply changes to the RHS. Thus, in the consecutive release (test set), we evaluate the commits in which LHS changes because we want to know what its impact on RHS is (co-change vs no co-change).

The performance of the classifier was measured in terms of recall, precision, F-measure, and Mathews Correlation Coefficient (MCC). Below, we describe each one of them:

Recall: We calculated recall to identify the proportion of instances that the model nominated for changing together and those actually changed. To obtain the recall value, we used the following formula: TP/TP+FN.

Precision: We measured precision to identify the rate of predicted co-changes that have actually changed together. To obtain the precision value, we used the following formula: TP/TP+FP.
F-measure is the harmonic mean of precision and recall. We used the following formula: 2×(precision × recall)/(precision + recall).
Mathews Correlation Coefficient (MCC) is a correlation coefficient between the observed and predicted classification. This measure takes into account TP, FP, TN, and FN values and is generally regarded as a balanced measure, which can be used even if the classes are unbalanced. Because we found many imbalances in our data, we do not report the area under the curve, which can be a biased measure (Powers 2011). For binary classification tasks, MCC has attracted the attention of the machine learning community as a method to summarize the confusion matrix into a single value (Powers 2011). An MCC coefficient of +1 represents a perfect prediction and 0 means a random prediction between prediction and observations. We calculated MCC using the following expression (Powers 2011):

$$\mathrm{MCC} = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

Finally, to study the most influential contextual information, we computed Breiman's variable importance score (Breiman 2001) for each numeric contextual information: the larger the score, the greater the importance of the contextual information. We used the variable importance function (varImp) of the Caret R package to obtain normalized Z-scores (scale = TRUE); thus, the values returned by each prediction range from 0 to 100.

2.3 Studied software projects

We studied 10 Apache Software Foundation projects. We selected projects from different domains and with different numbers of lines of code (LOC), numbers of active developers (#Devs), and activity levels according to OpenHub[4] data. All projects have more than 300 k of LOC, at least 36 developers contributing actively, and generally very high activity levels. All projects were implemented in Java and had XML files.

Table 2 summarizes the data collected from each project. Cloudstack and Solr are newer projects, thus, both projects have fewer data. In total, we used data from around 24 k issues and 54 k commits linked to issues, and we analyzed 7 k distinct association rules over 129 releases. The collected commits represent 26% of the total number of commits found in the version control system of the subject systems.

3 Results

3.1 (RQ1) How do co-change predictions based on contextual information models compare to association rules in terms of accuracy and coverage?

We built a classifier for each co-change following the approach described in Section 2. It is important to mention that we built two different types of classifiers using contextual

[4] OpenHub can be accessed at https://www.openhub.net

Table 2 Studied projects

Projects	Releases	Issues	Commits linked to issues	Association rules
Camel	22	3494	7765 (30.54%)	479
Cassandra	14	1597	2405 (14.18%)	519
Cloudstack	4	556	1382 (3.36%)	47
CXF	9	3139	8308 (40.95%)	1464
Derby	11	2605	7469 (69.80%)	1547
Hadoop	22	2453	4306 (12.33%)	363
Hbase	14	5494	13,372 (64.59%)	1954
Hive	12	2014	2393 (32.81%)	310
Lucene	17	2032	6433 (24.07%)	294
Solr	4	418	555 (15.99%)	54
Total	129	23,802	54,388	7031

information: (i) ACI models, which used all contextual information metrics calculated in each release, and (ii) NCI models, which only considered numeric contextual information metrics. The reason to test both models is related to applicability; ACI models can be less useful, since a value for categorical information can be present in the test set, but not in the training set.

To answer RQ1, we calculated how many co-changes we could correctly predict compared to the number of correct predictions made with association rules with support ≥ 2 and confidence value $\geq 50\%$. We used these thresholds based on the literature that used association rules to infer change couplings (Zimmermann et al. 2005; Bavota et al. 2013). It is important to mention that the association rule model is widely used in the literature and can be considered a baseline model (Ball et al. 1997; Ying et al. 2004; Zimmermann et al. 2005; Gethers et al. 2012; Kagdi et al. 2013; Dit et al. 2014).

In Table 3, we report the confusion matrix for each technique. TP is the number of right recommendations for a single co-change, similar to the practical scenario described in Section 2.2. TN values are related to recommendations in which the co-change did not occur but only one file changed in a specific commit.

We found that models built using numeric contextual information correctly predicted 19 k co-changes covering 24.62% of all possible co-changes that happened in all tested releases. Association rules covered 32.94% of all co-changes, correctly predicting 26 k co-changes. The coverage of the former ranged from 2.54% (SolR) to 68.76% (Derby). Association coverage ranged from 5.42 (SolR) to 79.70% (Derby). The models built with numeric contextual information and association rules were able to test all 137 k co-changes that happened in 54 k commits.

