Representations of User Feedback in an Agile, Collocated Software Team

Michael J. Lee and Amy J. Ko
The Information School | DUB Group
University of Washington
Seattle, USA
{mjslee, ajko}@uw.edu

***Abstract*—Support requests are a major source of feedback in software development. However, like sentences whispered in a children's telephone game, as an issue moves through a software development process, it may undergo many changes, making it difficult to ensure that the problem reported was actually fixed. To better understand how software teams preserve the integrity of user-reported issues, we observed a software team over a 6-month period, analyzing the trajectory of user requests through their software development processes. Our observations revealed several representations of issues, highlighting several points where information about an issue was lost or transformed. Although this information loss appeared to be unavoidable as an issue went through the resolution process, we found it was offset by a web of transactive memory distributed throughout the support and product teams. This memory was reinforced by asynchronous chat about recent and commonly reported issues, and shared notions of what issues should be considered severe or urgent.**

***Keywords*-User-centered design; feedback; issue tracking**

I. Introduction

Technical support is a major source of user feedback in most software development teams. Support requests can reveal defects, the desire for new features, and usability problems that users encounter daily, among a variety of other user concerns. This feedback is a particularly important resource from a user-centered design perspective because, unlike lab-based usability testing, it comes directly from the user and their experience interacting with the software.

One challenge in utilizing support feedback to better satisfy user needs is in preserving its integrity throughout a software process, so that teams can be sure the changes being made to address a matter actually attend to the originally reported issue. While recent studies have specifically examined the role of issue and bug tracking tools in facilitating coordination around issues [2,3,6,12,16], there has been little work to understand the range of ways that software teams represent issues in support tickets, bug reports, discussions of issues, and work plans. For example, a software support member trying to explain a reported issue via e-mail to other team members remotely, might lead to information loss or miscommunication; similarly, a brief description of an issue in a bug report title might appear to capture the original issue, but have subtleties that might cause a patch to overlook the originally reported problem with the user experience. Such information loss may lead a team to fix the wrong issue or erroneously believe a user's issue has been resolved when it has not. Exploring the trajectory of user-reported issues may reveal better ways to capture and represent issues, as well as surfacing better ways for developers and support teams to coordinate and communicate user issues to incorporate into their software.

To better understand how representations of issues affect issue tracking and resolution, we performed a 6-month field study of a software team with 20 full-time and 40 part-time staff. Our study analyzed how issues are represented, through what processes issues are transformed into new representations, to what extent each of these representation types are prone to information loss, and whether there are consequences to information loss. To answer these questions, we attended meetings, observed and interviewed staff, analyzed chat logs, and monitored 2,142 user reported issues, tracking them from initial request through final resolution.

Our results contribute several findings about how a small, collocated software team following an agile process manage and represent user-reported issues. We found that although information loss appeared to be unavoidable, this was largely offset by a web of transactive memory, in which groups collectively encode, store, and retrieve knowledge [19], highlighting the importance of communication in identifying and preserving critical user feedback. We also found that most representations of issues primarily serve not to store actual details about user-reported issues, but rather cues for recalling the combined transactive memory about issues.

II. Related Work

In a recent study, Bertram et al. examined issue-tracking practices of a small, collocated software engineering team from a social perspective [3]. They found that issue-tracking software is more than a repository to track bugs and tasks, but also serves as a multifaceted social medium for communication and coordination for many stakeholders, even though face-to-face exchange is readily available. Our study also concerns issue trackers, but focuses on the issues themselves and how they are represented.

Other related work focuses on the contents of user issue requests and bug reports. This includes guidelines for effective bug reporting, such as Bettenburg et al.'s work exploring the mismatch between what users provide and what developers consider helpful [4], and Zimmermann et al.'s work proposing changes to bug tracking systems to increase relevant information for reported issues [21]. Our work considers many of the same questions, but focuses on user-reported issues and how they are converted into bug reports and work plans by experienced bug report writers.

There have also been several studies on technical support and help requests. One example is Bowers and Marin's work [5] analyzing the process in which incoming phone calls to a bank's help center are opened, closed, and resolved. Their work provides a detailed account of the role of the operators in understanding how customer requests fit with the organizational procedures and terminology. Halverson et al.'s work analyzed the evolution of a CSCW system by analyzing a help desk [10], revealing how the system was designed by bricolage. Our work compliments these studies, exploring how support and software teams collaborate to ultimately resolve user-reported software issues.

