Online Supplement to "Collaboration Process Pattern Approach to Improving Teamwork Performance: A Data-Mining-Based Methodology"

Shaokun Fan
College of Business, Oregon State University, Corvallis, Oregon 97331, USA
Shaokun.fan@oregonstate.edu

Xin Li, J. Leon Zhao
Department of Information Systems, College of Business, City University of Hong Kong, Kowloon, Hong Kong SAR
Xin.Li.PhD@gmail.com; jlzhao@cityu.edu.hk

---

1. Additional Literature Review

1.1. Studies on Software Evolution

One particular perspective previous research often takes to study software development log is the software development life cycle. For example, Kemerer and Slaughter (Kemerer and Slaughter 1999) study software evolution based on software maintenance logs. They discovered routines of maintenance actions (such as testing, coding, etc.) in the evolutionary life span of each software system. They also studied the impact of software systems' characteristics on the appearance of different maintenance actions (Kemerer and Slaughter 1997, Barry et al. 2003). These studies take one software system as the unit for analysis, and explore recurrent actions in the system's development life cycle. Inspecting at a finer granularity, Crowston and Scozzi (Crowston and Scozzi 2008) studied the recurrent actions performed at the issue level in the bug-fixing process by coding messages recorded in issue-tracking systems. Their observed routines were mostly sequential, such as submit, fix, and close. In general, these previous studies provide insights on managing software projects.

There are also studies examining the evolution of software using software source code, particularly functional calls, which are the outputs of programmers' collaborative work. Stewart et al. (Stewart et al. 2006) studied how the complexity of program functional calls changes over time. Using open source software project data, they found patterns in the evolution of software size and functional call complexity (Daniel et al. 2009, Darcy et al. 2010). Their findings help us understand software changes, but do not explain the reasons for such changes.

1.2. Techniques for Mining Processes, Sequences, and Graphs

From a technical perspective, identifying patterns and routines from tracking logs (or logs of other collaboration systems) is related to process mining (Huang and Kumar 2007, Huang and Kumar 2012) , sequence mining (Han et al. 2000), and graph mining (Cook and Holder 1994).

Although collaborations may contain parallel and/or circular routines, the tracking logs are often stored in a sequential manner, where the activities are associated with time. Thus, from a data mining perspective, identifying patterns from tracking logs is related to sequence mining (Agrawal and Srikant 1995), which searches for frequent sequential structures from data. In data mining fields, several sequence mining algorithms (Han et al. 2007) were developed to tackle various problems, such as the famous DNA analysis application (Han et al. 2001). In sequence mining, the identified routines are only sequences.

Process mining deals with the problem of reverse engineering a graphical process model with parallel and/or circular structures from a sequence business process logs (van der Aalst et al. 2004). Since there is a process model governing the set of cases in process logs, the task is to identify common interaction structures as components of the process model, such as using Petri nets. Process mining is primarily focused on identifying relationship among structured tasks in business processes (van der Aalst et al.

2004). Process mining is similar to collaboration pattern mining in the sense that both approaches try to identify graph structures from sequential log data. However, people and their roles in a team are very important for collaboration process management and have not been considered by process mining yet.

In collaboration, the underlying dependencies between tasks, actors, and resources are essentially a graph. Thus, identifying routines in collaboration is also related to graph mining (Cook and Holder 1994), which aims to discover frequent subgraphs from a graph. Graph mining has been widely used in bioinformatics and website analysis (Koyutürk et al. 2004, Milo et al. 2002). However, these studies often ignore the time relations between nodes, which cannot be directly applied to the collaboration context.