Inspecting the dataset, we observed that models built with all contextual information (ACI) discarded commits from the test set for which string values were not found in the training set, e.g., if a new developer committed a file in release 2.1 (test set) but not in release 2.0 (training set). Because of this, using categorical information combined with numeric information

Table 3 Confusion matrix values for contextual information and association rules models

Models	TP	FP	FN	TN
ACI	5264	765	652	13,288
NCI	19,746	10,007	6443	101,167
AR	26,189	111,174	0	0

Table 4 Overview of evaluation metrics

Models	Recall×	Precision×	F-1×	MCC×	% FP	% FN
ACI	0.87	0.89	0.88	0.83	14.53	4.91
NCI	0.66	0.75	0.71	0.63	51.31	6.49
AR	0.19	1.00	0.32	N/A	81.17	0.00

×Average values over all releases evaluated

reduced the number of analyzed co-changes to 6.62% of all co-changes, correctly predicting 5 k co-changes.

To evaluate the quality of the predictions, we compared the values of precision, recall, F-measure, and MCC by project, support, and confidence thresholds. Table 4 presents an overview of the evaluation metrics. Recall, precision, F-measure, and MCC indicate how accurately the all contextual information, numeric contextual information, and association rules models can predict co-changes. We could observe that all contextual information models (ACI) have higher values of MCC and F-measure; however, they have fewer true positives than numeric contextual information (NCI) and association rules (AR) models.

We found that numeric contextual information models have a value of F-measure twice as high as that of association rules models. In terms of correct recommendations, numeric contextual information models issued 1 correct co-change recommendation every 1.38 attempts. Association rules issued 1 correct co-change recommendation every 5.24 attempts. We also observed that NCI models have 57.8% false alarms (% FP + % FN). Comparing the number of false alarms with AR models, we observed that NCI models have 23.37% fewer false alarms, leading to 30,828 commits with fewer false alarms. MCC shows the correlation between what we observed in each commit and what we predicted for each commit. Contextual information showed a high correlation between observed and predicted values. MCC can be interpreted similarly to other types of correlation, e.g., Pearson and Spearman. In this sense, we consider that values higher than 0.6 indicate strong correlation and values higher than 0.8 indicate very strong correlation. We could not calculate MCC values for AR since these models are used only to recommend co-changes.

Figure 2 presents the F-measures for all contextual information models (first boxplot), association rules models (second boxplot), and numeric contextual information models (third boxplot). We notice that the boxplots for the model using the all-contextual-information range from 0.0 to 1.0 in six cases (Camel, Cassandra, Hadoop, Hive, Lucene, and Solr). These models can thus be very good or very bad when predicting co-changes. However, the median F-measure of these models was higher in 7 out of 10 projects when compared to the F-measure from numeric contextual information and association rule models.

Observing the median values, association rule models performed better than numeric contextual information and all contextual information models in the Camel project. Numeric contextual information models performed better than all contextual information and association rules models in the Cassandra and Solr projects. In five projects (Cassandra, CXF, Derby, Hbase, and Lucene), models based on contextual information performed better than association rule models. Especially in CXF and Derby, the difference of quality is very evident.

Figure 3 presents the F-measure values to predict co-changes for different thresholds of support and confidence. We chose these values based on the literature (Zimmermann et al. 2005; Bavota et al. 2013) and the analysis of the distribution of these values. In the graph depicted in Fig. 3, the black line represents the all-contextual-information models, the light

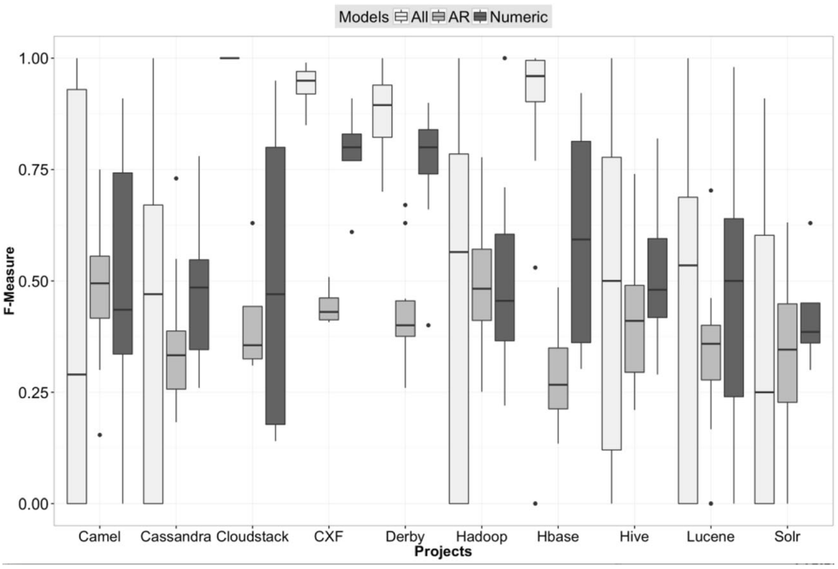

Fig. 2 F-measure values to predict co-changes by project considering all releases