Our work is also related to prior work on systems for knowledge management and organizational memory. Lutters and Ackerman have proposed organizational memory systems, showing how they transfer information between groups to better support collaboration [14]. Ackerman and Halverson have described help-line operators working together to solve an individual's request, illustrating the wide variety of resources combined into a working organizational memory that keeps track of how to deal with past issues [1]. This is consistent with subsequent research by Yamauchi et al. [20], who found that service technicians encountering new, difficult problems had a large number of resources that they could refer to for assistance, but preferred asking other technicians for help, or referring to informal tips written by other technicians. This is in line with Cunningham et al.'s work studying technical support workers, who "selectively use resources that will enable them to effectively and efficiently solve problems" [9]. Our study compliments these works, contributing observations in the similar domain of software teams, but explicitly focusing on the relationship between support and development staff.

III. Method

The goal of our study was to learn how software teams represent issues, how these representations change throughout a software process, and how limitations in these representations affect a team's user-centered design (UCD) efforts. To answer these questions, we sought a team committed to user-centered design. We ultimately chose to study a software development team (SDT), which works on the Spark Tool Suite for the local university (names have been changed for anonymity). This group was responsible for the creation, support, and improvement of a suite of web applications designed for faculty staff, and students. This suite included a grade book for courses, a survey generator, and an assignment submission tool, among several other applications used broadly by the university community.

There were several factors behind our choice of SDT: (1) the group was situated closely, allowing convenient access, (2) both researchers were familiar with at least a few of the applications, and most importantly (3) SDT's commitment to UCD, reflected in their mission statement:

> "We follow an iterative, user-centered design and development process that focuses on understanding the needs and experiences of our users. […] our design decisions are based on direct feedback, user research, and findings from usability studies."

This was an important trait because we wanted to observe how the team organized itself to process user feedback in a user-centered manner.

SDT followed a variant of the Scrum development process, an agile, or iterative methodology focused on regular, two-week release schedules called sprint cycles [15,17], with a flat management hierarchy.

We collected data over 6 months of direct observation:

- *Attending, recording, and transcribing team meetings*. This helped us gain familiarity with the team, establish rapport, and understand their practices and processes.
- *Observing the shared office space*. This allowed us to observe the staff's individual work and ask clarification questions about interesting observations.
- *Monitoring the Internet Relay Chat (IRC) channel*. This was important to gain understanding how different team members, particularly those who did not regularly attend team meetings, exchanged information.
- *Monitoring the Issue Tracking System (RT & Bugzilla)*. This allowed us to access the complete set of issues, and enabled us to backtrack through multiple logs to see the progression of an issue, what was worked on, and how it changed throughout the process.

A. Team Structure

SDT consisted of two distinct groups with offices half a mile apart. When necessary to distinguish between the two groups, we will refer to them as the *support team* (on-campus) and the *product team* (off-campus).

The support team's staff included a full time manager and system administrator and 11 part-time student workers, called consultants, who responded to phone and e-mail support requests. Four of the more experienced students, called leads, had additional managerial responsibilities and helped the manager train and coordinate new consultants.

The product team included the development team, design team, quality assurance team, managers, and administration. The full-time staff we interacted with included the developers, designers, the quality assurance (QA) engineer, and the project manager (PM). The part-time, student workers we interacted with here were the quality assurance assistants, namely the quality assurance leads (QA leads).

Communication *within* each site occurred in open office spaces with desks. Staff walked over to a colleague's desk, e-mailed them, or met in conference rooms to discuss questions or concerns. The support team did not hold any team meetings during our observations, which was normal according to the support manager. The product team had a daily scrum meeting at 8:45am, where everyone briefly reported their previous day's progress [17]. In addition, they had biweekly sprint planning meetings, where the agenda was to discuss current progress on assignments, get an update on what other individuals and groups were working on, demo projects, and set 2-week goals [17].

Communication *between* sites took place daily over e-mail and through a shared IRC channel. In addition, a weekly "leads' meeting" took place, where the support manager, QA engineer, and at least one lead and QA lead each, would meet to discuss ongoing support issues and upcoming software patches and features.

