A Preliminary Study on the Explicitness of Bug Associations

Zengyang Li†, Jieling Xu†, Guangzong Cai†, Peng Liang‡, Ran Mo†
†School of Computer Science & Hubei Provincial Key Laboratory of Artificial Intelligence and Smart Learning, Central China Normal University, Wuhan, China
‡School of Computer Science, Wuhan University, Wuhan, China
zengyangli@ccnu.edu.cn, {xjling, guangzongcai}@mails.ccnu.edu.cn, liangp@whu.edu.cn, moran@ccnu.edu.cn

***Abstract*—Bugs are usually in associations with other bugs in a software system, e.g., a bug may result from another bug. However, such bug associations are implicit and usually cannot be traced without a significant amount of effort. Intuitively, if a bug association is easier to trace, the involved bugs can be fixed in a cleaner way. However, there is little evidence on the explicitness of bug associations. In this paper, we aim to evaluate the explicitness of bug associations, so as to get a basic understanding on such associations. To this end, we defined a metric to quantify the explicitness of a bug association, and conducted an empirical study on 11 non-trivial Apache open source software systems. The main findings are summarized as follows: (1) From the perspective of code change history, around 29% of bug pairs are not explicitly associated, and about 71% are explicitly associated to some extent; (2) Bugs in the association of Container have relatively strong association explicitness, while bugs in the association of Blocked or Blocker, Cloners, and Dependent have relatively weak association explicitness. These findings provide insights on software analyzability to practitioners and researchers.**

***Keywords*—bug association, association type, empirical study, open source software**

I. Introduction

Bugs are usually in association with other bugs in a software system [1], [2]. For instance, in project Apache *HBase*, bug *HBASE-21551* on memory leak was caused by bug *HBASE-20704* that did not handle file storage properly. Such associations are important to software maintenance in the sense that they can facilitate locating bugs and analyzing the impact of the bugs on the system [3], [4]. Intuitively, if a bug association is easier to trace, the involved bugs can be fixed in a cleaner way, i.e., the bugs can be solved more completely. Despite of the importance of bug associations, there is little evidence on the explicitness of bug associations from the perspective of change history of the software system. Hence, in this work, we aim to evaluate the explicitness of bug associations, so as to get a basic understanding on such associations.

To this end, we conducted an empirical study on 11 non-trivial Apache open source software (OSS) projects, which bugs are managed in JIRA, an issue tracking system deployed by Apache Software Foundation. In JIRA, for each project, a proportion of bug associations are manually labeled by practitioners, but it is not clear whether and to what extent such manually labeled associations can be reflected in the code change history (i.e., commits) of the project. The results of this question can reflect the difficulty of identifying the bug associations and the possibility of automatic identification.

Our main contributions lie in the following two aspects. (1) This work is an early attempt to explore the explicitness of bug associations from the perspective of software change history. (2) We defined a metric to quantify the explicitness of bug associations, and examined the bug association explicitness using this metric in 11 Apache OSS projects.

II. Background and Related Work

A. Background

There are different types of the association between two bugs. We collected 16 types of bug associations from JIRA[1], including: (1) Blocked, (2) Blocker, (3) ChildIssue, (4) Cloners, (5) Container, (6) Dependency, (7) Dependent, (8) Duplicate, (9) Incorporates, (10) ParentFeature, (11) Problem/Incident, (12) Reference, (13) Regression, (14) Related, (15) Required, and (16) Supercedes. The details of all the 16 bug association types are provided online[2].

B. Links between Issues

Many studies investigated various links (e.g., associations or dependencies) between issues (including bugs) and the characteristics and applications of these links. Kucuk *et al.* classified duplicate bugs based on analyzing the difference between duplicate and non-duplicate bugs [5]. Tomova *et al.* studied issue link type selection [6]. Researchers use trace links to support various development and maintenance tasks, such as impact analysis [3] and bug tracking [7]. However, our work is not to explore what kind of relationship between bugs, but to study the explicitness of the association between bugs that have been manually linked from the perspective of code change history.

