An Empirical Study of the Relation Between Strong Change Coupling and Defects Using History and Social Metrics in the Apache Aries Project

Igor Scaliante Wiese[1(✉)], Rodrigo Takashi Kuroda[2], Reginaldo Re[1], Gustavo Ansaldi Oliva[3], and Marco Aurélio Gerosa[3]

[1] Departament of Computing, UTFPR – Universidade Tecnológica Federal Do Paraná/Campus Campo Mourão, Campo Mourão, Brazil
{igor,reginaldo}@utfpr.edu.br
[2] PPGI - UTFPR/Campus Cornélio Procópio, Cornélio Procópio, Brazil
rodrigokuroda@gmail.com
[3] Departament of Computer Science, IME/USP – University of Sao Paulo, Sao Paulo, Brazil
{goliva,gerosa}@ime.usp.br

Abstract. Change coupling is an implicit relationship observed when artifacts change together during software evolution. The literature leverages change coupling analysis for several purposes. For example, researchers discovered that change coupling is associated with software defects and reveals relationships between software artifacts that cannot be found by scanning code or documentation. In this paper, we empirically investigate the strongest change couplings from the Apache Aries project to characterize and identify their impact in software development. We used historical and social metrics collected from commits and issue reports to build classification models to identify strong change couplings. Historical metrics were used because change coupling is a phenomenon associated with recurrent co-changes found in the software history. In turn, social metrics were used because developers often interact with each other in issue trackers to accomplish the tasks. Our classification models showed high accuracy, with 70-99% F-measure and 88-99% AUC. Using the same set of metrics, we also predicted the number of future defects for the artifacts involved in strong change couplings. More specifically, we were able to predict 45.7% of defects where these strong change couplings reoccurred in the post-release. These findings suggest that developers and projects managers should detect and monitor strong change couplings, because they can be associated with defects and tend to happen again in the subsequent release.

1 Introduction

Some artifacts are changed together throughout software development. The concept of *change coupling* captures this implicit connection [1]. Some benefits of change coupling analysis were discussed by D'Ambros and colleagues [2]. For example, change couplings reveal relationships not present in the code or in the documentation. Other researchers showed that change couplings affect software quality [3,4]. Previous

E. Damiani et al. (Eds.): OSS 2015, IFIP AICT 451, pp. 3–12, 2015.
DOI: 10.1007/978-3-319-17837-0_1

studies discovered which artifacts changed together in the past by mining logs from version control systems, such as SVN and Git [1,5,6]. Zimmermann et al. [6], for example, implemented a tool to predict change propagation based on change coupling. The underlying assumption in their approach is that entities that changed together in the past are likely to change together in the future.

Differently from Zimmermann's work, our focus is to characterize and identify the impact of *strong change couplings*. Literature studies have provided some evidence that these couplings are associated with defects. D'ambros, Robbles & Lanza [4] mined historical data from three open source projects and showed that change couplings correlate with defects extracted from a bug repository. Cataldo et al. [7] reported that the effect of change coupling on fault proneness was complementary and significantly more relevant than the impact of structural coupling in two software projects from different companies. In another study, Cataldo and Nambiar [8] investigated the impact of geographic distribution and technical coupling on the quality of 189 global software development projects. By technical coupling, they mean overall measures of the extent to which artifacts of the system are connected. Their results indicated that the number of change couplings among architectural components were the most significant factor explaining the number of reported defects. Other factors they took into consideration include the number of structural coupling, process maturity, and the number of geographical sites.

In this sense, the main goal of this paper is twofold. First, we empirically investigate the relation between strong change couplings and the number of defects associated with them. Afterwards, we characterize strong change couplings using historical and social metrics.

We investigated the following research questions using data from the Apache Aries project:

- **RQ 1. Are strong change couplings related to defects?** We found that strong change coupling are associated with defects. In releases with more change couplings identified, more than 50% of them are associated with at least one defect. In releases with fewer change couplings identified, we found that at least ¾ of change coupling are associated with at least one defect. These values suggest that strong change couplings can be problematic for software projects.
- **RQ 2. Can historical and social metrics identify if a change coupling is strong?** We built models that identify strong change couplings with high accuracy (70-99% F-measure and 88-99% AUC). In addition, we applied the feature selection analysis to reduce the effort in building the prediction models and gain insights about which metrics are more important. We found that the length of a task discussion in the issue tracker, number of distinct committers, experience of committers, number of defect tasks associated with a change coupling, and age of change coupling were the best predictors.
- **RQ 3. Can we predict defects associated with strong change couplings?** We built a defect prediction model to help developers and managers to predict which strong change dependencies will have associated defects in the post-release. We correctly predicted 45.7% of the defects.