B. Support and Development Process

When users needed help, they generally e-mailed the support group with a message describing their issue using either an online form or an e-mail address. The support team's issue tracking software, Request Tracker 3.4.4 (RT) (www.bestpractical.com/rt), automatically processed the e-mails, creating a ticket in its database. If a solution could not be provided for the user, the issue was escalated to a shared IRC-channel populated with personnel from both the support team and product team. The few issues that were not resolvable in IRC discussion were closed in RT and a new report was created in the product team's issue tracker, Bugzilla 3.0 (www.bugzilla.org). Non-urgent issues that were not immediately resolvable (e.g. feature requests) were closed in RT and logged in Bugzilla for future review.

In accordance to the scrum methodology [17], the product team worked in 2-week sprints, releasing major updates to their software on a biweekly schedule. Deciding what reports the team would work on occurred in a biweekly sprint planning meeting which typically took place 2 days before a release. During these meetings, post-it notes on the whiteboard were updated to reflect the new two-week sprint. The developer, design, and QA teams met within their groups to decide what to work on, then each individual created physical representations of the Bugzilla reports on post-it notes and then posted it on the whiteboard. Issues or projects that would take longer than one sprint to complete were reclassified as stories, and written on index cards to post on the whiteboard. In addition to the biweekly sprint planning meetings, morning scrum meetings were scheduled to update post-it notes on the sprint backlog whiteboard. This allowed the product team to visually track members' progress during the sprint.

In all, the team logged a total of 2,142 reports into RT during our observations. There was a median of 17 incoming e-mails per business day, with 302 (14.1%) RT tickets with corresponding IRC conversation, and 223 (10.4%) Bugzilla reports originating from RT tickets.

IV. Results

Our observations revealed five issue representations:

1. User e-mails sent to technical support;
2. The automatically generated RT ticket that the support team used to annotate the e-mail;
3. IRC discussions between the support and product team about issues deserving of escalation;
4. The Bugzilla reports written by product team members to capture the discussed issues; and
5. The post-it notes and index cards used to represent Bugzilla reports in face-to-face meetings.

In this section, we describe each of these representations, analyzing their role in capturing user concerns and how they were transformed into other representations. Further, we examine the information loss that occurs at these transition points, and how SDT coped with this. All names are pseudonyms and any personally identifiable or private information has been replaced with asterisks. All communications are reported verbatim.

TABLE I. Information content of user Requests

Category	Definition	% of sample
Expected behavior	What should have happened?	100%
Observed behavior	What actually happened?	76.6%
University affiliation	Am I a student, faculty, or staff?	57.9%
User actions prior to issue	What did I type or click?	47.2%
Activity context	What was I trying to do?	30.7%
Application	What application was it?	27.7%
Attempted workaround(s)	How did I try to fix it?	16.8%
URL	What was the URL?	16.7%
Personal consequence	How did it affect my goals?	12.6%
Browser	What browser was I using?	12.0%
Operating System	What OS was I using?	10.2%
Pleas for help	Am I pleading or begging?	9.8%
Attachment(s)	Did I include any attachments?	1.3%
Emotional	Am I being emotional?	0.8%

A. Limited Information Provided in Support Request E-Mails

The primary source of user feedback was through e-mail generated by hyperlinks embedded in the team's web applications. Users' messages tended to consist of a few lines of text (as in Fig. 1), limiting what information the team had to understand and resolve the issues.

Prior works have examined the content of similar user-reported content; for example, studies have considered the linguistic content of bug report titles [13], the expectations stated in bug report summaries [8], and the need for reproduction steps [4]. These studies focused heavily on how users state *expected behavior*, *observed behavior*, and *input* leading up to a problem, and so we included these in the set of content categories we looked for in user e-mails (Fig. 1). The team also explicitly requested (but did not require) several details, including the browser being used, the application and URL in which the problem was encountered, and the operating system used; we also analyzed these.

To understand the extent these various types of content were included in requests, we devised the coding scheme shown in Table 1 and selected a uniformly random sample of 268 tickets from the population of 2,142 RT tickets (12.5%) for detailed coding. The sample had the expected number of IRC and Bugzilla reports in proportion to the population. The first author coded the 268 issues using the established categories, marking either yes or no for the presence of the information type; examples of the information we coded are given in Fig. 1. To test the reliability of this categorization, the second author redundantly coded 25% of the sample; 85.6% of classifications were in agreement. Table 1 shows the statistics of the sample data. The numerical values are the percentage of user requests that contained the corresponding categorical information in descending order of prevalence. These results show that users described issues as a contrast between expected and observed behavior.

i sent out a messge several weeks ago drawing your attention to the fact that my courses for the winter quarter of 2010 have NOT been listed in my spark account page up tll now[1,2]. will you please tell me WHY? [4,5] i need to use the grade book for my chinese 412 class[6,7]. please advise and i'd appreciate a prompt answer[2]. thanks.