2. Data Preprocessing

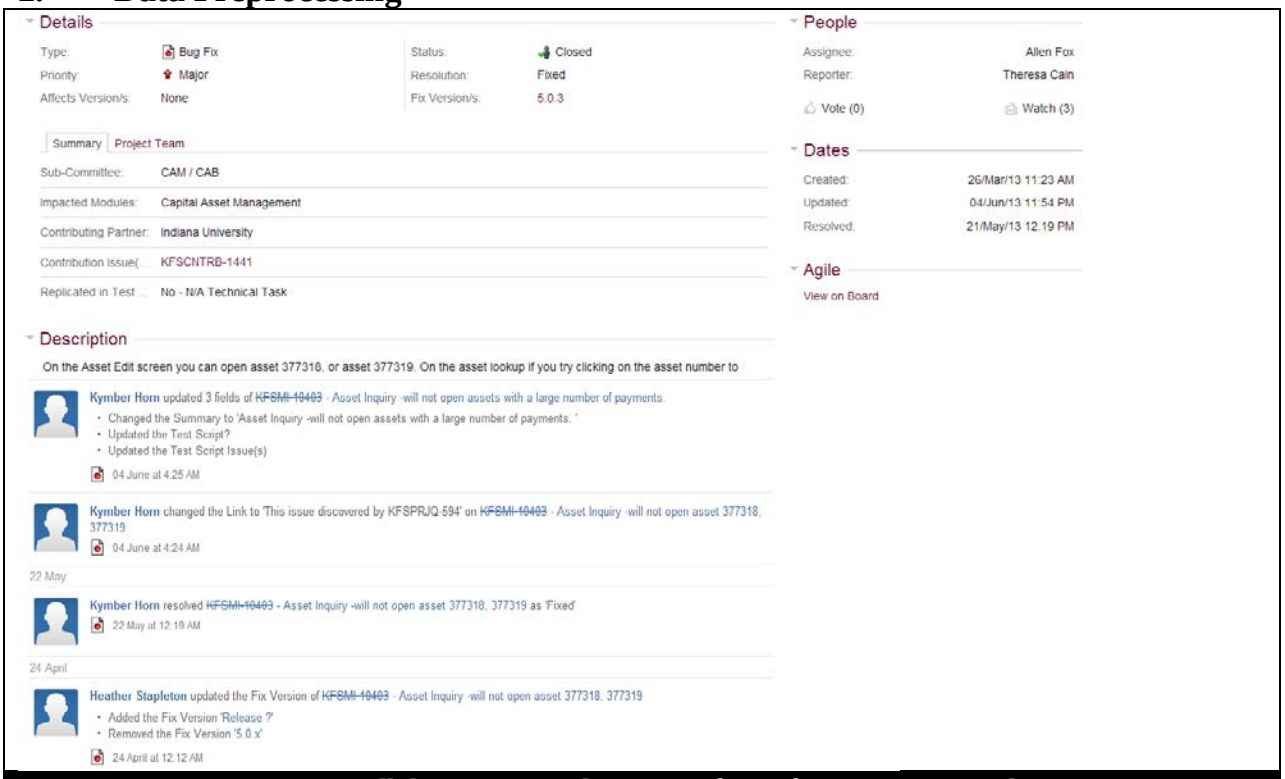

Figure A1. Collaborative Tracking Logs for Software Issue Resolution

By 2010, the community had successfully released four software packages. Our dataset contains the community's issue resolution data from February 2005 (the beginning use of the system) to June 2009, which includes 14,049 issues. As shown in Table 1, we identify three types of (in total 3,729) issues that cannot be used in this research since they are not involved with any collaboration or development efforts:

- *Unprocessed issues* are issues that have not been dealt by the developers.
- *Invalid issues* are issues that considered as unsolvable by the development team, such as the ones without complete description, cannot be reproduced (probably false reporting), and decided not to be fixed.
- *Trivial issues* are issues that needs no substantial efforts, such as duplicate reports of issues (which will be addressed in the efforts of other issues) and issues that will be naturally fixed when working on other issues.

The remaining 10,320 issues are valid issues that involve development efforts. In general, each issue takes from a few days to a few years to fix. The valid issues belong to about 30 issue types. The largest category is "bug fixing," which contains 4,973 issues. The second-largest category is "regular tasks," which contains 3,880 issues. There are mainly two types of regular tasks: "creating maintenance documents" and "conducting scenario tests on developed modules." The third- and the fourth-largest types are "system improvement" and "system enhancement," which in total contain 810 issues. Not only

are the names of two categories similar, we found the issues in these two categories both focus on improving features of the current system. Thus, we pool them to one class called "system enhancement." The remaining issue types contain a very small number of issues. Thus, we pool them to one category named "others," totaling 657 issues[1]. Table 1 also reports the number of four types of issues in atomic and non-atomic issues and in open and closed issues. The distributions are generally consistent in the different settings. The "bug fixing" category occupies about 50% issues. "Regular tasks" occupy about 35%. "System enhancement" occupies about 9%.