C. Links between Bugs and Commits

Numerous studies investigated the links between bugs and commits on the detection and application of the links. Le *et al.*

This work is supported by the Natural Science Foundation of Hubei Province of China under Grant No. 2021CFB577 and the Natural Science Foundation of China (NSFC) under Grant No. 62172311.
DOI reference number: 10.18293/SEKE2022-027

[1]https://issues.apache.org/jira
[2]https://github.com/breezesway/BugAssociationType

created a discriminative model for predicting whether there is a link between commit messages and bug reports [8]. Li *et al.* made use of links between bugs and commits to identify bug-fixing commits and then calculated the change complexity of bug-fixing commits [9], [10]. However, these studies do not discuss the relationship between the corresponding commits and associated bugs, which is the focus of our study.

III. Study Design

To investigate the explicitness of bug associations, we conducted a preliminary case study on Apache OSS projects.

A. Research Questions

RQ1: How much explicitness do the bug associations have?
Rationale: The explicitness of bug associations helps to reveal how difficult the association is to be identified, thereby assessing the likelihood that the bug can be completely resolved to some extent. We quantify the explicitness of bug associations for each project, and study how the explicitness is distributed.
RQ2: Is there a difference on the association explicitness between different association types?
Rationale: We study whether there are significant differences between different association types, in order to understand whether the association explicitness of different association types is different. This gives researchers and developers inspiration for whether to pay different attention to different association types.

B. Case Selection

In this study, we only investigated Apache OSS projects which main programming language is Java. For selecting each case (i.e., OSS project) included in our study, we applied the following criteria: C1) Over 85% of the source code is written in Java; C2) The history of the project is more than 5 years; C3) The number of commits of code repository of the project is more than 4000; and C4) The number of bug pairs that are manually associated in JIRA is more than 100. C1 is set to ensure that the strength of the association between bugs is clearly defined. C2 and C3 are set to ensure that the selected project is non-trivial and has sufficient data. C4 was set to ensure that the final sample dataset for analysis was large enough for statistical analysis.

C. Data Collection

To answer the RQs formulated in Section III-A, we collected the data items listed in TABLE I, which also provides the mapping between the data items and the target RQ(s). All data items were collected from JIRA and GitHub.

TABLE I: Data items to be collected.

#	Name	Description	Target RQ(s)
D1	Associated bug pair	A pair of bugs in association.	RQ1
D2	Association type	The type of association between each pair of bugs.	RQ2
D3	Changed source files	The number of Java source files modified in the bug-fixing commit(s).	RQ1, RQ2

D. Data Analysis

To answer RQ1, we first defined a metric, namely Association Explicitness or AE, to quantify the explicitness of the association between two bugs. Assuming that the pair of bugs α and β are in an association manually labeled in JIRA, the set of Java source files modified in the commits for fixing α is F_α, and the set of source files modified in the commits for fixing β is F_β. The AE of the association between α and β is defined as follows:

$$AE = \frac{|F_\alpha \bigcap F_\beta|}{|F_\alpha \bigcup F_\beta|} \quad (1)$$

The AE of a bug pair falls into [0.0,1.0]. Second, we calculated the distribution of the proportion of bug pairs against total bug pairs over different intervals of AE value. We divide the interval into [0.0,0.0], (0.0,0.1], (0.1,0.2], ..., (0.9,1.0), [1.0,1.0]. Especially, we count the cases where two bugs are fixed in the same commit(s) in the case of [1.0,1.0].

To answer RQ2, taking all bug pairs of the selected projects as a whole, we ran the Mann-Whitney U tests to calculate whether there is a significant difference on AE between bug pairs of different association types.

IV. Study Results

A. Explicitness of Bug Associations (RQ1)

TABLE II shows the distribution of percentage of bug pairs over intervals of the AE value for the 11 projects. (1) The percentage of the bug pairs with $AE = 0.0$ of each project ranges from 22.5% to 41.1%. When taking all projects as a whole, there are around 29.4% bug pairs with $AE = 0.0$. This indicates that for those bug pairs, no source files are changed in the bug-fixing commits for both bugs of each bug pair. (2) Consider the AE interval of (0.0,0.5]. Taking all projects as a whole, the AE of about 52.6% of bug pairs falls into this interval. (3) Consider the AE interval of (0.5,1.0). Taking all projects as a whole, the AE of about 4.1% of bug pairs falls into this interval. (4) There are 4.6%-35.1% of bug pairs that have a perfect association explicitness, i.e., $AE = 1.0$, for each project. Taking all projects as a whole, 13.9% of the bug pairs are with $AE = 1.0$, which means that the same source files are changed in the bug-fixing commits of the two bugs of each of those bug pairs.

