Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Software Engineering

Dane Bertram, Amy Voida, Saul Greenberg and Robert Walker
Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta, CANADA T2N 1N4
{dbertram, saul.greenberg, avoida, walker}@ucalgary.ca

ABSTRACT
Issue tracking systems help organizations manage issue reporting, assignment, tracking, resolution, and archiving. Traditionally, it is the Software Engineering community that researches issue tracking systems, where software defects are reported and tracked as 'bug reports' within an archival database. Yet issue tracking is fundamentally a social process and, as such, it is important to understand the design and use of issue tracking systems from that perspective. Consequently, we conducted a qualitative study of the use of issue tracking systems by small, collocated software development teams. We found that an issue tracker is not just a database for tracking bugs, features, and inquiries, but also a focal point for communication and coordination for many stakeholders within and beyond the software team. Customers, project managers, quality assurance personnel, and programmers all contribute to the shared knowledge and persistent communication that exists within the issue tracking system. We articulate various real-world practices surrounding issue trackers and offer design implications for future systems.

Author Keywords
Shared knowledge, issue tracking, software engineering

ACM Classification Keywords
H.5.3 [Information Interfaces and Presentation (e.g., HCI)]: Group and Organization Interfaces—CSCW; K.6.3 [Management of Computing and Information Systems]: Software Management—Software development.

INTRODUCTION

Commercial software development is by and large a group activity. Project managers create specifications, developers implement features, and quality assurance (QA) teams find and verify defects along with their associated fixes. Because of the large number and broad range of stakeholders involved in this form of group work, communication and collaboration become an integral part of the process.

Cite as:
Bertram, D., Voida, A., and Greenberg, S. (2009) Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Software Engineering. Report 2009-933-12, Department of Computer Science, University of Calgary, Calgary, AB, Canada T2N 1N4. June.

In addition to requiring the efforts of many, the creation of quality software also requires the careful management of many small, interrelated parts. Features and implementation tasks are borne out of specifications and defects or 'bugs' are inevitably discovered. These defects are often managed through the use of an *issue tracking system*. The issue tracker helps software teams manage issue reporting, assignment, tracking, resolution, and archiving via a reliable, shared to-do list that also serves as an archive of completed work [16].

A software product progresses from initial conception to version completion to ongoing maintenance work and then to preparation for subsequent releases. Issues arising during this lifecycle are typically manipulated by a number of stakeholders on the software team. Various statuses and other attributes can be applied to each issue to help distinguish among the phases of this lifecycle. Additionally, the issue's current ownership can be 'assigned' back and forth among project managers, developers, and quality assurance staff. All of a bug's many states, transitions, and annotations are tracked, archived, and made available to the team through the various interfaces provided by the issue tracker and its underlying database.

We discovered that many software teams rely on issue trackers to serve as both an "outboard brain" and implicit communication hub to help coordinate the intricate dance that comprises the production of any large software system. Thus, issue tracking is fundamentally a social process and, as such, it is important to understand the design and use of issue tracking systems from that perspective.

We conducted a qualitative study of the use of issue tracking systems by small, collocated software development teams. For each development team studied, we interviewed multiple stakeholders—each fulfilling a different role—in order to gain a more holistic understanding of the team's relationship with the issue tracker. From our results, we argue that the issue tracker for a software team serves as much more than a simple bug database. The issue tracker serves as a key knowledge repository, a communication and collaboration hub, and as a communication channel in and of itself. In addition, we identify and characterize several aspects of issue tracking about which its stakeholders hold differing and sometimes conflicting perspectives. Finally,

we provide implications for the design of issue tracking and software development coordination tools.

RELATED WORK

Just as strong leadership is required to guide a team toward success, so too are strong communication and collaboration tools essential in completing that journey. Much research has been done in the field of computer-supported cooperative work and software engineering to examine how software teams communicate with each other and coordinate their work. Perry et al., for example, studied individual developers' perceived and actual time allocation for various activities throughout their day-to-day work [14]. Their studies found that over half of each developer's time was spent interacting with co-workers. Ye argued that software development is both knowledge-intensive and collaborative [18]. Her work identified not only the technical axis of knowledge collaboration undertaken by software developers, but also its *social* axis. The deeply social element of software development has also been illustrated by LaToza, Venolia, and Deline [12]. In addition to noting developers' reliance on their social network for determining code rationale, their study also found that developers at Microsoft "reported spending nearly half their time fixing bugs."