Table A1. Summary of the dataset

Issue type								
Unprocessed Issues	Just reported, haven't been processed.						***# of Issues***	
							53	
Invalid Issues	Not completely described						56	
	Cannot be reproduced						495	
	Decide not to fix (by the management team)						1,115	
Trivial Issues	Duplicately reported issues						1,824	
	Fixed by other issues (no effort is invested)						186	
Total Dropped Issues:							**3,729**	
			Atomic		*Non-Atomic*		*All*	
Valid Issues	*Completed issues*	*Bug Fixing*	3,545	6,598	1,046	2,183	4,591	8,781
		Regular Tasks	2,269		909		3,178	
		System Enhancement	398		122		520	
		Others	386		106		492	
	Open issues	*Bug Fixing*	284	1,214	98	325	382	1,539
		Regular Tasks	553		149		702	
		System Enhancement	220		70		290	
		Others	157		8		165	
Total:			**7,812**		**2,508**		**10,320**	

3. Collaboration Process Patterns Mining

3.1. Definition and Theorems for Process Pattern Mining

***Definition* 9: Inclusion of activity dependency structures**. Given a set of process instances, an activity dependency structure *ADS* is said to be included in another activity dependency structure *ADS'*, if there exists a process segment *PS* mapped to *ADS* and included in *ADS'*.

Lemma 1. If an activity dependency structure *ADS* is NOT included in another activity dependency structure *ADS'*, given candidate set *S*, the unique support of *ADS* does not change if *ADS'* is added into *S*.
Proof. Given an arbitrary process instance *PI*:

1) If *ADS* is NOT uniquely supported by *PI*, there does not exist any *PS* that can be mapped to *ADS* given *S*. Adding any other activity dependency structures into *S* has no effect on this fact.
2) If *ADS* is uniquely supported by *PI* under S, there is a process segment *PS* in *PI* that can be mapped to *ADS*. Assume *PS* is also included in *ADS'*. Then, there exists a *PS'* mapped to *ADS'*, of which *PS* is a subsequence. Thus, *ADS* is included in *ADS'*, which conflicts with the condition. Thus, *PS* cannot be included in *ADS'*. The unique support of *ADS* does not change if *ADS'* is added into *S*.□

[1] The results reported in this research are also conducted on original issue types, which do not have significant difference. Thus, we pool small categories together for the sake of simplicity and semantic interpretability.

Lemma 2. If an activity dependency structure *ADS = (V; E)* is included in another activity dependency structure *ADS' = (V'; E')*, then *ADS* is subgraph isomorphic to *ADS'*, i.e., there exists an injective function $f : V \rightarrow V'$, such that $f(V) \subseteq V'$ and $\forall (v_i, v_j) \in E \Rightarrow (f(v_i), f(v_j)) \in E'$.

Proof. When *ADS* is included in *ADS'*, there exists a process segment *PS* of process instance *PI* that can be mapped to *ADS*, which is a subsequence of a process segment *PS'* of *PI* that can be mapped to *ADS'*. Without loss of generality, we assume $PI = <a_1, \ldots, a_n>$, $PS' = <a_m, a_{m+1}, \ldots, a_{m+k}>$ where $m >= 1$ and $m + k <= n$. Since *PS* is a subsequence of *PS'*, we further assume $PS = <a_{m+l}, a_{m+l+1}, \ldots, a_{m+j}>$, where $l >= 1$ and $j <= k$.

We first prove the subgraph isomorphic relationship by finding the mapping function between two activity dependency structures. According to Definition 4, vertexes in *ADS* are derived from activities in *PS*, $v_i = (L_{PS}(m_i,r_i), r_i, c_i)$. Correspondingly, vertexes in *ADS'* are $v_i' = (L_{PS'}(m_i,r_i), r_i, c_i)$. Noticing that $L_{PS}()$ is simply a renaming function, there exists a one-to-one mapping f_1: $(m_i,r_i) \rightarrow L_{PS}(m_i,r_i)$. Similarly, we have f_2: $(m_i,r_i) \rightarrow L_{PS'}(m_i,r_i)$. For the activities in *PS*, which is a subsequence of *PS'*, we can define a mapping function between the actor identifiers in *ADS* and *ADS'*: $f_0 = f_2 \circ f_1^{-1}$. With the help of this function, we can create a one-to-one mapping between vertexes in *ADS* and *ADS'*: f: $(L_{PS}(m_i,r_i), r_i, c_i) \rightarrow (f_0(L_{PS}(m_i,r_i)), r_i, c_i) = (L_{PS'}(m_i,r_i), r_i, c_i)$. With this function, we have $f_0(V) \subseteq V'$ *and* $\forall (v_i, v_j) \in E \Rightarrow (f_0(v_i), f_0(v_j)) \in E'$. □

Lemma 3. If an activity dependency structure *ADS* is included in another activity dependency structure *ADS'* and *ADS* is not isomorphic to *ADS'*, then *ADS'* cannot be included in *ADS*.