We further studied the bug pairs in which the two associated bugs are fixed in the same commit(s). The results are shown in TABLE III, where column *#BugPairA* denotes the number of bug pairs with $AE = 1.0$, column *#BugPairSC* denotes the number of bug pairs fixed in the same commit(s), and *%BugPairSC* denotes the percentage of *#BugPairSC* over *#BugPairA*. For each project, the *%BugPairSC* ranges from 0.0% from 68.6%. Especially, projects *Accumulo* and *Hadoop* do not have any bug pairs fixed in the same commit(s).

Summary: From the perspective of code change history, on average, 29.4% of bug pairs are not explicitly associated, while 70.6% of bug pairs are explicitly associated; furthermore, 52.6% of bug pairs have a relatively low association

TABLE II: Distribution of the percentage of bug pairs against the total bug pairs over intervals of AE for the selected projects.

Project	[0.0,0.0]	(0.0,0.1]	(0.1,0.2]	(0.2,0.3]	(0.3,0.4]	(0.4,0.5]	(0.5,0.6]	(0.6,0.7]	(0.7,0.8]	(0.8,0.9]	(0.9,1.0)	[1.0,1.0]
Accumulo	38.2	13.6	20.0	7.3	10.0	6.4	0.0	0.0	0.0	0.0	0.0	4.6
ActiveMQ	33.1	9.7	17.7	9.7	6.5	4.8	1.6	3.2	0.8	1.6	0.0	11.3
Calcite	22.5	23.9	15.2	12.3	6.5	5.8	0.0	2.2	0.0	0.0	0.0	11.6
Hadoop	23.2	10.6	16.7	8.9	12.3	9.2	1.4	2.1	0.3	0.3	0.0	15.0
HBase	29.1	14.9	17.9	7.6	8.9	7.6	0.7	2.7	1.0	0.0	0.3	9.3
Hive	25.5	12.2	17.8	7.4	9.0	12.4	0.7	2.1	0.9	0.5	0.0	11.6
Jackrabbit Oak	38.4	11.6	16.4	9.6	7.2	6.4	0.4	2.0	0.0	0.0	0.0	8.0
Maven	41.1	4.5	10.7	3.6	10.7	11.6	0.0	0.9	0.0	0.0	0.0	17.0
PDFBox	25.0	8.8	6.1	2.7	6.1	10.8	2.0	2.7	0.0	0.7	0.0	35.1
Solr	31.8	9.9	9.9	7.1	7.5	8.3	2.0	2.4	0.8	0.0	0.0	20.2
Wicket	30.8	3.0	15.0	7.5	9.0	9.8	0.0	7.5	0.0	0.0	0.0	17.3
Average	29.4	11.6	15.5	7.7	8.7	9.1	0.9	2.4	0.5	0.3	0.0	13.9

TABLE III: Proportion of associated bugs that are fixed in the same commit for each selected project.

Project	#BugPairA	#BugPairSC	%BugPairSC
Accumulo	5	0	0.0
ActiveMQ	14	4	28.6
Calcite	16	5	31.2
Hadoop	44	0	0.0
HBase	28	7	25.0
Hive	67	3	4.5
Jackrabbit Oak	20	5	25.0
Maven	19	5	26.3
PDFBox	52	7	13.5
Solr	51	35	68.6
Wicket	23	6	26.1

explicitness ($\leqslant 0.5$), and 13.9% of bug pairs are perfectly associated.

B. AE of Different Association Types (RQ2)

We calculated the average AE values of different association types for each selected project as shown in TABLE IV, where the last row is the average for all projects. Association types Blocked/Blocker, Cloners, Dependent, and Required have relatively small AE values on average, while Container has a relatively large AE on average. Association types ChildIssue/ParentFeature and Dependency each has only one bug pair, the average AE values for these two types do not make much sense.