Figure 1. An annotated user request showing:
1) observed behavior, 2) expected behavior, 3) personal consequence, 4) plea for help, 5) emotion, 6) application, and 7) activity.

B. Resolving Issue Ambiguity with Transactive Memory

The results in the previous section are consistent with prior work on bug reports, showing that most users do not provide the information that teams need to reproduce and resolve problems [4,11]. However, in contrast to the problems this limited information caused in the bug reporting processes in previous studies, the staff in our study reported that users' provided more than enough information for them to infer the problem they were experiencing, even when the users' e-mail was vague. For example:

> Is there a way to turn off the results notification? I find it doesn't give me any useful information (like who took the quiz) and so I'd rather not receive the notification (or I'd rather receive it but with info about who took the quiz).

Here, the user request was characteristic of the type of request received (in that it primarily indicated an expectation), but the user did not say what application they were using, what they meant by "notification," or which quiz they were referring to, all of which were important in either providing help or adding a new feature.

Despite this lack of detail, the support team resolved this issue without any additional information from the user, sending a particularly targeted set of instructions as shown:

> You can turn off the results notification by following these steps:
> 1) Go to the Summary section of your WebQ quiz/survey. 2) Click "Results notification" under Settings. 3) De-select the checkbox "Notify me about submissions to this quiz." 4) Click Save.

Of the sample of 268 tickets in the last section, 86.6% were resolved without requesting additional information from the user, showing that in most cases, the support team found the limited context users provided adequate for inferring the specific issue the user was encountering.

Why was this enough information for the support team to interpret the meaning of users' issues? We found through observations and interviews that the support team relied heavily on *keywords* in the user e-mails to match the reported issue against the staff's knowledge of known issues. A consultant confirmed this in an interview stating, "*I just look for keywords [in the e-mail] to determine what they [the user] are talking about*." During repeated observations of the support team, we learned that the keywords the consultant mentioned were specific words or phrases that would commonly occur for certain tools and their functionality (e.g. "quiz," "submit," "late" for the team's survey-making tool). Therefore, the RT tickets that stored users' issues were less a *container* for a description of an issue and more an *index* into the support teams' knowledge of existing prior issues.

In our observations of the support team, we also noticed staff querying fellow staff for knowledge. Shown in Figure 2, these short, ad-hoc conversations happened between individuals in close physical proximity in the office. In this exchange, the 1st consultant attempted to replicate a user's issue but could not do so. By asking her colleague about a potential cause, she learned that there was an issue the previous day when she was not on duty. She went on to let the user know that it was an isolated incident, and that the developers had fixed the problem. From this point, the 1st consultant was able to answer subsequent questions that appeared to be about this without asking for assistance.

Exchanges likes these were especially important since most support staff worked less than 20 hours per week. This meant that there were regular intervals where staff would develop gaps in knowledge about the software suite and trends in user issues. The support manager was the exception to this, but did not look at all the incoming tickets because of other duties. There was also always one lead on duty, so they tended to be the persons other consultants would ask questions. Ad-hoc conversations, such as the one shown in Figure 2, filled in gaps in knowledge of current issues, spreading such knowledge throughout the team.

> *Alex: Hey, look at [RT] 716424, did we have some kind of connection problem yesterday?*
> *Cameron: Oh yeah, there were some issues about that, but the devs fixed it and everything should be working by now.*
> *Alex: Yup, I wasn't getting any errors, so just checking.*

Figure 2. One consultant obtaining knowledge about an issue from another.

Experience appeared to be associated with the number of keywords one could recognize. This was most apparent when comparing the leads with newly hired consultants. Since leads typically worked more hours than consultants, others usually went to them with questions. On the other hand, newly hired consultants had large gaps in knowledge of known issues, and tended to be those who asked questions. New consultants spent their first two weeks reading through resolved RT tickets and observing senior consultants. They would spend the following two weeks mock-answering users' issues. Leads or the manager would review these before approving them to be sent to users.

The support team explicitly avoided requesting information from users. According to the manager, the team "*tries to minimize asking the user for more information because it really slows down the process waiting for someone to reply*." If a user did not provide a complete picture of their problem, the support team used other resources to narrow it down. Most cases, especially user-interface issues, were resolvable using the limited information provided by the user. However, in cases where there was a specific problem with a specific aspect of an application (e.g. a quiz not submitting for a particular user), leads and the support manager were able to log in as the user to try to replicate the problem.

