Deciphering the Story of Software Development through Frequent Pattern Mining

Nicolas Bettenburg
Software Analysis and Intelligence Lab (SAIL)
Queen's University, School of Computing
Kingston, Ontario K1N 3L6 Canada
nicbet@cs.queensu.ca

Andrew Begel
Microsoft Research
Microsoft Corporation
Redmond, WA, USA
abegel@microsoft.com

***Abstract*—Software teams record their work progress in task repositories which often require them to encode their activities in a set of edits to field values in a form-based user interface. When others read the tasks, they must decode the schema used to write the activities down. We interviewed four software teams and found out how they used the task repository fields to record their work activities. However, we also found that they had trouble interpreting task revisions that encoded for multiple activities at the same time. To assist engineers in decoding tasks, we developed a scalable method based on frequent pattern mining to identify patterns of frequently co-edited fields that each represent a conceptual work activity. We applied our method to our two years of our interviewee's task repositories and were able to abstract 83,000 field changes into just 27 patterns that cover 95% of the task revisions. We used the 27 patterns to render the teams' tasks in web-based English newsfeeds and evaluated them with the product teams. The team agreed with most of our patterns and English interpretations, but outlined a number of improvements that we will incorporate into future work.**

***Index Terms*—software teams; task tracking; mining software repositories; pattern mining**

I. Introduction

Software teams commonly record their progress on software development tasks in a work item repository such as Team Foundation Server (TFS), Trac, or Bugzilla. These repositories often employ general data storage architectures and require form-based I/O. Thus, teams are required to define a schema of fields and values to represent the current state of their work practices. As team members do their work, they must "encode" the activities they perform into values stored in particular combinations of fields in the repository. When reading the history of a work item, for example, when assigned a bug, team members must "decode" those fields and values back into the specific practices they describe.

In a multi-case study of four teams, all working within one software product group inside Microsoft, we learned *how* teams appropriate the fields of the work item repository for their own process recording needs. We asked team members to explain to us in English their interpretation for each of the field changes in revisions to several work items, e.g., "what is the meaning of setting the priority field to 1?". Many work items contained multiple field edits that corresponded to one or more activities, and team members found that individually, those activities could be described in up to seven independent, English sentences per revision of each work item, e.g., the creation of a task on which work has already started. Unfortunately, they did not always succeed in describing each of the activities; sometimes they missed a few, or were misled by several field edits, which could not easily be placed into a single activity, e.g., commits to the version control system were believed to be actual bug fixes, but when observing surrounding field changes turned out to be additions of `printf` debugging code.

To help decipher the task database and reveal the story of the task within, we propose a novel method for automatically discovering and classifying frequent field edit patterns found in work item repositories. These frequently occurring patterns represent fields that are often edited together and thus describe a particular development activity recorded in the task repository. Our method is easily scalable because we use a quantitative statistical approach, called frequent pattern mining (FPM). Based on this approach, we have been able to create a prototype of a web-based newsfeed that can represent a chronological English language history using the frequent patterns as news item templates.

We applied our method to two years of a Microsoft product team's task repository containing 801,621 work item revisions. The team uses a total of 71 unique fields to record their work. Our analysis abstracts 375 unique combinations of field edits into 27 patterns that cover 95% of the instances of field edits. In addition, we conducted focus group with one of the teams working on that product to measure their subjective understanding of the newsfeed. Our preliminary results are promising. The team reported that most of the newsfeed made sense to them, but outlined a number of possible improvements that we plan to incorporate into future work.

II. Formative Study

In order to record and express their activities in their task repository, software team members must transform their recollection of their work into values for a set of fields in a form-based editor. We hypothesize that this cognitively demanding recording and decoding process impedes the ability of the engineers' to make sense of each task's ongoing storyline. To identify the extent and possible sources for these difficulties, we interviewed four randomly-selected software developers, testers, and program managers each working on different,

loosely-coupled teams collaborating together to build a single, large, enterprise product.

We asked the four interviewees to read through a printed transcript of the individual field values set in every revision of a task from their Team Foundation Server that they had worked on in the two weeks preceding the interview. We were interested to hear how each expressed the activities recorded in the task history in his own vernacular. We recorded the interviews and transcribed them verbatim.

Each engineer explained the task as he would tell a story. Instead of explaining each field state independently of the others, **they frequently referred to the previous and next states** of the field to tell us **what it meant** for the field's value to have changed. The engineers had an easy time explaining the purpose of many of the fields. For example, the "How Found" field was used by a tester to say whether he had discovered the defect via his own test, an ad hoc test, or an automated test.