Results of our empirical study suggest that software engineers should detect and monitor strong change couplings, since they are associated with defects. Moreover, strong change couplings tend to happen again in the post-release to fix new defects.

This paper is structured as follows. In Section 2, we describe the methodology. In Section 3, we answer RQ1 and RQ2. Section 4 discusses the results of RQ3. Finally, conclusions and plans for future work are presented in Section 5.

2 Methodology

This section describes the methodology we followed to collect data and identify change couplings.

2.1 Identifying Strong Change Couplings

To identify the strong change couplings, we mined the change history of each release of the Apache Aries project. We considered just the changes submitted as a patch to an issue. If an issue had more than one associated commit, we grouped all commits in one single change transaction and, for each transaction, we employed a data mining technique called frequent itemset mining [1]. This technique was used in previous research [1,5,6] to uncover frequently occurring patterns (co-changed classes or methods) in a given set of transactions (change-sets/commits).

The frequency is typically measured by the metric of *support value*, which simply gives the number of transactions in which an itemset appears. In our study, the support value of an itemset consisting of files A and B corresponds to the number of issues in which they appeared together. The strength of a change coupling from A to B is determined by the ratio of co-changes (*support value*) and the number of times the artifact B changed. The artifact B in this example was the file that changed in more issues compared to artifact A. Based on these metrics, we used a quartile analysis to determine the “relevant” change couplings: all couplings with support higher than the third quartile were labeled as “strong”. All other couplings were labeled as “weak.”

2.2 Data Collection

In this paper, we collected data from the Apache Aries project, which delivers a set of pluggable Java OSGi components. We started the data collection extracting all issues from the Jira issue tracking system. For each issue, we collected its metadata and the associated source code changes from the version control repository. Since these two pieces of information were stored in different environments, we searched by words “defect, bug, fix” and an issue ID normally annotated by developers as “#”+issue number (e.g. #10). Using this query, we parse the commit messages and link each issue to their respective set of commits.

Table 1 summarizes the data collected from the Apache Aries project. It presents the release number, the number of issues, the number of change couplings, and the ratio of change couplings per issue for each release.

Table 1. Data collection summarization

Release	# of Issues	# of change couplings	Change couplings per issue	Duration of release (mm/dd/yy) - # months
0.1	194	4919	25.35	09/01/09 - 05/13/10 (8 months)
0.2	61	136	2.22	05/13/10 - 09/06/10 (4 months)
0.3	129	2465	19.10	09/06/10 - 02/21/11 (5 months)
0.4	85	469	5.51	02/21/11 - 11/08/11 (9 months)
1.0	62	300	4.83	11/08/11 - 10/12/12 (11 months)
1.1	25	48	1.92	01/29/13 - 01/23/14 (12 months)

We notice from the data above that Aries' releases have differences in the ratio of change couplings per issue report and in their duration. For example, release 0.2 lasted 4 months and involved fixing 61 issues and changing few files together. On the other hand, even though release 0.3 was one month longer, the number of fixed issues were higher (129) and many more files were changed together (2465) to fix these issues.

2.3 Classification Approach

We run the *random forest* technique to construct classifiers to identify strong change couplings. The random forest technique builds a large number of decision trees at training time using a random subset of all of the attributes. In our study, these attributes correspond to the historical and social metrics [9]. The technique performs a random split to ensure that all of the trees have a low correlation between them [9]. The random forest technique was already used in previous research [10].

Using an aggregation of votes from all trees, the random forest technique decides whether the final score is higher than a chosen threshold to determine if a specific change coupling will be deemed as strong or weak. To obtain the testing set and evaluate the performance of our classifiers, we used 10-fold cross-validation. Cross-validation splits the data into ten equal parts using nine parts for the training set and one part for the testing set.

We used two well-known metrics to evaluate our classifiers: F-measure and the Area Under the Curve (AUC). F-measure computes the harmonic mean of precision and recall for each class. AUC plots true positive rates against the false positive rates for various values of the chosen threshold used to determine whether a change coupling is classified as strong. The values of both metrics range from 0 to 1. Values close to 1 are desirable and indicate the best classifiers. We also analyzed the number of change couplings correctly predicted to further evaluate our classifiers.