While the staff relied heavily on keywords to infer missing context and detail, they explicitly did not use users' statements about personal consequences, pleas for help, or emotional distress. The staff indicated that they "*mostly ignore that information*," because "*it's not useful*" and "*does not help [...] replicate [the problem]*." Extremely emotional cases were forwarded to the support manger, who had more training in customer relations; however, he mentioned that being emotional did not help the user's case; the concern was more with whether the issue was known or new.

C. Escalating Issues via IRC

RT tickets that could not be resolved by providing help or workarounds were escalated by a lead or the support manager to an IRC channel shared by SDT staff. A median of three RT tickets per day were escalated to IRC, for a total of 333 during our six months of observations.

From the team's perspective, the purpose of escalation was for the support team to get the information they needed to close tickets. However, we observed several other functions that these brief discussions played. One was to

decide whether the issue was *known* or whether it was *new*; these discussions were rarely explicit, but were implied at the conclusion of the discussions. For example, here, a developer and designer are having a conversation about a recently discovered bug, but never explicitly state that it is new:

15:05 <taylor> her grid view is set to a specific filter group, but that filter group no longer exists!
15:05 <devin> huh […] interesting
15:05 <taylor> I will file a bug
15:06 <devin> indeed
15:06 <taylor> the code assumes that the filter groups will always exist, it doesnt bother to check
15:07 <taylor> I guess in this case the filter shold revert to 'all'
15:08 <devin> that seems right
15:08 <taylor > k
15:10 <taylor > alright, bug 11501 […]

We also found that these discussions transferred knowledge from the product team about what would be changing in the near future, and application functionality that had broken recently. In addition to the support team learning a great deal about the product team's recent work, they were also an opportunity for the support team to communicate trends in recurring issues such as usability problems.

Determining whether an escalated RT ticket was known or new ultimately determined whether it received attention by the product team. Known issues tended to be spotted and resolved quickly by others in IRC, often by targeting requests for knowledge to particular staff by preceding messages with the intended recipient's handle:

09:56 <drew> we might have a bug here [...]
09:56 <drew> is you look into the grade book for june titled *****
09:56 <drew> we can't get sections to show [...]
09:57 <hayden> drew: yup that is a known issue, already fixed in development

On the other hand, new issues required much more conversation and verification before they were escalated:

15:18 <alex> hayden: I'm with an instructor ***, who is attempting to sync a webQ quiz (***) to a Gradebook (***) assignment (***). However, when he attempts to import the WebQ, he receives the "application experienced an error ..." message. What's happening?
15:20 <hayden> Alex: i'm looking into it now, it appears the instructor may have found a bug
15:23 <hayden> Alex: the survey hasn't been deleted has it?
15:24 <hayden> actaully nevermind, that shouldn't matter
15:31 <alex> I noticed that the url says "survey", although the summary page says "About this quiz"
15:34 <hayden> Alex: the quiz is deleted, which should be ok, but i'm checking
15:37 <hayden> Alex: being deleted is what the problem is. it is a bug for sure, but that is why
15:39 <hayden> can you submit that as a bug for me?

Discussion about whether an issue was known or new in IRC was similar to that of the exchange between consultants. People would ask clarifying questions in IRC, sometimes directed at an individual, or groups of people. However, since this was a shared, asynchronous exchange, other people in the IRC channel (or those who joined later in the day) could see the entire day's conversations at any time and interject as needed. By having conversations in this medium, a wider set of people with different expertise and transactive memory could and did participate at any point in the conversation. Although it was not mandatory, we observed that the leads, managers, and all other staff on the product team logged into the IRC channel during their shifts. In analyzing the IRC logs, we found that *every* staff member contributed to conversations in IRC, though the amount of participation from each individual varied.