Proof. Based on Lemma 1, *ADS* is subgraph isomorphic to *ADS'*. There exists an injective function f_1: $V \rightarrow V'$, such that $f_1(V) \subseteq V'$ and $\forall (v_i, v_j) \in E \Rightarrow (f_1(v_i), f_1(v_j)) \in E'$. Assume *ADS'* is also included in *ADS*, then *ADS'* is subgraph isomorphic to *ADS*. There exists an injective function f_2: $V' \rightarrow V$, such that $f_2(V') \subseteq V$ and $\forall (v'_i, v'_j) \in E' \Rightarrow (f_2(v'_i), f_2(v'_j)) \in E$. In other words, we can create an injective function $f_2 \circ f_1$, such that $f_2 \circ f_1(V) \subseteq V' \subseteq V$ and $\forall (v_i, v_j) \in E \Rightarrow (f_2 \circ f_1(v_i), f_2 \circ f_1(v_j)) \in E$. Thus, $f_2 \circ f_1$ is a mapping from *ADS* to itself. This is only possible if *ADS* and *ADS'* have an isomorphic relationship to each other. This contradicts the condition that *ADS* is not isomorphic to *ADS'*. Therefore, *ADS'* cannot be included in *ADS*. □

Theorem 1. If an activity dependency structure *ADS'* is subgraph isomorphic to *ADS* and *ADS'* is not isomorphic to *ADS*, the unique support of *ADS* does not change if *ADS'* is also in the candidate set.

Proof. Since *ADS'* is subgraph isomorphic to *ADS* and *ADS'* is not isomorphic to *ADS*, *ADS* is NOT subgraph isomorphic to *ADS'*. According to Lemma 3, *ADS* is not included in *ADS'*. According to Lemma 1, the unique support of *ADS* does not change if *ADS'* is also in the candidate set. □

Theorem 2. If there is no subgraph isomorphic relationship between two activity dependency structures *ADS* and *ADS'*, their unique supports will not be affected by whether the other is in the candidate set.

Proof. Since there is no subgraph isomorphic relationship between two activity dependency structures *ADS* and *ADS'*, *ADS* is not included in *ADS'* and *ADS'* is not included in *ADS*. According to Lemma 1, their unique supports will not be affected by whether the other is in the candidate set. □

3.2. Computational Complexity for Subgraph Isomorphic Relationship Detection

In Phase 2 of the mining algorithm, we identify subgraph isomorphic relationships between activity dependency structures to prioritize activity dependency structures. As shown in Figure 4, lines 5-19 use a pair-wise approach to determine subgraph isomorphic relations among activity dependency structures. The subgraph isomorphic detection part is most time consuming step in our approach and has been proved to be NP-complete (Garey and Johnson 1979). There are many studies in Computer Science that tackles this problem. To judge subgraph-isomorphic for graphs with N nodes, previous research has achieved

time complexity from $O(N^2)$ in the best case to O(N!N) in the worst case (Cordella et al. 2004). Such an algorithm is capable to handle graphs with hundreds of nodes (Cordella et al. 2004).