gray represents the numeric contextual information models, and the gray line indicates the association rules models. The last line indicates the percentage of co-changes that can be predicted using only the specific threshold of support and confidence.

The F-measure for numeric contextual information models values ranged from 0.56 to 0.86. The F-measure for association rules models values ranged from 0.26 to 0.50. Numeric contextual information models had more true positives than the all-contextual-information models, but the latter had higher F-measures.

Association rules models obtained their best F-measure when support was at least 8 and confidence was at least 70%. However, using only these rules to predict when the expected co-change occurred, the coverage decreases considerably.

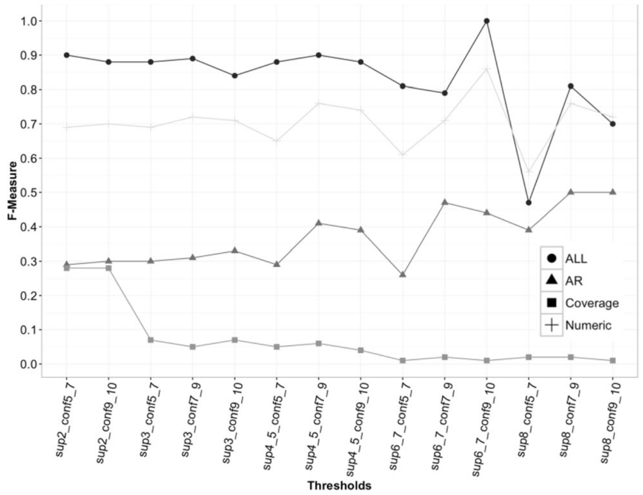

Fig. 3 Average F-measure values to predict co-changes by thresholds of support and confidence

Models based on contextual information improved the quality of co-change prediction (F measure values) compared to the association rules model. Even with high values of support and confidence, contextual information models outperformed association rules.

3.2 (RQ2) What are the most influential kind of contextual information when predicting co-changes?

As we mentioned in Section 2.2—Step (5), we computed Breiman's variable importance score (Breiman 2001) to determine the influence of each predictor. In Table 5, we report the average variable importance score considering the numeric contextual information models. The gray cells highlight the TOP 5 types of contextual information for each project. It is hard to define the best subset, since all types of contextual information were frequently used by random forest to predict co-changes. However, we can observe that six metrics (#3, #4, #6, #15, #16, and #17) obtained the best scores across all projects.

We also analyzed the performance considering only the three best metrics from the commit context (metrics #15, #16, and #17—Table 5) against the three best metrics from the social information dimension (metrics #3, #4, and #6—Table 5). We found a small effect size difference in favor of commit context when the F-measure average was compared (56% vs. 50%) across all projects analyzed.

The results suggest that, to identify whether a co-change will occur in a specific commit, it is important to analyze the number of lines added and removed and the code churn. On the other hand, it is also important to use information from social aspects related to the number of comments and how "close" the developers are in the communication network. Closeness is an indicator of how spread the information is between the developers (Bird et al. 2009). We also observe that the length of the discussion plus the length of the issue report (description), measured by wordiness, is an influential characteristic to predict co-changes.

Finally, it is important to highlight that it is possible to use only data from the commit context to build the classifiers. The metrics related to the number of lines of code added and removed and the code churn were frequently selected by the classifier as important contextual information in 6 out of 10 projects. We found that it is possible to achieve an F-measure of 56% (on average) by using only this information versus 32% by using association rules models.

This result is particularly interesting because even using only information from commits—the same source necessary to perform association rules analysis—the co-change prediction can be improved.

On average the numeric contextual information models used 6 out of 17 metrics. The metrics related to Commit (#15, #16, and #17), Communication (#3 and #4), and Developer Role (#6) obtained the highest importance score over all releases analyzed.

4 Discussion

We discuss our results from the perspective of the implications for practice and research and how developers see the results.