The product team also used the IRC medium to disseminate information to the support team. Oftentimes, this was used to teach a workaround or to report changing or broken features:

09:10 <jordon> everyone: old tools are down (simplesite, portfolio). Taylor is working on getting them up and running. Updates to come.
09:10 <alex> ok […] thanks for letting us know

IRC was also used by the support team to let the product team know of recurring issues that they had collected over time. These issues were captured in a Bugzilla report, and once there were more than two requests, a lead would bring it to the attention of the product team in IRC. In the following example, a lead starts a conversation by linking to a recurring issue and giving a brief description of the problem. The product team members evaluate it and subsequently take ownership of the issue:

10:17 <cameron> hayden: can you look at gradebook [url removed]
10:17 <cameron> it has an administrator *** that does not have the gradebook showing up their account
[14 lines of unrelated conversation removed]
10:49 <parker> Taylor is telling us its designed behavior
10:49 <parker> when you're an admin on both the cv and the tool
10:49 <devin> or a participant, i assume
10:50 <parker> he said that the CV hides the tools in it when its in the inbox. so the user was looking for a GB, but didn't see it
10:50 <riley> wow
10:50 <parker> anyway... cameron: what's the bug number on that?
10:51 <cameron> Bug 11182

Finally, IRC was also used to converse with people in close proximity (i.e. same office space):

12:56 <devin> devs: is there any chance that the fix for bug 9931 did not get merged onto trunk? i'm still seeing the bug

When asked why they would chat using IRC instead of talking face-to-face, a designer said, "*it's much more convenient to say it here [in IRC] because the other person might be busy at the moment, and they and other people will see the message*." As the designer mentioned, posting a question in IRC, even if it was directed towards a specific individual, had the benefit of everyone else in the channel having a chance to see it. Conversely, it also had the advantage of everyone being able to see the conversation and result, adding to everyone's knowledge. Ultimately, IRC was an opportunity for team members to not only access shared transactive memory, but synchronize theirs with the team.

D. Relating Issues as Bug Report Titles and Descriptions

When issues from technical support were escalated to bug reports, all the information from past representations was consolidated into bug report titles and descriptions. This act of consolidating issue content was particularly important since the product team might not work on the issue for several weeks, months, or years, making the word choice and phrasing critical for cueing the team's knowledge of the issue in relation to the software implementation.

To understand how knowledge of the issue was consolidated into bug report titles and summaries, we analyzed 29 RT tickets in the sample of 268 RT tickets discussed in the previous sections (due to authorization issues, however, we were unable to access two of these bug reports). We focused on how knowledge of the issue was consolidated into report titles, how the bug report titles were used, and what the team wrote in each report's description.

First we consider how the titles were derived from the users' e-mails. We compared each of the titles of the original RT ticket with their corresponding Bugzilla report's. While the original issues contained a variety of information (as

discussed earlier), including expected and observed behavior and other context, the titles almost exclusively indicated one of two types of information: (1) a noun phrase indicating some desired feature (e.g., "Copy-Paste to and from grading sheet column"), or (2) a phrase indicating some existing feature, with a characterization of some aspect of that feature implied to be undesirable (e.g., "Plain text editor treats return character as
"). The first phrasing focused on expected behavior and the context in which it was expected, while the second focused on the observed behavior, implying the expectation. The authors classified each report as one of these two independently, arriving at 93% agreement; the sample was half of each type. Little of the other information from user e-mails or IRC discussions appeared in the titles. We found that the 27 Bugzilla titles in our sample contained a median of 8 words; the corresponding user requests in RT were significantly longer, with a median of 89.5 words.

In our observations and interviews, we found that the report title and number were the two primary pieces of information the support team used to select issues for upcoming sprints. Having a good title reminded them about what the issue was, which application it concerned, and how complicated it would be relative to the time and resources available to address it. The support manager described the goal of title writing as "*making it easier […] to distinguish between [Bugzilla reports] and kind of know what it's about without having to open [it]*." We observed this same goal in sprint planning meetings: some reports (usually older or not created by the group members present) were opened to read the description before deciding whether or not to work on it. Other reports were not opened at all, but added to their to-do list, suggesting that they had a general idea about what the issue was, and approximately how long it would take to resolve. Interviews with the product team confirmed this:

"yeah, [I didn't have to open the report] because I was the one who put it in [to Bugzilla] [….] I think about … 3 …or 4 sprints [1-2 months] ago. Oh, and that other [report], Devin knew what it was […] and I think I know what it is, so we didn't bother opening it."

When asked why he opened some of the reports, he said,

"ah, it's because I don't think I was the one who put it in there [Bugzilla]. I remember talking about it… and I was pretty sure what it was … but I just wanted to make sure it was the one I was thinking of. […] it just takes […] 5 seconds to open up the [report] to check it out."