3 Characterizing Strong and Weak Change Couplings

In this study, we conjecture that the set of strong change couplings are more relevant and consequently demands more attention from software developers In Section 3.1

(RQ1), we characterize strong change couplings by investigating if they are associated with software defects. In Section 3.2 (RQ2), we investigate whether historical and social metrics aids in the identification of strong couplings.

3.1 RQ1: Are Strong Change Couplings Related to Defects?

To answer this research question, we first counted the number of defect issues associated with strong change couplings in each release.

Table 2 depicts the number of instances labeled as strong change coupling with defects, the total of strong change couplings found, and the ratio of change couplings with defects. We found that the majority of the strong change couplings could be associated with at least one defect.

Table 2. Strong Change couplings (sCC) summary

Release	sCC with defects	Total of sCC	sCC with defect / total sCC (%)	By number of defects						
				1	2	3	4	5	6	7
0.1	719	1269	57.00%	531	151	20	15	2	0	0
0.2	9	9	100.00%	5	3	1	0	0	0	0
0.3	497	529	94.00%	128	276	67	19	3	5	1
0.4	63	82	77.00%	21	39	3	0	0	0	0
1.0	10	10	100.00%	0	6	4	0	0	0	0
1.1	3	4	75%%	1	2	0	0	0	0	0

To check just how correlated strong change couplings and defects are, Spearman's rank correlation coefficient (*rho*) was used. The Spearman correlation is a nonparametric measure of statistical dependence between two variables. The value returned by the Spearman correlation can range between +1 (positive correlation) to -1 (negative correlation). We calculated the correlation between the number of defects and the number of co-changes for each strong change coupling (considering all the releases in a whole).We found that strong change couplings are moderately correlated (*rho* 0.46, $p < 0.001$) with the number of defects. We noticed that releases 0.1, 0.3, and 0.4 have the majority of the strong change couplings that are correlated with defects. It is important to highlight that releases 0.1, 0.3 and 0.4 had more files changed and issues fixed compared to the releases 0.2, 1.0 and 1.1 (as shown in Table 1).

Considering the minor releases, we observed that the relation between strong change couplings and defects were higher, showing that at least 75% of the strong change couplings have at least one defect associated. Previous research also shows that, in general, change coupling is correlated with defects, both in open source [4] and industrial projects [3]. A possible reason for that is related to design issues that change couplings can be associated with. For example, some authors have associated these change couplings with the information hiding principle [11,12] described by Parnas [13]. The principle of information hiding indicates that two elements depending on the same internal class should be placed into the same module to hide the design decisions.

We found that strong change couplings are correlated with defects in the Aries project. We found positive correlation even when couplings happened in a small amount (fewer pairs of file co-changing).

3.2 RQ 2: Can Historical and Social Metrics Identify if a Change Coupling is Strong?

Previous research investigated the interplay between structural dependencies and change coupling [14,15]. They concluded that structural dependencies often could not explain or justify the emergence of change couplings. Thus, we do not yet have a clear idea of the nature of change couplings [7]. In this paper, instead of structural dependencies, we relied on historical and social metrics to build models and classify change couplings as “strong” or “weak.” As we mentioned in Section 2.2, we distinguish between strong and weak change couplings based on a quartile analysis of their support value. Table 3 presents the number of change couplings per class and the values of F-measure and AUC.

Table 3. Prediction results to strong and weak change couplings using cross-validation 10-fold

Release	#strong	#weak	Total of Instances	F-measure Strong	F-measure Weak	AUC
0.1	1269	3650	4919	0.98	0.99	0.98
0.2	9	127	136	0.75	0.98	0.88
0.3	529	1936	2465	0.98	0.98	0.98
0.4	82	387	469	0.90	0.97	0.94
1.0	10	290	300	0.70	0.99	0.99

**The release 1.1 was used only as a test set

Table 4 presents the results when we train the models using data from a previous release to identify strong change couplings in the current release. The results show that for two releases we correctly predict more than 78% of all strong change couplings. However, the class imbalance problem in these cases affected the results for three releases (0.3, 1.0, and 1.1), since we got a few number of strong change couplings instances in the training set to perform the prediction in the following release.