In spite of all the analysis that has been done on software development in general, little if any of this attention has been directed toward the in-depth study of how issue or bug tracking systems fit into the role of facilitating communication and collaboration.

If we look at issue tracking systems specifically, we see a body of work that has primarily focused on improving the quality of bug reports [4], identifying who should work on a given issue [2], and improving developers' ability to detect defects in their systems [9]. In contrast, much of the broader software development literature mentions issue tracking only in passing; few have specifically brought its communication and collaboration aspects to the forefront.

Fortunately, the coordination component of specific development tools, such as configuration management systems, *have* been studied (e.g. [10]) and more recently Aranda and Venolia's work has started to bridge the gap between broader CSCW research and specific software engineering tools such as the bug tracker [3].

Aranda & Venolia set out to verify the seemingly rational assumption that bug tracking databases provide developers (and in turn, researchers) with an accurate—or at the very least sufficient—account of the history of the work items stored within them. They did so by tracing the complete lineage of 10 randomly selected, recently closed bugs. After exhausting all available electronic breadcrumbs, they proceeded to interview every person directly or indirectly involved in each bug's "life" until either a complete history was formed or the trail petered out. Through their intensive study they identified the following challenge:

> "If we want to understand and improve coordination dynamics we need our bug histories to include the social, political, and otherwise tacit information that is also part of the bread and butter of software development. This is often subtle, not always apparent, and it must be read between the lines of the evidence collected."

It was precisely these hidden coordination dynamics we set out to uncover in our study.

STUDY DESIGN AND MOTIVATION

Despite software engineers' best efforts to accurately model and manage the presence of bugs within the software development process, there is still an inescapably human component to the process. Therefore, we employed qualitative research methods to better understand the use of issue trackers within small software development teams.

Design

We conducted a qualitative study of collocated software development teams that make use of one or more formal issue tracking systems. We recruited 15 participants spread across 4 North American software teams. Each participant was asked to engage in two research activities:

1. **Questionnaire.** Participants were asked how long they had worked with the support of an issue tracking system, both continuously and including any gaps. They were also asked to identify their primary uses for the issue tracking software along with the frequency with which they used each of a collection of communication channels (instant messaging, email, micro-blogging, face-to-face, etc.). Basic demographic information such as sex and age were also collected.
2. **Semi-structured interview.** Participants were asked to describe various aspects of their issue tracking system and the processes and communication surrounding it. The interview protocol included questions pertaining to workflow, coordination, issue lifecycle, information seeking habits, and motivations for selecting specific communication channels. We also explored shortcomings and workarounds related to existing tools—both related to communications and those targeting general software development.

The interviews were carried out in each participant's regular workspace whenever possible. One team's participants each met with the interviewer at a common office due to logistical constraints. In this situation, a terminal remotely connected to the participant's regular workstation was provided to maintain as much software work-context as possible.

The results presented below were obtained through a grounded theory analysis of our interview data [6]. We generated concepts inductively from the data (e.g., "outboard brain" and "evidentiary system") and then refined and structured the concepts to arrive at the higher-level themes that make up the outline for much of this

Software Team	Participant Demographics						Issue Tracker Used
	Developer/Team Lead		Quality Assurance/Tester		Project/Program Manager		
	Male	Female	Male	Female	Male	Female	
Team A University IT Dept.	P1			P2			SharePoint & Excel
Team B Gaming Company	P6		P4, P5			P3	ExtraView & proprietary
Team C Issue Tracker Vendor	P8, P9, P11			P7	P10		FogBugz
Team D Large Display Vendor	P15	P13		P12	P14		FogBugz & Excel

Table 1. Overview of participant population

paper. We used selected questionnaire data to help characterize our participant population.

Participants

The participants in this study included 15 individuals from 4 different software development teams (Table 1). Each team belonged to a different organization and wrote software targeting a different market within the software development field. Each team made use of at least one formal issue tracking system in support of their day-to-day development tasks. Team A was part of a university's IT department whose job was maintaining and customizing a third party human resource management system. Team B was a product development team within a large gaming company. Team C wrote commercial issue tracking and project management software as well as a remote desktop application. Team D was a product development team writing software to accompany a large display hardware product.