Table 5 Average of variable importance score for each kind of contextual information

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Camel	24.7	21.5	41.8	59.5	3.9	32.0	22.1	27.0	23.3	22.2	24.5	27.2	22.5	15.1	50.7	49.9	63.1
Cassandra	23.4	22.5	48.2	56.5	13.8	51.4	45.7	43.0	46.6	45.5	23.9	35.4	33.2	8.8	51.0	51.6	65.8
Cloudstack	19.1	19.5	54.3	52.2	22.1	48.6	41.2	35.8	46.4	40.7	19.6	27.0	39.0	15.5	37.8	32.4	57.2
CXF	22.5	21.4	32.8	58.0	1.5	21.8	15.7	17.7	13.6	16.4	22.9	17.7	10.6	13.2	53.3	52.1	62.2
Derby	25.5	17.8	37.1	40.8	13.6	38.7	34.8	32.8	30.8	33.1	25.6	32.6	26.3	13.0	53.9	53.4	64.1
Hadoop	27.8	23.8	45.5	50.5	25.0	50.5	45.4	44.5	45.9	45.9	28.7	40.7	41.4	11.7	39.9	39.0	53.5
Hbase	26.5	24.5	43.7	49.5	26.7	47.8	41.7	43.9	43.0	44.2	26.6	39.9	37.6	12.5	40.2	39.5	53.6
Hive	27.9	20.5	49.3	55.4	21.0	49.3	44.2	46.3	44.5	46.0	28.1	39.1	35.5	12.0	38.6	38.9	52.9
Lucene	23.5	13.0	34.3	38.9	17.6	37.3	32.0	32.0	31.0	33.3	25.2	29.4	26.4	14.4	56.4	54.6	65.7
Solr	21.3	16.4	48.3	52.9	15.7	45.2	34.4	37.7	41.5	37.2	23.7	32.5	27.8	14.2	51.7	55.2	76.0
Total	24.2	20.1	43.5	51.4	16.1	42.3	35.7	36.1	36.7	36.4	24.9	32.1	30.0	13.1	47.3	46.7	61.4

1—# of Issue discussants, 2—# of Issue Developer Commenters, 3—# of Issue Comments, 4—Issue Wordiness, 5—Betweenness centrality, 6—Closeness centrality, 7—Efficiency, 8—Effective Size, 9—Constraint, 10—Hierarchy, 11—Size, 12—Ties, 13—Density, 14—Diameter, 15—# of lines of code added, 16—# of lines of code deleted, 17—Code Churn

4.1 Implications for practice and research

We investigated the tradeoff between the coverage and accuracy when we predict if an expected co-change occurred in a specific commit. We found that association rules models can retrieve more co-changes, but these models suffer from false recommendation that decreases their prediction accuracy.

To improve the accuracy of association models, the rules need to be more relevant with high values of support and confidence, but the coverage becomes smaller. In a practical scenario, choosing the best thresholds is not easy (Zimmermann et al. 2005). Hence, models based on contextual information are simpler because they do not require prior configuration. We used a random forest algorithm with default configuration. However, it is important to mention that the effort to collect data is higher for contextual information models because association rules only depend on commit data.

Models based on contextual information have a smaller coverage, but are more accurate, improving the precision in relation to association rule models. The use of machine learning techniques combined with contextual information reduced the number of false recommendations, capturing the context when an expected co-change occurs in a specific commit. The contextual information models achieved high accuracy in projects with a large number of commits (such as CXF, Derby, and Hbase). On the other hand, in projects with fewer commits (such as Cloudstack and Solr), association rule models showed better results.

The use of contextual information is promising, however, it is an open question how much context is necessary to improve coverage and accuracy. We did not test the entropy of changes in these projects, but the entropy of changes can show the possible effects of disorder caused by continuous changes (Hassan 2009; Canfora et al. 2014).

Another important question that arises from our study is related to the period when the training and test set were created. A release period often focuses on a specific goal, for example, correcting critical bugs, refactoring some part of the project, or implementing new features. It is not clear what the effect was of choosing this timeframe. We conjecture that a release can capture “related context” to build training sets.

Using textual and Boolean metrics in contextual information models, we could improve the accuracy of co-change predictions. However, the coverage was penalized. In open source projects, these sources of contextual information may not be feasible, since developers may contribute only occasionally, making sporadic commits or reporting issues. However, these metrics can be good indicators in enterprise environments, when all contributors are previously known.

4.2 Community feedback

To adopt the approach in practice, developers’ perspectives are crucial. To explore how developers of the analyzed project see the results, we sent messages to the developers’ mailing list of each analyzed project. We asked the developers to share their impressions about our results, highlighting that we did not use any information related to structural dependencies to predict which files would co-change. We also asked their impression of the influence of contextual information and how the approach would help them in practice.