The engineers also told us **why a field was changed** to a particular value. One said that when his manager changes the "Assigned To" field to Jason in Revision 1, it means that Jason should triage the problem (not fix it). But, when he changed "Assigned To" again in Revision 2, he said he realized it some other team's issue, and should be triaged further by one of their team members. Since, it was not uncommon to modify several of the fields in groups, rather than one at a time, in Revision 2, the manager also changed "Substatus" from "Not Started" to "Investigation." The combination of field changes indicated that he had now found the person who could look at the problem.

The interviewees reported some cases of **incorrect, missing, and misleading** data, just as was found by Aranda et al. [1]. Most pointed out **omissions**, such as not recording decisions that took place during a face to face meeting; **errors**, such as that the "author" recorded by the repository as having reassigned the task to someone in another team was not the person who was responsible for the task, but was simply the meeting scribe recording the decision made by the true author at the weekly triage meeting; and **misleading data**, such as mischaracterizing the team members' relative contributions to a task to make the team's workload appear more balanced to his management chain.

A new kind of problem appeared, however. Interviewees had trouble **interpreting** some of the combinations of field edits that impeded their ability to recall the events of the task. In some cases, they noticed that a field that should have been set was missing from the edit. We noticed that two task revisions were done by the same editor, within a few minutes of one another. When we pointed this out, the interviewees told us that since there were so many individual fields to edit on the forms, and they had to set them all from scratch every time, they would sometimes forget to set a few of them when the task edit was submitted. They would then remember, re-open the task edit, and submit another revision with the missing field. For example, during a triage meeting, one of our interviewee's team members had to change the milestone in which the task would be completed. He updated the "Iteration Path" field to new hierarchical value that represented the milestone, but forgot to add the milestone name itself (the leaf of the path) to the "Internal Milestone" field, and had to add it in the next revision, 2 minutes later.[1]

In other cases, they saw an unexpected field being set in a revision, and were confused as to its purpose. Eventually, they realized that there were two different activities encoded into the same task revision, and the extra field was part of the second activity. The developer told us that he had done this often. After creating and triaging a task, he spent an afternoon working on it. He had to then indicate that work on the task progressed by changing the "Completed Work" and "Remaining Work" fields. However, while working on the task, he realized that the task needed work in a different subsystem than he initially assumed, recorded this knowledge in the "Area Path" and "Baseline Work" fields. Since he was busy working toward his deadline, he updated all of those fields in a single task edit right before the weekly meeting, so that the task would be current during the meeting. The developer told us that keeping the task repository up to date has a high overhead, so updates to tasks are often carried out in bulk. Thus engineers may update multiple fields that are conceptually different and do not belong together in a single revision in the repository.

Combining conceptually different field edits into the same task revision makes it difficult to tell which set of fields represent each activity. In the next section, we outline the use of frequent pattern mining, a statistical analysis, to recover frequently occurring field edit patterns and make it easier for engineers to decipher the story of the task.

III. Activity Analysis Method

To recover the true software development activities of the product teams, we use a statistical knowledge discovery approach called Frequent Pattern Mining (FPM) [2]. FPM has most popularly been used to analyze commercial transaction databases of customers' shopping baskets [3], enabling the discovery of patterns, such as "people who buy diapers also buy beer." In software engineering, Zimmermann et al. use FPM to mine the co-occurrences of files checked into a source control repository [4]. In our method, we use FPM to discover field edit relationships in each revision of the task repository, such as "developers who edited Completed Work also changed Remaining Work." In this section, we explain how FPM works and how we use it in more detail.

Data Representation. To begin FPM, we must represent the transitions between each revision in a task as a binary vector in a k-dimensional space. Let k be the number of fields in the task repository schema. We represent each field as one entry in a k-dimensional binary vector, where a 1 indicates that the field value changed from the previous revision, and 0 means it did not. The collection of all transitions over all

[1]The apparent duplication of field values enables filtering on the milestone name during triage meetings.

Change	Effort Required	Priority	Effort Spent	Assigned To
1	1	0	1	1
2	0	1	0	1
3	1	1	1	0
4	0	1	0	1

TABLE I
FOUR CHANGES TO FOUR FIELDS FOR A TASK.

tasks is represented as a $k \times m$ "binary incidence matrix," (where m is the total number of revisions counted over all tasks), and forms the input into the FPM algorithm. We use the `aRules` [5] package in R to execute FPM, and generate lists, called "itemsets," of the fields that are often changed together.