We ran Mann-Whitney U tests to examine if there are significant differences on AE between bug pairs of different association types. Since ChildIssue/ParentFeature and Dependenccy have only one bug pair, and Duplicate has even no bug pair, we did not run the tests for these three association types. The test results are shown in TABLE V, where cells with *p-value*$<$0.05 are filled in gray. Specifically, a gray-filled cell with a number in bold (resp. regular) indicates that the AE of bug pairs with the association type of the corresponding row is significantly larger (resp. smaller) than the AE of bug pairs with the association type of the corresponding column. The main points are reported as follows: (1) The average AE of bug pairs with association type Blocked/Blocker is significantly smaller than the average AE of bug pairs with association types Container, Problem/Incident, Required, and Supercedes. (2) The average AE of bug pairs with association type Cloners is significantly smaller than the average AE of bug pairs with association types Container, Problem/Incident, Reference/Related, Regression, and Supercedes. (3) The average AE of bug pairs with association type Container is significantly larger than the average AE of bug pairs with association types Dependent, Incorporates, Reference/Related, Regression, and Required. (4) The average AE of bug pairs with association type Dependent is significantly smaller than the average AE of bug pairs with association types Problem/Incident, Reference/Related, Regression, and Supercedes.

Summary: Relatively speaking, bug pairs with association types Blocked/Blocker, Cloners, Dependent have relatively weak association explicitness, while bug pairs with association type Container have relatively strong association explicitness.

V. Discussion

A. Interpretation of Study Results

RQ1: (1) As we can see from TABLE II, the association explicitness of each project shows a distribution pattern: a significant proportion of bug pairs are not explicitly associated, a majority of bug pairs are explicitly associated to some extent, and a non-trivial proportion of bug pairs are perfectly associated in the bug-fixing commits. (2) On average, 70.6% of the bug pairs are with an association explicitness larger than 0, which indicates that the association of a majority of manually-associated bug pairs can be traced with some clues in the code change history. In contrast, on average 29.4% of bug pairs in the selected projects are not explicitly associated, indicating that the association of a minority of manually-associated bug pairs cannot be reflected in the code change history. (3) 13.9% of the bug pairs are fixed in the same commit(s), and a potential reason is that those bug pairs may be in specific types of associations so that the bug pairs tend to be fixed simultaneously in the same commit(s). For instance, in project *ActiveMQ*, bugs *AMQ-2967* and *AMQ-2959* are in the association of Supercedes; by definition, when *AMQ-2967* is fixed, *AMQ-2959* should also be fixed, and consequently these two bugs are fixed in the same commit.

RQ2: (1) The association explicitness of bug pairs with association type Container is significantly larger than that of bug pairs with most of other association types. One potential reason for this phenomenon is that the two bugs in an association of Container tend to be fixed in the same commit(s) according to the definition of Container. (2) The association

TABLE IV: Average AE values of different association types for each selected project.

Project	Blocked/ Blocker	ChildIssue/ ParentFeature	Cloners	Container	Dependency	Dependent	Duplicate	Incorporates	Problem/ Incident	Reference/ Related	Regression	Required	Supercedes
Accumulo	0.133		0.063			0.069		0.083	0.271	0.147	0.230		0.056
ActiveMQ						0.159		0.353	0.500	0.256	0.370	0.109	0.385
Calcite	0.110		0.292	0.667		0.142		0.206	0.410	0.245	0.341	0.071	0.643
Hadoop	0.068	1.000	0.142			0.269		0.333	0.378	0.345	0.291	0.148	0.200
Hbase	0.218		0.000			0.305		0.546	0.271	0.249	0.174	0.322	0.500
Hive	0.166		0.202	1.000	0.000	0.219		0.099	0.544	0.317	0.371	0.195	0.325
Jackrabbit Oak	0.129		0.177	0.125		0.108		0.500	0.089	0.218	0.289	0.414	0.000
Maven	1.000					0.398			0.778	0.246			1.000
PDFBox	1.000		1.000	1.000		0.193			0.500	0.470	0.750		0.500
Solr	0.161		0.333	0.639		0.280		0.273	0.202	0.362	0.339		0.235
Wicket	0.167		0.357			0.429		0.000	0.281	0.341	0.375		
All	0.180	1.000	0.235	0.668	0.000	0.224		0.287	0.345	0.302	0.318	0.233	0.384

TABLE V: *P-values* of Mann-Whitney U tests between different association types.