Participants were recruited from three primary roles within the software team: developer/team lead, quality assurance/tester (QA), and project/program manager. Because of their constant, intimate interaction with the issue tracker, we focused, in particular, on the first two of these roles and the interplay between them. Participants' ages ranged from 25 to 51 years with a mean of 33.4. Our participants' experience with issue tracking software ranged from 2 to 16 years (including gaps) with a mean of 5.6 years. Gender balancing did not play a role in the recruiting process; however we were fortunate in having at least one female participant in each of the roles of interest.

WHAT IS AN ISSUE TRACKER?

In this research, we have identified a number of social dimensions to issue trackers that extend well beyond the conventional definitions found elsewhere in the literature.

Conventional definition

Most definitions approximate a defect or issue tracker as a software tool "that enables the [development] team to record and track the status of all outstanding issues associated with each configuration object" [15]. While this may be a perfectly functional surface reading, in reality, most issue trackers also track a variety of other artifacts including feature requests and inquiries.

An issue tracker serves as a centralized database for tracking bugs, features, and inquiries as they progress from their initial creation to a final closed state. Bugs outline defects in the software that need to be addressed by the development team and may be reported by customers, the QA team, or anyone else with sufficient access to the tracker. Features represent new functionality or enhancements to existing code and usually originate from project managers or team leads. Inquiries largely originate from customers and can include everything from sales questions to technical support.

As issues transition among various states (open, resolved, closed, etc.) various annotations are stored along with each issue. These annotations include attributes such as the title or due date for resolving a bug, as well as a history of ownership and discussion about the issue. Through our interviews and the analysis of their transcripts, our research has uncovered a number of other *social* dimensions of issue tracking systems.

Issue Tracker as a Knowledge Repository

In addition to recording a simple list of all the bugs, features, and customer inquiries worked on by a team, the issue tracker also stores a massive amount of organizational knowledge [1].

> We've had this thing for 8 years, starting from case 1, and now we're at 1,354,487 cases. All these little bits… little bits of insight and little bits of things are in there and you can search them. … It has been really valuable for us, and in the future it's just going to grow and grow and be more valuable (P08).

Over time, the issue tracker builds up a staggering amount of information concerning nagging customer support issues, partially fleshed out feature ideas, and a large portion of the communication surrounding these developments. This enormous pool of information is often useful for both the organization and the software team on a number of different levels [7].

Project managers in our study argued on behalf of customers and used the issue tracker's rich history to back up their arguments with concrete data by querying the issue tracker: "We've had 50 customers who have complained about this problem. It's time to stop punting this issue and actually fix it" (P10).

With the source control integration provided by many of our participants' issue trackers, programmers delved into previously resolved bug discussions to gain a better understanding of how a section of code came to be:

> Say I came across this code and I didn't understand why they implemented it this way. Then I would look up the [issue tracker] cases to see what they're trying to fix (P13).

More broadly speaking, this massive store of interrelated data is highly valued by those who use the issue tracker, even when they can't find the words to succinctly quantify it: "Having this archive is huge, in the bug tracker. It's one of the biggest…one of the primary reasons we use [our issue tracker]" (P09).

This ever-growing knowledge store is not entirely contained within the walls of the issue tracker, however. Instead, this knowledge is distributed across the issue tracker and those who use it both directly (such as programmers and testers) and indirectly (such as customers). It is in this way that the issue tracker plays a significant role in the *distributed cognition* of the software development team [11]. Each stakeholder in the issue tracking lifecycle contributes information and knowledge to the issue tracker via individual bugs and features. This knowledge is then distributed amongst those who add to these records as well as those who simply monitor issues and keep abreast of their progress.

We talked at length with project managers, developers, and those on QA teams about how they use their issue trackers. After doing so, we uncovered a multitude of other stakeholders involved in the process of entering, tracking, and resolving issues.

Issue Tracker as a Boundary Object

Although primarily aimed at software developers and quality assurance, the issue tracker's social network extends to include a wide range of roles and stakeholders within the broader software ecology. We found that project managers would use feature requests from customers to help formulate the direction of the software product; customer support leveraged the ability to create, view, and monitor bug status to fix customer problems and notify them of important updates and potential workarounds; and some teams even integrated the issue tracker with the sales team to track potential clients and resolve their inquires. Finally, and perhaps most importantly, our developer and quality assurance participants used the issue tracker on a day-to-day basis for prioritizing work and steadily improving the quality of their product.