We created a website[5] for each community, presenting a practical example such as the one described in Section 2, the details of the method used to predict co-changes, and a spreadsheet

[5] Example for Derby project: http://igor.pro.br/cochanges/derby.html

with the results. We received more than 500 pageviews and received feedback and interacted with 20 distinct developers.

According to the developers, the proposed approach can be used in practice, especially to support newcomer developers to navigate through the code and complete the changes. This can be observed in a message from a CXF developer: "*... I agree with Chris [fictitious name] that may be useful to help newcomers to navigate in the project [to make their contributions] ...*" However, some developers disagreed that the approach would be useful during code review.

Regarding the importance of contextual information to predict co-changes, developers were surprised by the results, since it is a common practice to find artifacts that are structurally connected or contained in the same package. A developer from Derby said: *"... yes, I agree that is the normal way. Also through code review, running tests, and messages from the compiler. Is your idea that, given a database of change history as you have described it, some tool would be able to notice when the developer makes a certain type of change, and then suggest other related changes that are typically made at the same time ...? [Yes] ... I think that's a pretty interesting idea.*" CloudStack developers suggested that the approach could be used to guide developers during the test phase. CloudStack uses Test Driven Development, and sometimes it is very hard to find related classes that need to be instantiated to write test cases.

Developers also discussed the real example described in the message. For example, a developer from Lucene/Solr inspected the results and identified co-changes in which the prediction models captured commits involving changes in both projects at the same time. In such cases, the developer suggested that it is interesting to make a prediction "between projects," especially when commits to a project cause or require changes in an associated sub-project.

In addition to the suggestions, developers discussed and pointed out a limitation of the proposed approach. There are cases in which a new file could have been created and the absence of historical information would not allow execution of the prediction models. A CloudStack developer suggested using the history of files in the same package to build prediction models for new files. According to the developer, files of the same package could have "a similar context," and this context could be used to deal with this "cold start" problem.

5 Related work

Researchers have relied on coupling concepts to recommend co-changes. For example, Zimmermann and colleagues (Zimmermann et al. 2005) built an Eclipse plug-in that collects information about source code changes from repositories and warns developers about probable co-changes. They used association rules to suggest change coupling among files at method and file levels. The authors reported precision values around 30% and recommended that the analysis should be made at the file level instead of method level. More recently, the TARMAQ algorithm was proposed (Rolfsnes et al. 2016) to improve this approach.

Zhou et al. (Zhou et al. 2008) proposed a model to predict co-changes using Bayesian networks. They extracted features like static source code dependencies, past co-change frequency, age of change, author information, change request, and change candidate. They conducted experiments on two open source projects (Azureus and ArgoUML) and reported precision values around 60% and recall values of 40%.

Recently, Sun et al. (Sun et al. 2015) compared three tools based on coupling concepts. They compared ROSE (association rules) (Zimmermann et al. 2005), IRC2M (conceptual coupling), and Columbus (structural coupling). The combination of any two approaches improved the accuracy in general terms, however, the combination of three approaches did not improve the results. The authors reported recall values ranging from 55 to 70% and precision values between 30 and 45%.

Our paper differs from previous work since we consider previously unused contextual information. We also tested the results in 10 projects. In general, 4 projects were used by previous research. Using information related to developers' communication and issue context is new and presented promising results to reduce the number of false positives and negatives. Future work can investigate how to develop hybrid approaches to obtain even better results.

6 Threats to validity

In the following, we discuss the threats to the validity of our study.

A threat related to co-change identification is tangled commits (Herzig and Zeller 2013), since developers often commit unrelated or loosely related code changes in a single commit. In our study, this threat is limited, as we are linking commits per issue. In addition, we performed a careful selection of issues, using issues that were closed, fixed, and for which the source code was integrated.

The set of metrics used might not be complete. We dealt with this threat by performing a selection of measurements along different dimensions of software development. We chose metrics from contextual information related to issue, communication, and commit metadata. To select the set of metrics, we were inspired by previous work on prediction models and a mapping study (Wiese et al. 2014a).

Another concern is related to overfitting. In prediction models, overfitting occurs when a prediction model has random error or noise instead of an underlying relationship. Our models were planned to be specific to each pair of files. To address the overfitting of our classifiers, we used the random forest algorithm. According to the literature (Breiman 2001; McIntosh et al. 2014), during the model training phase, the algorithm selects contextual information with less correlation. Furthermore, training and test sets were always from different releases, i.e., release N to build the training set and release N+1 to test. In any case, the usage of a high number of metrics may influence the results for each classifier, making them more accurate. However, it is important to highlight that on average, the numeric contextual information models used 6 out of 17 metrics, and it is possible to achieve F-measure values of 56% (on average) by using only three metrics from commit information versus 32% by using association rules models (RQ2).