These observations show that the team used the report titles to quickly recall transactive memory of the issue and the kind of engineering and design work it would involve.

In addition to analyzing report titles, we also analyzed what information in Table 1 the team members left out from RT tickets when creating a Bugzilla report and what information was added from IRC discussions or elsewhere. Using our list of Bugzilla reports originating from RT tickets, we worked backwards tracking the origins of what was in the Bugzilla report. We compared each pair, checking to see if the report contained verbatim text from IRC or RT. We also checked to see if the report author elaborated on the issue with original text. Finally, we checked whether the author included text either confirming that they replicated the issue, or providing steps to reproduce it. There was no clear pattern in the combinations of these sources of information. The product team verified this, telling us that the descriptions were completely up to the person making them, and there was no standard practice or expected way to create them. There were, however, preferred sources. Authors elaborated on issues in 70% of the reports; 44% of reports used verbatim text from the original RT ticket in the description. Only one report contained text from IRC discussions, and 15% included reproduction steps.

E. Issues as Physical, Organizational Cues

Near the end of every 2-week sprint, the product team would meet to go through the list of Bugzilla reports and chose the ones they felt they should work on based on what they were currently working on, the age of a report, and the estimated time it would take to resolve. The team kept a physical record of the active issues on a whiteboard with post-it notes and index cards attached.

The post-it notes and index cards were similar in structure. Bugzilla reports were written by the individual who was going to work on it onto post-it notes with any combination of the report number, short description, and estimated time it would take to resolve in hours. Likewise, Bugzilla stories were written as they arose by the PM onto 3x5 inch index cards, always containing the report number, story identification number, verbatim description from the digital version, and the estimated number of hours or sprints it would take to resolve. Though these physical representations of user issues were created using data from Bugzilla reports, almost all the accumulated data up to this point is lost in this transition, stripped to the bare minimum. However, as the rest of this section will demonstrate, these meager representations of users' issues were enough of a cue for the product team member(s) to access their knowledge of the issue and work towards resolving it.

During the morning scrum meeting, everyone would go to the whiteboard, make any necessary changes to time estimates, and move their post-its to the next column, signifying that there was progress made. The whiteboard acted as a wall-sized to-do list and allowed individuals to track their own progress; it also enabled everyone on the team to see others' progress. As one designer stated, "*it's embarrassing if you have a bunch of… not finished tasks… when everyone else is clearly ahead of you and can see that*."

These meetings had a secondary function of maintaining, reinforcing, and updating the knowledge of members in the product team about active reports. Through their colleagues, teammates were exposed to more Bugzilla reports, adding to individuals' transactive memory about the work of their colleagues, enabling any of the staff to share this knowledge to the support team on IRC. In addition, everyone saw who was working on what, creating knowledge about who the expert for certain types of reports was, so that they could direct questions to that person about similar reports later on.

The physical representations of Bugzilla reports, the post-it notes, served as indexes into transactive memory. Post-its were created by the individual who was going to work on it, so it could have any combination of the report number, short description, and estimated number of hours it would take to resolve. Even though a lot of information was omitted when creating the physical representation of the Bugzilla report, it did not appear to matter at all for the

person working on it, or others on the team. They were simply an efficient way to remind everyone what was being worked on, without having to verbalize or explain the whole issue. Everyone we interviewed on the team thought of and referred to the functionality of the post-its as "bookmarks" to their memory and originating Bugzilla report. These results illustrated that the product team did not rely on the digital representations issues to do their work; they were cues for accessing and coordinating around transactive memory.

V. Discussion

Throughout our observations and analyses, we increasingly realized that the actual content in the representations were secondary; most of the information about user-reported issues was already in the team's collective knowledge. Users' e-mails occasionally provided new knowledge, but ultimately, the information conveyed by users was used primarily to cue knowledge already known by the team. Moreover, the content in subsequent representations of issues throughout the team's processes were secondary as well, serving primarily to cue the much richer knowledge of issues distributed throughout the team's collective memory. Finally, we found that many of the team's support and engineering processes, while primarily focused on accomplishing engineering work and making decisions, had the secondary effect of sharing and reinforcing the groups' transactive memory. Our finding showing SDT's ability to work effectively with incomplete information is consistent with prior works showing that transactive memory improves both information integration processes [7] and decision-making processes [18].