4. Impact Assessment Using Time-dependent Cox Regression

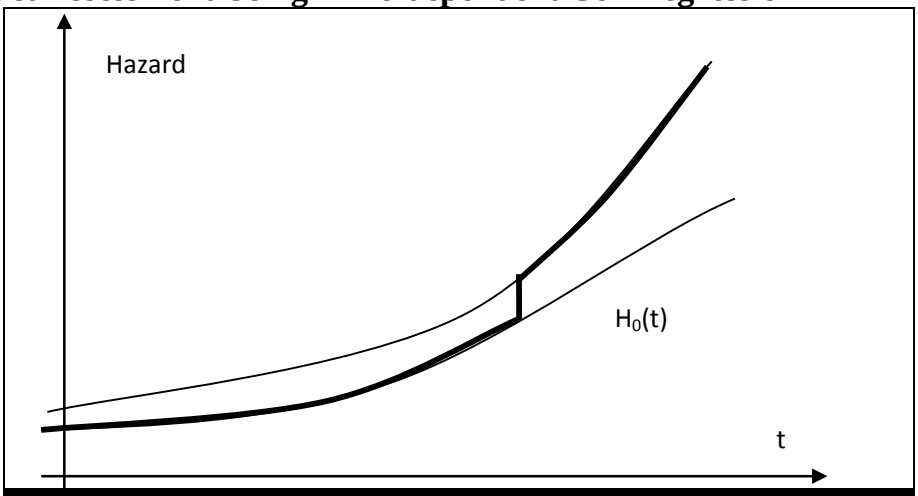

Figure A2. Time-dependent Hazard of a Pattern (Occurred Variable)

5. Additional Empirical Results

5.1. Identified Collaboration Patterns

Table A2. Summary of Identified Collaboration Patterns

	Valid Atomic Issues					Valid Issues/All Issues					
	Unique	Occurrence		Length		Unique	Occurrence		Length		
	Support	Mean	Std. Dev	Mean	Std. Dev	Support	Mean	Std. Dev	Mean	Std. Dev	Group
P1	1,685	1.21	0.66	3.40	1.85	2,140	1.26	0.80	3.54	1.90	Short_PreManage_Seq
P2	1,318	1.23	0.69	3.46	2.76	1,741	1.28	0.84	3.69	2.68	Short_PostManage_Seq
P3	1,276	1.10	0.34	5.94	2.08	1,930	1.15	0.47	6.16	2.37	Long_PostManage_Seq
P4	1,083	1.07	0.27	6.09	1.95	1,682	1. 10	0.36	6.39	2.25	Long_PostManage_Seq
P5	1,062	1.05	0.22	5.77	2.84	1,551	1.08	0.29	6.03	3.14	Two_Worker_Loop
P6	1,018	1.16	0.46	6.24	2.25	1,642	1.26	0.66	6.40	2.33	Manager_MultWorker_Loop
P7	989	1.30	0.73	7.28	4.68	1,665	1.42	0.91	7.67	4.58	Manager_Worker_Loop
P8	983	1.10	0.37	7.40	4.71	1,593	1.18	0.52	7.78	4.49	Manager_Worker_Loop
P9	977	1.00	0.03	3.50	1.64	1,253	1.00	0.04	3.71	2.66	Short_Worker_Seq
P10	921	1. 72	1.31	5.23	2.15	1,383	1. 81	1.38	5.26	2.15	Long_Worker_Seq
P11	913	1.28	0.63	6.48	2.55	1,445	1.38	0.77	6.73	2.89	Two_Worker_Loop
P12	892	1.12	0.40	5.70	2.04	1,456	1.18	0.51	5.91	2.17	Long_PreManage_Seq
P13	891	1.26	0.56	4.40	1.76	1,270	1.27	0.58	4.50	2.15	Short_Worker_Seq
P14	884	1.13	0.40	7.23	3.31	1,409	1.18	0.50	7.56	3.47	Manager_Worker_Loop
P15	849	1.30	0.67	6.76	2.73	1,370	1.39	0.78	6.94	2.99	Two_Worker_Loop
P16	824	1.34	0.77	6.78	2.90	1,389	1.45	0.92	7.25	3.61	Manager_Worker_Loop