External validity In our analysis, we collected data from 10 Apache projects. Each project was selected to reflect different domains and communities, which may reflect the way that the software changes. However, our results might not generalize to other communities and projects. Replication of this work in a large set of systems is required in order to achieve more general conclusions.

7 Conclusions

Co-change prediction approaches aim to assist developers in identifying artifacts that are likely to change together. Previous approaches rely on different types of software coupling to make recommendations of co-changes. In this paper, we investigated novel sources of information. Based on the idea that artifacts change for different reasons, we gathered contextual sociotechnical information from tasks, communication, and commit metadata from software changes to build prediction models to identify when a co-change occurs in a specific commit.

We set out to investigate the coverage and accuracy of contextual information by studying 10 Apache projects. The main contributions of this work are:

- An approach to build customized models for each pair of files based on contextual information: according to the effect size measure, contextual information models have more accuracy to predict co-changes than association rules, regardless of the support and confidence thresholds (RQ1).
- To identify the most influential subset of contextual information: we could identify the best subset of metrics related to contextual information. We found that social aspects are relevant to predict co-changes, especially the length of discussion collected from issue reports and how "close" developers are in the social communication network. The number of lines of code added and removed and code churn were also important metrics used by the co-change prediction models (RQ2).

In conclusion, we found that contextual information has two advantages when compared to association rule models: (i) it reduces the number of false recommendations, and (ii) it determines, independent of thresholds, if a dependency is strong enough to be used as a co-change indicator. We are currently building a tool to test our approach in projects from outside the Apache Software Foundation, as well as compare it to other types of couplings used to recommend co-changes.

References

Ball, T., Kim, J., & Siy, H. P. (1997). If your version control system could talk. *ICSE Work Process Model Empir Stud Softw Eng.*.

Bavota, G., Dit, B., Oliveto, R., et al. (2013). An empirical study on the developers' perception of software coupling. *Proc - Int Conf Softw Eng*, 692–701. https://doi.org/10.1109/ICSE.2013.6606615.

Beyer D, Noack A (2005) Clustering software artifacts based on frequent common changes. In: 13th International Workshop on Program Comprehension (IWPC'05). pp 259–268.

Bird, C., Nagappan, N., Murphy, B., Gall, H., Devanbu, P., 2009. Putting it all together: using socio-technical networks to predict failures. In: Proceedings - International Symposium on Software Reliability Engineering, ISSRE. pp. 109–119.

Bohner, S. A., & Arnold, R. S. (1996). *Software change impact analysis*. IEEE Computer Society Press.

Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5–32. https://doi.org/10.1023/A:1010933404324.

Briand LC, Wust J, Lounis H (1999) Using coupling measurement for impact analysis in object-orientedsystems. Proc IEEE Int Conf Softw Maint - 1999 (ICSM'99) Software Maint Bus Chang (Cat No99CB36360). https://doi.org/10.1109/ICSM.1999.792645.

Canfora, G., Cerulo, L., Cimitile, M., & Di Penta, M. (2014). How changes affect software entropy: an empirical study. *Empirical Software Engineering, 19*(1), 1–38. https://doi.org/10.1007/s10664-012-9214-z.

Conway, M. E. (1968). How do committees invent. *Datamation, 14*, 28–31.

Powers, D. M. (2011). Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies, 2*, 37–63.

Dias M, Bacchelli A, Gousios G, et al (2015) Untangling fine-grained code changes. In: 2015 IEEE 22nd international conference on software analysis, evolution, and reengineering, SANER 2015 - proceedings. pp 341–350.

Dit B., Wagner M., Wen S., et al (2014) ImpactMiner: a tool for change impact analysis. In: 36th international conference on software engineering, ICSE companion 2014 - proceedings. pp 540–543.

Gall, H., Hajek, K., & Jazayeri, M. (1998). Detection of logical coupling based on product release history. In *Proceedings Int Conf Softw Maint (cat no 98CB36272)*. https://doi.org/10.1109/ICSM.1998.738508.

Gethers M, Dit B, Kagdi H, Poshyvanyk D (2012) Integrated impact analysis for managing software changes. In: Proceedings - International Conference on Software Engineering pp 430–440.

Gethers, M., & Poshyvanyk, D. (2010). *Using relational topic models to capture coupling among classes in object-oriented software systems*. IEEE International Conference on Software Maintenance, ICSM.

Hassan, A. E. (2009). Predicting faults using the complexity of code changes. *Proceedings - International Conference on Software Engineering.*, 78–88.