To reduce the class imbalance problem, we grouped all previous releases to train the models and the next release to test. Table 5 presents the results of this new prediction model. Three (0.3, 1.0, and 1.1) out of four releases had better results. For example, in release 0.3 we noticed an improvement of 35.54% (20.79% to 56.33%). It is important to mention that in a practical scenario, a project may not have sufficient history to group the data. Furthermore, when classification models are used, the smallest the training set size, the lower the effort to build it.

Table 4. Prediction results to strong change couplings using a previous release to train and next release to test

Release Test	% of Correct predictions for strong CC	# of strong CC tested
0.2	100.00%	9 (9)
0.3	20.79%	110 (529)
0.4	78.04%	64 (82)
1.0	30.00%	3 (10)
1.1	0.00%	0 (4)

Table 5. Prediction results to strong change couplings using all previous releases to train and next release to test

Release Train	Release Test	% of Corrected strong CC predicted	# of strong CC tested
0.1 + 0.2	0.3	56.33%	298 (529)
0.1 + 0.2 + 0.3	0.4	76.82%	63 (82)
0.1 + 0.2 + 0.3 + 0.4	1.0	60.00%	6 (10)
0.1 + 0.2 + 0.3 + 0.4 + 1.0	1.1	100.00%	4 (4)

Our prediction model based on historical and social metrics accurately identified strong change couplings, with F-measure ranging from 70% to 98% and AUC ranging from 88 to 99%. As expected, grouping data from previous releases to predict subsequent change couplings improved the results. The percentage of correctly predicted strong change couplings range from 56% to 100%.

3.2.1 Which are the Best Metrics to Identify Strong Change Couplings?

To identify the best set of historical and social metrics to predict change couplings, we used the Correlation Feature Selection (CFS) algorithm. CFS evaluate subsets of features based on identified good feature subsets that contain features highly correlated with the classification, yet uncorrelated with each other. By using CFS, we wanted to reduce the training time, enhance generalization by reducing overfitting, and improving the model interpretability by finding the best metrics that predict strong change couplings. It is also important to find the best set of metrics to reduce the effort to collect the metrics and apply the models in practice.

Table 6 presents the number of selections made by the feature selection technique for each metric selected in at least one release. All metrics were sorted by their relevance. Wordiness (number 1 in the table) was the most important metric to identify strong change couplings. This metric was selected in four out of the five releases that were used to train the models.

In average, we concluded that 5 metrics by release were sufficient to identify strong change couplings. We deemed as relevant the metrics 1 to 6, since they were chosen in at least two different releases.

Table 6. Strong change couplings (CC) summary

	Software Metrics															
Release	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0.1	X															
0.2		X	X	X		X	X	X								X
0.3	X	X			X											
0.4	X	X	X	X	X	X					X	X	X	X	X	
1.0	X								X	X						

**blue columns are social metrics | white columns are historical metrics
1-wordiness, 2-devCommitSUM, 3-oexp, 4-oexp2, 5-taskDefect, 6-ageTotal, 7-devCommenters, 8-add, 9-clsMAX, 10-dgrSUM, 11-commits, 12-taskImprovement, 13-efvSizeMdn, 14-efvSizeMax, 15-MinorContributors, 16-devCommitAvg

*experience of committers is given by: dividing the total number of lines changed in the file in each release by the number of lines changed by each developer that changed the file in the same release. We got the maximum value of experience for each file involved in a change coupling.

**ageTotal is measured for each release and corresponds to the number of weeks in the period delimited by the first and the last commits in which the two files co-changed.

For the Aries Project, the best subset of metrics were composed by: length of discussion in terms of the number of words used (wordiness), number of distinct committers (devCommitSUM), experience of committers* (oexp, oexp2), number of defect tasks associated with a change coupling, and ageTotal**.

4 Application

4.1 RQ 3. Can we Predict Defects Associated with Strong Change Couplings?

To evaluate the applicability of our models, we wanted to check if pairs of files deemed as strongly change coupled in a release tend to co-change in the post-release. This gives us an idea of how many change couplings relationships "propagate" to the subsequent release.

Table 7. Prediction results to identify strong change couplings with defects

Release Train / Release Test	# of Strong CC with defects (next release)	% of correct predictions
0.1 / 0.2	40	60% (24)
0.2 / 0.3	7	100% (7)
0.3 / 0.4	30	20% (6)
0.4 / 1/0	4	0% (0)
1.0 / 1.1	0	-

Table 7 presents the defect prediction results for the strong change couplings identified in each release. For example, release 0.1 has 1270 strong change couplings (Table 2 – column total of strong change coupling). We found that 40 out of these

1270 couplings occur in at least one bug-fixing issue in the consecutive release. We labeled these 40 change coupling as "defective" and all the others strong change couplings as "clean." Performing the same machine learning analysis used in the previous section, we predicted whether a strong change coupling would have a defect.