These findings reveal the importance of frequent communication between support staff and the rest of the product development team. The communication in IRC we observed appeared to be a critical part of ensuring not only that the support staff was able to help the users requesting assistance in a timely and detailed manner, but it was also an essential part of ensuring that the product team had a daily awareness of the types of new or recurring problems that users were encountering. Our findings also highlight the cost of support staff turnover, as the knowledge that support staff acquired over time was critical not only in providing effective user assistance, but it was an important part of ensuring that recurring user concerns that support staff saw daily could be escalated to the product team.

Our findings have several implications for issue tracking software like RT and Bugzilla, and other similar products. For example, our results suggest that the way issue content should be presented in these tools might focus on emphasizing the noun phrases in issue descriptions, to aid staff in identifying what a ticket concerns. Such tools might also consider visualizations that highlight what noun phrases are *current*, especially for products developed with Agile methods, to help teams identify recurring topics that might be overlooked as support staff change shifts.

As with any case study, our results should be interpreted with caution. The team we studied did compete with other products, but not financially; the team was also focused on serving a directly affiliated user community. Its products were cloud-based, web applications and these applications were evolved iteratively over two week sprint cycles. All of these characteristics of the team we studied may have contributed to our findings, meaning that the phenomena we observed might not occur in teams in different contexts.

References

[1] Ackerman, M.S. & Halverson, C. (1998). Considering an organization's memory. CSCW, 39-48.

[2] Aranda, J. & Venolia, G. (2009). The secret life of bugs: Going past the errors and omissions in software repositories. ICSE, 298-308.

[3] Bertram, D., Voida, A., Greenberg, S., & Walker, R. (2010). Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. CSCW, 291-300.

[4] Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008). What makes a good bug report? FSE, 308-318.

[5] Bowers, J. & Martin, D. (2000). Machinery in the new factories: interaction and technology in a bank's telephone call centre. CSCW, 49-58.

[6] Breu, S., Premraj, R., Sillito, J., & Zimmermann, T. (2010). Information needs in bug reports: improving cooperation between developers and users. CSCW, 301-310.

[7] Cannon-Bowers, J.A. & Salas, E. (2001). Reflections on shared cognition. Journal of Organizational Behavior 22 (2): 195–202.

[8] Chilana, P., Ko, A.J., & Wobbrock, J.O. (2010). Understanding expressions of unwanted behaviors in open bug reporting. VL/HCC, 203-206.

[9] Cunningham, S.J., Knowles, C., & Reeves, N. (2001). An ethnographic study of technical support workers: why we didn't build a tech support digital library. ACM/IEEE Joint Conf. on Digital Libraries, 189-198.

[10] Halverson, C.A., Erickson, T., & Ackerman, M.S. (2004). Behind the help desk: evolution of a knowledge management system in a large organization. CSCW, 304-313.

[11] Ko, A.J. & Chilana, P.K. (2010). How power users help and hinder open bug reporting. CHI, 1665-1674.

[12] Ko, A.J., DeLine, R., & Venolia, G. (2007). Information needs in collocated software development teams. ICSE, 344-353.

[13] Ko, A.J., Myers, B.A., & Chau, D.H. (2006). A linguistic analysis of how people describe software problems. VL/HCC, 127-134.

[14] Lutters, W.G. & Ackerman, M.S. (2007). Beyond boundary objects: collaborative reuse in aircraft technical support. JCSCW 16(3): 341-372.

[15] Rising, L. & Janoff, N.S. (2000). The scrum software development process for small teams. IEEE Software 17, 26-32.

[16] Schmidt, K. & Simone, C. (1996). Coordination mechanisms: towards a conceptual foundation of CSCW systems design. JCSCW, 5(2-3), 155-200.

[17] Schwaber, K. & Beedle, M. (2001). Agile software development with Scrum. Prentice Hall, 2001.

[18] Stasser, G., Stewart, D.D., & Wittenbaum, G.M. (1995). Expert roles and information exchange during discussion: The importance of knowing who knows what. J. of Experimental Social Psychology, 31 (3): 244–265.

[19] Wegner, D.M. (1986). Transactive memory: A contemporary analysis of the group mind. In B. Mullen & G. R. Goethals (Eds.), Theories of group behavior. New York: Springer-Verlag, 185-208.

[20] Yamauchi, Y., Whalen, J., & Bobrow, D.G. (2003). Information use of service technicians in difficult cases. CHI, 81-88.

[21] Zimmerman, T., Premraj, R., Sillito, J., & Breu, S. (2009). Improving bug tracking systems. ICSE Companion.