5.2. Impact on Teamwork Efficiency

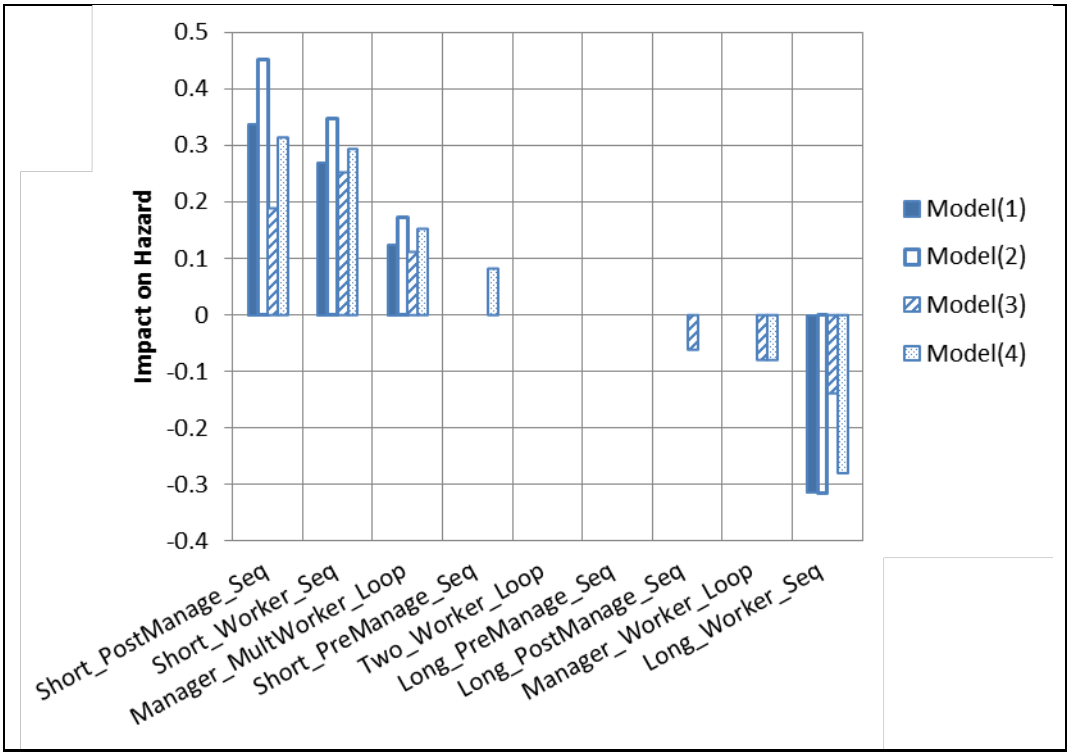

Figure A3. Impact of Collaboration Patterns on Efficiency

Case 1: The issue evaluates the functionality of "Cancel." It contains a Short_PostManage_Seq pattern. The worker first looks into the code of "Cancel" and assignee further checks and evaluates the task.
Action log:
[Worker A] comment: "Just an FYI that the method public void cancelExtractedCreditMemo (CreditMemoDocument cmDocument, String note) is now being called from the PDP cancel processing code. I will commit the code tonight or tomorrow."
[Assignee B] comment: "Thanks Jay I saw the todo there although I haven't hooked it up to do anything yet. I will before I commit (which should also be tonight or tomorrow.) Just checking but that method is only called by PDP right? Because I believe I have rules in there to prevent cancelling after extraction on the Kuali side."
[Assignee B] comment: "closing this issue because the focus of this task is done (i.e. totally revamping cancel and sharing logic between PREQ/CM) I think at this point it would be more productive to open new tasks for bugs and feature additions as I run into them."
[Assignee B] closes the issue.

Case 2: The issue is to verify whether the note deletion function works well. It contains a Two_Worker_Loop. Workers A and B worked together to solve this issue and did not know whether the issue should be closed or not. They waited for about two weeks before closing the issue.
Action log:
(Worker A and B work interactively on the issue.)
[Worker B] comment: "Notes can be added and deleted on maintenance docs - but are throwing server errors when deleting attachments- KFSMI-3791. Not sure if this one should be closed?"
(The process pauses for about two weeks and none of the workers does anything.)
[Worker A] comment: "I am fine with closing the issue" and closes the issue.

Case 3: The issue is to change data fields in the purchase order. The issue contains a Short_PreManage_Seq pattern. The assignee pointed out how to solve the issue and the worker followed the instruction and finished the task.

Action log:

[Assignee A] comment: "It can be easily changed in the po pdf to use itemQuantity instead of ordered-Quantity."

[Worker B] comment: "Thanks. I'll make the changes to remove the orderedQuantity and change it to your itemquantity. Let me know if you see any problems."

[Worker B] comment: "This is good now."

[Worker B] closes the issue.

Case 4: The issue fixes the bug of displaying wrong document ID. It contains a Short_PreManage_Seq pattern. Two managers first discussed the problem. But the discussion does not seem to be related to the resolution of the issue by the worker.

[Manager A] comment: "I guess my question is, is it as helpful to have this as part of the note now that they can easily see the documents in date order in the search? Seems like it's not as valuable as it was in EPIC."