Maximally Frequent Itemsets. We use a variant of FPM that looks for the "maximally frequent itemsets." We want to capture the most detailed itemsets that retain enough support to represent a frequent co-occurring pattern of field changes. Otherwise, they might have been produced by chance, rather than by a true association.

Consider Table I. It shows four changes from one task in our repository. Each cell shows a 1 if the field heading that column was changed in the row for that change, and a 0 if it did not change. We can see that fields `Effort Required` and `Effort Spent` were changed together 2 / 4 times. Hence, the support of their itemset, {*Effort Required, Effort Spent*}, is $\frac{2}{4} = 0.5$. Two other sets of edited fields include the itemset {*Effort Required, Effort Spent*}: {*Effort Required, Effort Spent, Assigned To*} and {*Effort Required, Effort Spent, Priority*}. However, the support for these itemsets is only half as large as their common superset. They both appear just 1 / 4 times ($sup = \frac{1}{4} = 0.25$).

The maximally frequent itemsets variant of FPM computes exactly those itemsets that lie on the boundary between most specific and noise. Conceptually, itemsets are maximally frequent if adding another item to the set would make the itemset too infrequent. This boundary condition is defined as a parameter to the algorithm, called the *support threshold*. It sets a lower bound on the number of times the particular set of fields actually changed together in the input data. If we set support to 0.01, the algorithm would filter out any itemset that occurred in less than 1% of all revisions, thus filtering out infrequently co-occurring field changes, and helping us focus on the signal in the result, rather than the noise.

IV. RESULTS

To test our activity analysis approach, we applied FPM to a subset of task repository data from the same product team we encountered in our formative study. Since they were approached first, and readily available for our study, we chose two of the teams that our four interviewees worked with (Team A and Team B) and subset the data to just the tasks that any of the chosen team members had ever edited over the last complete release of the product. This way, we could take our analyses to that product team and have them help us interpret our results.

We collected a total of 1,314 tasks: 522 from Team A, and 792 from Team B. These tasks contain a total of 16,477 revisions: 6,370 for Team A, and 10,107 for Team B. These, in turn, describe a total of 82,986 field changes: 37,915 for Team A, and 45,071 for Team B. There were 375 unique combinations of field changes in the task data.

A. Activity Patterns

From the product teams' tasks, our FPM analysis method distilled 27 activity patterns, which we categorized into six kinds of activities.

- **Scheduling** activities are carried out to plan the execution of tasks and situate them in the software development life-cycle. Effort estimation is also contained in this category.
- **Status** reporting activities show engineers and managers the current state of the task (e.g., scheduled, in progress, blocked, finished, ...) and let them take appropriate action.
- **Progress** reporting activities show how far along a task is towards completion, which is essential to making sure deadlines are met.
- **Assignment** activities are used to give the responsibility to other team members, other teams and business units. They are also used to associate tasks with product work scenarios and release themes used by the upper management.
- **Communication** activities help stakeholders of a task to stay connected. These range from simple messages, to the rationale for particular activities. For example, status was set to "Blocked" because no resources were available. Automated tools also communicate through these activities (e.g., build warnings).
- **Other** miscellaneous activities which did not fit in any other categories, such as updating a team-custom field.

To perform a coverage analysis we calculated a histogram of patterns across instances of field changes and found that these 27 patterns covered 95% of the actual instances of field edits found all of the task revisions in the data, thus showing that almost all of the 348 other possible patterns are mainly in the noise.

B. Newsfeed

We built a prototype for a web-based newsfeed that renders the revisions of each task in English. Each of our 27 activity patterns is encoded in an HTML template. Whenever a set of field edits from a task revision matches the pattern, we instantiate the template as a news item. To check for a match, we use represent the revision and the pattern as a bit vector. The pattern bit vectors, however, contain wildcards in their unset bits, so that they can match revisions with extra field edits. A revision may match multiple patterns, in which case, each pattern would generate a news item for the newsfeed.

We evaluated the newsfeed in a focus group setting with the product teams. We showed them newsfeeds for several tasks and asked them to tell us if we got the English explanation

correct. They agreed with most of our interpretations of their activities, but sometimes suggested a better explanation than we had, or recognized individual news items that should have been merged together. They were also able to explain what some of the more obscure fields were for us, which we will apply in future versions of our newsfeed.

Based on our results, we believe that recovery of team activities through frequent pattern mining is a valuable asset for scalable and simple recovery of the story of software development activities.

V. Related Work

Understanding how software engineers enact their software process, record their development activities, and communicate with one another through the stories of their work, and helping them to do so more easily and effectively is a common goal we share with other researchers in this area.