Initially, it may seem that many of these stakeholders operate in relative isolation. After interviewing participants in project management, developer, and quality assurance roles, we found this was most definitely not the case. Issues were frequently assigned between project managers and developers to clarify priority and direction. Customer support assigned potential bugs to QA to confirm their existence and refine their reproduction steps (step-by-step instructions on how to recreate or 'repro' the problem). After this stage was complete, we found that QA would then either assign the bug directly to the developer who was likely responsible for that area of code or to the lead developer of the team for them to triage the bug's priority and determine who best to tackle it. It is the nuance between these roles and the flexibility of the underlying issue tracking system that has resulted in the tracker serving as not merely a repository of data, but also as a communication hub that fosters coordination among the entire product team and its related stakeholders.

Each of these stakeholders has a different set of needs they look to the issue tracker to satisfy. While each participant recognized a similar overarching goal of what the issue tracker was for, each had a slightly different set of expectations and uses for the underlying issues based on their role within the development process. The issue tracker, and in some ways its issues, served as *boundary objects* within this setting; that is, participants in different roles leveraged the data stored in the issue tracker in different ways and had differing views on what the underlying issues represented [17].

One example of the different ways in which stakeholders viewed the issue tracker was reflected in the way participants in different roles filtered and sorted cases within the issue tracker. Project managers often looked at high-level summaries of outstanding issues broken down by their priority level to get a feel for how close the release under construction might be to shipping. In contrast, team leads would break down outstanding tasks by the members of their team to ensure that no individual team member was over- or under-burdened. Finally, quality assurance regularly grouped open and resolved cases by project area or category to ensure that an even distribution of product coverage had been met through their testing or to focus their attention on under-tested areas of the product.

As well, the collection of people who interacted with the issue tracker wasn't strictly limited to those who fell inside the development team or even company boundary. The sales team, management higher up the organizational hierarchy, and contract workers (often testers) also periodically worked with the issue tracking system. Customers external to the organization would interact with the tracker, although usually in an indirect manner such as via automated crash reports or email integration (see below). This wide range of people working and influencing the contents of the issue tracking system begins to illustrate the broad communicative power it provides.

Issue Tracker as Communication & Coordination Hub

With the large role that issue trackers often play in the development process, it is not surprising that a significant body of communication surrounds the data stored in them: priority discussions during triage meetings, repro step clarifications between the person who created an issue and its current owner, fix verification and regression testing, etc. All these conversations are communicated across a number of different channels and frequently end up being stored along with the associated bug in one way or another. Tracking all this information can be challenging but proves

to be crucial in providing the team the necessary tools to coordinate their work.

> You're not going to be able to do it in your head or in files or in multiple emails that are being sent to all these different people…they're not connected. … If it's in this one central place that other people can see, maybe there's a chance for it to get fixed twice. Not just fixed once for the customer we fixed it for, but fixed for all the customers (P08).

The idea of the issue tracker serving as an easily accessible place to tie together all the various threads of information involved in software development came up again and again in our interview data. During triage meetings, the newly agreed-upon priority would be stored in the audit log of each bug under discussion. The assignment back and forth between a bug's filer and whoever is currently working on it was also logged when reproduction steps were being clarified. Similarly, the verification of resolved bugs and their transition into a closed or verified state was also recorded. Not only were all the state and field changes recorded, but other forms of communication were also frequently transposed into the issue tracker: copied-and-pasted portions of instant messenger conversations or email threads, summaries of face-to-face meetings, and even self-reflective notes and ideas for potential causes and solutions to problems.

The bug tracker also served our participants as both the starting and ending point for various links between development artifacts. Duplicate and related bugs often linked amongst each other. Developers working on a case would often ask for a co-worker's opinion or suggestions on how to proceed, and in the process they would often synchronize their view by sharing the bug id of the issue in question via instant messenger or email (P04, P05). Outbound links from the issue tracker tied individual feature cases to their respective specifications in wiki documents and links to shared document repositories also frequently originated within the issue tracker. Finally, countless reports were either generated directly from the issue tracker itself or leveraged the data stored within it.

Additionally, many of our participants' issue trackers also supported tight integration with email. Email addresses could be configured to serve as public-facing entry-points to the issue tracker. Whenever an email was sent to such an address, an issue would be created in the tracker and a pre-designated person would be notified of its existence. As one might expect, replies could be sent from the issue tracker to the original email sender and later replies would be appended to the issue via a bug id that was automatically inserted into the email subject line.