Hassan, A. E., & Holt, R. C. (2004). Predicting change propagation in software systems. *IEEE International Conference on Software Maintenance, ICSM.*, 284–293.

Herzig, K., & Zeller, A. (2013). The impact of tangled code changes. *IEEE International Working Conference on Mining Software Repositories.*, 121–130.

Kagdi, H., Gethers, M., & Poshyvanyk, D. (2013). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering, 18*(5), 933–969. https://doi.org/10.1007/s10664-012-9233-9.

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software, 28*, 1–26.

Lessmann, S., Baesens, B., Mues, C., & Pietsch, S. (2008). Benchmarking classification models for software defect prediction: a proposed framework and novel findings. *IEEE Transactions on Software Engineering, 34*(4), 485–496. https://doi.org/10.1109/TSE.2008.35.

Macho, C., McIntosh, S., & Pinzger, M. (2016). Predicting build co-changes with source code change and commit categories. *Proc. of the International Conference on Software Analysis, Evolution, and Reengineering (SANER).*, 541–551.

McIntosh, S., Adams, B., Nagappan, M., & Hassan, A. E. (2014). Mining co-change information to understand when build changes are necessary. *Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME).*, 241–250.

Moonen L, Di Alesio S, Binkley D, Rolfsnes T (2016) Practical guidelines for change recommendation using association rule mining. In: International Conference on Automated Software Engineering (ASE). p 11.

Oliva GA, Gerosa MA (2015a) Experience report: how do structural dependencies influence change propagation? An empirical study. In: Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering.

Oliva, G. A., & Gerosa, M. A. (2015b). Change coupling between software artifacts: learning from past changes. In C. Bird, T. Menzies, & T. Zimmermann (Eds.), *The art and science of analyzing software data* (pp. 285–324). Morgan Kaufmann.

Oliva, G. A., Steinmacher, I., Wiese, I., & Gerosa, M. A. (2013). What can commit metadata tell us about design degradation? In *Proceedings of the 2013 international workshop on principles of software evolution - IWPSE 2013* (p. 18). ACM Press.

Orso, A., Apiwattanapong, T., Law, J., et al. (2004). An empirical comparison of dynamic impact analysis algorithms. *Proceedings 26th Int Conf Softw Eng.* https://doi.org/10.1109/ICSE.2004.1317471.

Revelle, M., Gethers, M., & Poshyvanyk, D. (2011). Using structural and textual information to capture feature coupling in object-oriented software. *Empirical Software Engineering, 16*(6), 773–811. https://doi.org/10.1007/s10664-011-9159-7.

Rolfsnes, T., Di, A. S., Behjati, R., et al. (2016). Generalizing the analysis of evolutionary coupling for software change impact analysis. *23rd IEEE Int Conf Softw Anal Evol Reengineering, 12*. https://doi.org/10.1109/SANER.2016.101.

Steinmacher I, Treude C, Conte T, Gerosa MA (2016) Overcoming open source project entry barriers with a portal for newcomers". In: 38th International Conference on Software Engineering. pp 1–12.

Sun, X., Li, B., Leung, H., Li, B., & Zhu, J. (2015). Static change impact analysis techniques: a comparative study. *Journal of Systems and Software, 109*, 137–149. https://doi.org/10.1016/j.jss.2015.07.047.

Wasserman, S., & Faust, K. (1994). *Social network analysis: methods and applications (structural analysis in the social sciences)*. Cambridge: Cambridge University Press. https://doi.org/10.1017/CBO9780511815478.

Wiese IS, Côgo FR, Ré R, et al (2014a) Social metrics included in prediction models on software engineering: a mapping study. In: Wagner S, Penta M Di (eds) The 10th International Conference on Predictive Models in Software Engineering, {PROMISE} '14, Torino, Italy, September 17, 2014. ACM, pp 72–81.

Wiese, I. S., Kuroda, R. T., Junior, D. N. R., et al. (2014b). Using structural holes metrics from communication networks to predict change dependencies. In N. Baloian, F. Burstein, H. Ogata, et al. (Eds.), *Collaboration and Technology - 20th International Conference, {CRIWG} 2014, Santiago, Chile, September 7–10, 2014. Proceedings. Springer* (pp. 294–310).

Wiese, I. S., Ré, R., Steinmacher, I., Kuroda, R. T., Oliva, G. A., Treude, C., & Gerosa, M. A. (2016). Using contextual information to predict co-changes. *Journal of Systems and Software, 128*, 220–235. https://doi.org/10.1016/j.jss.2016.07.016.

Wiese IS, Ré R, Steinmacher I, et al (2015) Predicting change propagation from repository information. In: Proceedings - 29th Brazilian symposium on software engineering, SBES 2015. pp 100–109.