Association type	Blocked/ Blocker	Cloners	Container	Dependent	Incorporates	Problem/ Incident	Reference/ Related	Regression	Required	Supercedes
Blocked/Blocker	-	0.755	0.001	0.465	0.151	<0.001	1.097	2.733	0.012	0.001
Cloners	0.755	-	0.003	0.419	0.210	0.021	0.034	0.007	0.098	0.019
Container	**0.001**	**0.003**	-	**0.002**	**0.010**	0.057	**0.009**	**0.017**	**0.010**	0.093
Dependent	0.465	0.419	0.002	-	0.394	0.008	0.006	<0.001	0.067	0.011
Incorporates	0.151	0.210	0.010	0.394	-	0.155	0.502	0.154	0.434	0.119
Problem Incident	**<0.001**	**0.021**	0.057	**0.008**	0.155	-	0.181	0.509	0.159	0.722
Reference/Related	1.097	**0.034**	0.009	**0.006**	0.502	0.181	-	0.072	0.942	0.130
Regression	2.733	**0.007**	0.017	**<0.001**	0.154	0.509	0.072	-	0.359	0.323
Required	**0.012**	0.098	0.010	0.067	0.434	0.159	0.942	0.359	-	0.095
Supercedes	**0.001**	**0.019**	0.093	**0.011**	0.119	0.722	0.130	0.323	0.095	-

explicitness of bug pairs with association type Dependent is significantly smaller than that of bug pairs with most of other association types. One potential reason is that the two bugs in an association of Dependent may be coupled more in code structure than in code change history.

B. Implications

Most bug pairs are explicitly associated to certain degree, which implies that the code change history is a helpful resource for developers to deal with bugs and their impact. A significant proportion of bug pairs are not explicitly associated at all in each project, which implies that it is necessary to turn to other sources (e.g., code structure dependencies) for bug analysis.

VI. Conclusions

This work investigates to what extent manually associated bugs can be explicitly traced in their code change history, and whether there is significant difference on the association explicitness between bugs in different types of association. To answer these questions, we performed an empirical study on 11 Apache OSS projects, and obtained the following findings: (1) Around 71% of manually-associated bug pairs can be traced with overlapped changed source files in the code change history, and in contrast, around 29% are not explicitly reflected in the code change history at all. (2) Bug pairs with association types Blocked/Blocker, Cloners, and Dependent have relatively weak association explicitness, while bug pairs with association type Container has relatively strong association explicitness.

References

[1] A. Nicholson and G. Jin L.C., "Issue link label recovery and prediction for open source software," in *Proceedings of REW*. IEEE, 2021, pp. 126–135.

[2] P. Heck and A. Zaidman, "Horizontal traceability for just-in-time requirements: the case for open source feature requests," *Journal of Software: Evolution and Process*, vol. 26, no. 12, pp. 1280–1296, 2014.

[3] D. Falessi, J. Roll, J. Guo, and J. Cleland-Huang, "Leveraging historical associations between requirements and source code to identify impacted classes," *IEEE Transactions on Software Engineering*, vol. 46, no. 4, pp. 420–441, 2020.

[4] P. Rempel and P. Mäder, "Preventing defects: The impact of requirements traceability completeness on software quality," *IEEE Transactions on Software Engineering*, vol. 43, no. 8, pp. 777–797, 2017.

[5] B. Kucuk and E. Tuzun, "Characterizing duplicate bugs: An empirical analysis," in *Proceedings of SANER 2021*. IEEE, 2021, pp. 661–668.

[6] M. T. Tomova, M. Rath, and P. Mäder, "Poster: Use of trace link types in issue tracking systems," in *Proceedings of ICSE Companion*. IEEE, 2018, pp. 181–182.

[7] D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework," *Empirical Software Engineering*, vol. 22, no. 3, pp. 967–995, 2016.

[8] T.-D. B. Le, M. Linares Vásquez, D. Lo, and D. Poshyvanyk, "Rclinker: Automated linking of issue reports and commits leveraging rich contextual information," in *Proceedings of ICPC*. IEEE, 2015, pp. 36–47.

[9] Z. Li, X. Qi, Q. Yu, P. Liang, R. Mo, and C. Yang, "Multi-programming-language commits in oss: An empirical study on apache projects," in *Proceedings of ICPC*. IEEE, 2021, pp. 219–229.

[10] Z. Li, Q. Yu, P. Liang, R. Mo, and C. Yang, "Interest of defect technical debt: An exploratory study on apache projects," in *Proceedings of ICSME*. IEEE, 2020, pp. 629–639.