We found that 81 strong change couplings happened again in the following release to fix defects and we correctly predicted 37 (45.67%) of them.

5 Conclusions

Our results show that strong change couplings are positively correlated with the number of defects. This corroborates previous results from the literature [4,7,8]. We noticed that by using the Random Forest machine-learning algorithm, it was possible to identify strong and weak change couplings for each release. In some cases, just the previous release was sufficient to train the models. In the cases where the previous releases had few strong couplings in the training set, we added all the previous history of strong and weak change couplings to improve the model accuracy. We were able to predict 45% of strong change couplings that happened again in the post-release to fix defects. These findings suggest that developers and projects managers should detect and monitor strong change couplings, since they propagate to future releases.

Potential threats to the validity can affect the results of our study. The first concern is the generalizability. In our analysis, we presented a single case study. However, based on this limited scope, our results might not generalize to other projects and domains. On the other hand, the choice of a single project allowed us to control more variables and better understand the data we collected. Another threat refers to the possible presence of tangled code changes [16] in the commits we mined.

As future work, we want to reevaluate our results on additional projects. We also want to go deeper and investigate the ways in which strong change couplings can influence code quality. This would serve as a basis for the development of new tools that would help managers monitor and track the damage caused by these couplings.

Acknowledgments. We thank Fundação Araucária, NAPSOL, NAWEB, FAPESP, and CNPQ (461101/2014-9) for the financial support. Igor receive grants from CAPES (BEX 2039-13-3). Gustavo receives individual grant from CAPES.

References

1. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Proceedings of the International Conference on Software Maintenance, p. 190. IEEE Computer Society, Washington, DC, USA (1998)
2. D'Ambros, M., Lanza, M., Lungu, M.: Visualizing co-change information with the evolution radar. IEEE Trans. Software Eng. **35**, 720–735 (2009)

3. Kirbas, S., Sen, A., Caglayan, B., Bener, A., Mahmutogullari, R.: The effect of evolutionary coupling on software defects: an industrial case study on a legacy system. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 6:1–6:7. Torino (2014)
4. D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. WCRE 2009, pp. 135–144 (2009)
5. Ying, A.T.T., Murphy, G.C., Ng, R., Chu-Carroll, M.C.: Predicting source code changes by mining change history. IEEE Trans. Softw. Eng. **30**, 574–586 (2004)
6. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: Mining version histories to guide software changes. IEEE Trans. Software Eng. **31**, 429–445 (2005)
7. Cataldo, M., Mockus, A., Roberts, J.A., Herbsleb, J.D.: Software dependencies, work dependencies, and their impact on failures. IEEE Trans. Software Eng. **35**, 864–878 (2009)
8. Cataldo, M., Nambiar, S.: The impact of geographic distribution and the nature of technical coupling on the quality of global software development projects. Journal of Software Maintenance and Evolution: Research and Practice (2010)
9. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
10. McIntosh, S., Adams, B., Nagappan, M., Hassan, A.E.: Mining co-change information to understand when build changes are necessary. In: Proc. of the 30th Int'l Conf. on Software Maintenance and Evolution (ICSME), pp. 241–250 (2014)
11. Beck, F., Diehl, S.: On the congruence of modularity and code coupling. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 354–364. ACM, Szeged (2011)
12. Silva, L.L., Valente, M.T., de A. Maia, M.: Assessing modularity using co-change clusters. In: Proceedings of the 13th International Conference on Modularity, pp. 49–60. ACM, Lugano (2014)
13. Parnas, D.L.: On the criteria to be used in decomposing systems into modules. Commun. ACM **15**, 1053–1058 (1972)
14. Geipel, M.M., Schweitzer, F.: The link between dependency and cochange: empirical evidence. IEEE Trans. Software Eng. **38**, 1432–1444 (2012)
15. Oliva, G.A., Gerosa, M.A.: On the interplay between structural and logical dependencies in open-source software. Simpósio Brasileiro de Engenharia de Software, pp. 144–153 (2011)
16. Herzig K., Zeller, A.: The impact of tangled code changes. In: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 121–130. IEEE Press, San Francisco (2013)