Ying, A. T. T., Murphy, G. C., Ng, R., & Chu-Carroll, M. C. (2004). Predicting source code changes by mining change history. *IEEE Transactions on Software Engineering, 30*(9), 574–586. https://doi.org/10.1109/TSE.2004.52.

Zhou, Y., Wursch, M., Giger, E., et al. (2008). A Bayesian network based approach for change coupling prediction. *Fifteenth Work Conf Reverse Eng Proc*, 27–36\r348.

Zimmermann, T., Weißgerber, P., Diehl, S., & Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering, 31*(6), 429–445. https://doi.org/10.1109/TSE.2005.72.

Igor Wiese is an associate professor in the Department of Computing at the Federal University of Technology—Paraná, where he researches Mining Software Repositories techniques, Human Aspects of Software Engineering, and related topics. Wiese received a PhD in computer science from the University of São Paulo. Contact him at igor@utfpr.edu.br

Rodrigo Takashi Kuroda received the M.Sc. degree in Informatics from the Federal University of Technology—Paraná (UTFPR) in 2017. Therein, he researched Machine Learning applied to Software Engineering to support developer to maintain the software. He graduated with a degree in Technology in System for Internet at UTFPR in 2011. Since 2013, he is a software developer at Atos Brazil located in Londrina (PR) city. His specialty includes Java language and related technologies.

Igor Steinmacher is an associate professor in the Department of Computing at the Federal University of Technology—Paraná and a postdoctoral scholar in the School of Informatics, Computing, and Cyber Systems at the Northern Arizona University, where he researches human aspects of software engineering and related topics. Steinmacher received a PhD in computer science from the University of São Paulo. Contact him at igorfs@utfpr.edu.br.

Gustavo Ansaldi Oliva is a postdoctoral fellow at Queen's University in Canada under the supervision of Professor Dr. Ahmed Hassan. His research focuses on understanding the rationale of software changes and their impact on Software Maintenance and Evolution. As such, his studies typically involve Mining Software Repositories (MSR) and applying static code analysis, evolutionary code analysis, and statistical learning techniques. Gustavo received his MSc and PhD from the University of São Paulo (USP) in Brazil under the supervision of professor Dr. Marco Aurélio Gerosa.

Reginaldo Ré received from the State University of São Paulo the MSc degree in 2002 and the Ph.D. degree in 2009, both degrees in computer science and computational mathematics. His research interests in the last years are data mining, defect prediction, co-changes prediction, and cloud computing. Presently, he is a software engineering professor at the Universidade Tecnológica Federal do Paraná—Campo Mourão, Brazil.

Christoph Treude is an ARC DECRA fellow and a senior lecturer in the School of Computer Science at the University of Adelaide, Australia. He completed his PhD in Computer Science at the University of Victoria, Canada, in 2012, and received his Diplom degree from the University of Siegen, Germany, in 2007. The goal of his research is to advance collaborative software engineering through empirical studies and the innovation of processes and tools that explicitly take the wide variety of artifacts available in a software repository into account.

Marco Aurélio Gerosa is an associate professor at the Northern Arizona University (NAU) and a Ph.D. advisor at the University of São Paulo (USP). He researches Software Engineering and CSCW. Recent projects include the development of tools and strategies to support newcomers onboarding to open source software communities and the design of chatbots for tourism. He published more than 200 papers served to the program committee of important conferences, such as FSE, CSCW, SANER, and MSR.

Affiliations

Igor Scaliante Wiese[1] • **Rodrigo Takashi Kuroda**[2] • **Igor Steinmacher**[3] • **Gustavo Ansaldi Oliva**[4] • **Reginaldo Ré**[1] • **Christoph Treude**[5] • **Marco Aurelio Gerosa**[3]

Rodrigo Takashi Kuroda
rodrigokuroda@gmail.com

Igor Steinmacher
Igor.Steinmacher@nau.edu

Gustavo Ansaldi Oliva
gustavo@cs.queensu.ca

Reginaldo Ré
reginaldo@utfpr.edu.br

Christoph Treude
christoph.treude@adelaide.edu.au

Marco Aurelio Gerosa
Marco.Gerosa@nau.edu

[1] Departamento de Ciência da Computação, Universidade Tecnológica Federal do Paraná, Paraná, Brazil

[2] PPGI/UTFPR–CP, Universidade Tecnológica Federal do Paraná, Paraná, Brazil

[3] School of Informatics, Computing, and Cyber Systems, Northern Arizona University, Flagstaff, AZ, USA

[4] School of Computing, Queen's University, Kingston, Canada

[5] School of Computer Science, University of Adelaide, Adelaide, Australia