[Manager B] comment: "A - This is an old one so I'm not sure if it is still happening, but could you take a look? Thanks."

[Manager B] comment: "I'm seeing something even stranger now than what you saw in the past."

[Worker C] comment: "The retransmit doc ID is now showing up as asked for."

[Worker C] closes the issue.

5.3. Across Dataset Generalizability

Table A3. Descriptive Statistics of The Second Dataset

	Valid Issues		
	n	Mean	*Std. Dev*
Resolved	12,784	0.807	0.395
Priority	12,784	3.804	0.716
No. of Actors	12,784	2.791	1.123
No. of Actions	12,784	5.519	3.061
No. of Comments	12,784	2.178	2.073
No. of Changes	12,784	3.342	3.406
has_Assignee	12,784	0.818	0.386
is_collaboration	12,784	0.669	0.471
ActorExp	40,295	481.2854	921.48
ActorSameTypeExp	40,295	327.2859	746.33
ActorSameProjExp	40,295	352.5512	655.11
ActorParalIssue	40,295	274.6926	397.17

References

Agrawal R., Srikant R. (1995). Mining sequential patterns. *International Conference on Data Engineering*. 3-14.

Barry E.J., Kemerer C.F., Slaughter S.A. (2003). On the uniformity of software evolution patterns. *Software Engineering, 2003. Proceedings. 25th International Conference on.* IEEE. 106-113.

Cook D.J., Holder L.B. (1994). Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research,* **1**(**1**)**,** 231-255.

Cordella L.P., Foggia P., Sansone C., Vento M. (2004). A (sub)graph isomorphism algorithm for matching large graphs. *Ieee Transactions on Pattern Analysis and Machine Intelligence,* **26**(**10**)**,** 1367-1372.

Crowston K., Scozzi B. (2008). Bug fixing practices within free/libre open source software development teams. *Journal of Database Management (JDM),* **19**(**2**)**,** 1-30.

Daniel S., Stewart K., Darcy D. (2009). Patterns of Evolution in Open Source Projects: A Categorization Schema and Implications.

Darcy D.P., Daniel S.L., Stewart K.J. (2010). Exploring complexity in open source software: Evolutionary patterns, antecedents, and outcomes. *System Sciences (HICSS), 2010 43rd Hawaii International Conference on.* IEEE. 1-11.

Garey M.R., Johnson D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York.

Han J., Cheng H., Xin D., Yan X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery,* **15**(**1**)**,** 55-86.

Han J., Jamil H., Lu Y., Chen L., Liao Y., Pei J. (2001). DNA-miner: a system prototype for mining DNA sequences. *SIGMOD Rec.,* **30**(**2**)**,** 618.

Han J., Pei J., Mortazavi-Asl B., Chen Q., Dayal U., Hsu M.-C. (2000). FreeSpan: frequent pattern-projected sequential pattern mining. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Boston, Massachusetts, United States: ACM. 355-359.

Huang Z., Kumar A. (2007). A Study of Process Mining: Quality and Accuracy Tradeoffs. *Proceedings of the 4th Workshop on Business Process Intelligence.*

Huang Z., Kumar A. (2012). A Study of Quality and Accuracy Trade-offs in Process Mining. *INFORMS Journal on Computing,* **24**(**2**)**,** 311-327.

Kemerer C.F., Slaughter S. (1999). An empirical approach to studying software evolution. *Software Engineering, IEEE Transactions on,* **25**(**4**)**,** 493-509.

Kemerer C.F., Slaughter S.A. (1997). Determinants of software maintenance profiles: An empirical investigation. *Journal of Software Maintenance-Research and Practice,* **9**(**4**)**,** 235-251.

Koyutürk M., Grama A., Szpankowski W. (2004). An efficient algorithm for detecting frequent subgraphs in biological networks. *Bioinformatics,* **20**(**suppl 1**)**,** i200-i207.

Milo R., Shen-Orr S., Itzkovitz S., Kashtan N., Chklovskii D., Alon U. (2002). Network motifs: simple building blocks of complex networks. *Science,* **298,** 824-827.

Stewart K.J., Darcy D.P., Daniel S.L. (2006). Opportunities and challenges applying functional data analysis to the study of open source software evolution. *Statistical Science,* **21**(**2**)**,** 167-178.

van der Aalst W., Weijters T., Maruster L. (2004). Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering,* **16**(**9**)**,** 1128-1142.