Recording Work in Task Databases. Bertram et al. interviewed small, collocated software development teams on their use of issue tracking systems, and found that each stakeholder has a unique set of reasons for what data they record about their development tasks and activities. This led team members to use the fields in the repository for their own purposes, which were sometimes at odds with the interpretations of others on different teams [6]. The software engineers we interviewed reported similar appropriation of repository fields, resulting in difficulty interpreting the activities of other engineers, even those working on the same task as them.

Summarization of Development Activities. Rastkar et al. [7] and Lotufo et al. [8] both tackle the problem of summarizing the bug reports in the Bugzilla issue tracking systems. Working solely with the freetext discussion fields, they use NLP techniques to identify representative sentences and phrases that can be accumulated into a single paragraph bug summary. Our work focuses on *non-freetext* fields, using FPM to recover frequently co-changing fields that represent activities, and then rendering those activities in colloquial English that the product team would have used. Our newsfeed adorns its news items with the raw freetext of the discussion from each revision in the task.

Frequent Pattern Mining. Zimmermann et. al used FPM to identify source code files that are often checked in together, and found that checkins following one of the patterns were more likely to have defects if they omitted one of the typically co-checked-in source code files [4]. They used their analysis to build an interactive tool to warn engineers of any files that were potentially missing from a checkin. Chang et al. [9] used FPM to identify combinations of actions in a business project that could best predict future defects in the project. We find FPM to be just as valuable, simply to cluster related field edits in order to enable a more concise and abstract interpretation of the activity it represents.

Telling the Story through a Newsfeed. Begel and Zimmermann proposed using an automatically generated newsfeed of activities occurring in a variety of software development-related repositories to help teams maintain awareness of dependent teams, and enable them to make their work practices more transparent to one another [10]. They warned, however, news summarization would be necessary, and difficult, to ensure that the feed's utility did not get overwhelmed by noise. This paper takes an early step toward constructing a useful news summarization function. Holmes and Walker describe another mechanism to reduce the noise in newsfeeds [11]. Their YooHoo system filters incoming checkin messages to remove those modify code on which the developer's own code does not currently have a dependency. Our work shrinks the feed as well, but operates on a single task at a time, abstracting individual edits into larger and fewer sets that make up an activity. Finally, Kuhn and Stocker propose using a set of information visualizations on a software development timeline to entice engineers to manually annotate the timeline with human-understandable explanations of important activities [12]. Our approach uses FPM to reduce the number of distinct activities to a very small number of activity patterns that can be explained by team members, and then used to render almost all of the important activities automatically.

VI. Conclusions

Deciphering the story of development activities is difficult, even when you lived through the story firsthand. Our approach of using FPM to identify frequently co-occurred field edits patterns in task repositories is a promising start to helping software engineers decode software work activities and reveal the real story behind what happened.

References

[1] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *Proceedings of ICSE*. IEEE Computer Society, 2009, pp. 298–308.

[2] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of SIGMOD*. New York, NY, USA: ACM, 1993, pp. 207–216.

[3] M. J. Berry and G. Linoff, *Data Mining Techniques: For Marketing, Sales, and Customer Support*. New York, NY, USA: John Wiley & Sons, Inc., 1997.

[4] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of ICSE*. IEEE C.S., 2004, pp. 563–572.

[5] M. Hahsler, B. Gruen, and K. Hornik, "arules – A computational environment for mining association rules and frequent item sets," *Journal of Statistical Software*, vol. 14, no. 15, pp. 1–25, October 2005.

[6] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," in *Proceedings of CSCW*, 2010, pp. 291–300.

[7] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: a case study of bug reports," in *Proceedings of ICSE*. ACM, 2010, pp. 505–514.

[8] "Modelling the 'hurried' bug report reading process to summarize bug reports," in *Proceedings of ICSM*. IEEE, 2012.

[9] C.-P. Chang, C.-P. Chu, and Y.-F. Yeh, "Integrating in-process software defect prediction with association mining to discover defect pattern," *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 375–384, 2009.

[10] A. Begel and T. Zimmermann, "Keeping up with your friends: function foo, library bar.dll, and work item 24," in *Proceedings of Web2SE*. ACM, 2010, pp. 20–23.

[11] R. Holmes and R. J. Walker, "Customized awareness: recommending relevant external change events," in *Proceedings of ICSE*. ACM, 2010, pp. 465–474.

[12] A. Kuhn and M. Stocker, "Codetimeline: storytelling with versioning data," in *Proceedings of ICSE*. IEEE Press, 2012, pp. 1333–1336.