These many interconnected channels all funneled their way through the issue tracker in a variety of baroque ways. Part of the glue that held these paths together was the tracker's ability to serve as a communication channel, in and of itself.

Issue Tracker as Communication Channel

Each of our participants' issue trackers provided a means for them to communicate directly with others about an issue. Sometimes this capability was presented as a "comments" field that could be filled in whenever an issue was being assigned from one person to another and was appended onto the end of an existing list of comments. In other systems, there was a single "task discussions" field that was edited collectively and repeatedly by the various stakeholders in the bug (P01, P02). Both variations of this practice were often augmented by the ability to attach or associate email messages, screenshots, or other artifacts to the issue.

This comment history or issue discussion was frequently viewed as the single most valuable asset of using an issue tracking system:

> Comment history. So this is huge. This running dialog on the bug, this is just…this is necessary. Like, you can't work without this. … You can just see a history and figure out what the decision pattern was. … Just having this record is invaluable (P04).

The majority of our participants explained that by providing a consolidated view of an issue's entire history, it became possible to later review that history and gain the necessary context to understand the rationale behind decisions made and directions taken.

The communication channel provided by each issue's comment history was often favored over other channels, such as instant messaging, that were viewed to be more interrupting for less urgent and timely communication [13]. Whenever an issue was assigned to someone, the new assignee would be notified of the event via email. Most of the issue trackers used by our participants also supported the idea of "subscribing" to an issue and being notified, again via email, whenever anything about that issue changed. Due to most instant messaging systems' propensity to alert, flash, and otherwise grab their users' attention, this more subtle form of back-and-forth communication was often favored and in this way, the case itself served as a persistent, asynchronous, and oftentimes multicast communication channel.

Because the comment history was an integral part of each issue and remained attached and visible throughout the issue's entire lifecycle, it served as a form of *anchored conversation* [5] that kept relevant communication embedded within the context of the issue to which it referred:

> Ok, so in that specific case I would use [the issue tracker] for sure because then there's a record of the communication with the bug, which is where I want it anyway. Like, if I ask someone separately in a different way than I am just going to write it in the bug anyway (P07).

After interviewing our participants, we came to realize that each issue was treated much like a threaded chat room conversation in which the comment history formed the

body of the conversation and the issue itself provided the topic for each thread. Participants in the conversation came and went over time (either via being assigned the issue or by subscribing to its changes) and the persistence of the issue allowed for asynchronous communication, and collaboration among its stakeholders [8].

CONTRASTING PERSPECTIVES ON ISSUE TRACKING

As we have already seen, our participants' issue trackers have served a variety of roles above and beyond that of a simple bug database. And although the description of an issue tracker's high-level purpose may be fairly consistent across its primary stakeholders, once examined more closely, we identified a number of different and sometimes contrasting perspectives on issue tracking processes and the applicability of various features of issue tracking systems. We also found that these perspectives suggested design implications for future bug-tracking systems.

A Bug List, a Task List, or a To-do List?

As discussed earlier, issue trackers and the issues contained within them form boundary objects among the various stakeholders involved in the software development process; that is, "they have different meanings in different social worlds but their structure is common enough to more than one world to make them recognizable, a means of translation" [17]. Although the primary purpose of an issue tracker was described consistently across participants on the whole, when examined more closely, the perspectives of participants in differing roles revealed fine-grained distinctions in the function the issue tracker served.

Developers and members of the QA team expressed the need for an organized list of their work to-be-completed: "primarily when I log in on a day-to-day basis I look at the cases that are assigned to me and I use it as a to-do list" (P07). The issue tracker's ability to apply a sense of order to outstanding tasks was largely what allowed developers to "just come in and sit down and start working" (P09).

However, some participants expressed frustration with the issue tracker being overloaded for non–bug-tracking purposes. One of the lead members of a QA team articulated strong opinions about the specific types of issues that should and should not be stored in the tracker and how that reflected its primary purpose:

> [The issue tracker is there] to track problems. Not tasks, *problems*. *Totally* different. You can track tasks with Excel if you have to. It can be done. You don't want to track problems with Excel. Because a problem is not perception, it's reality, right? That's the difference (P04).

From the perception of this QA team, the issue tracker needed to be an infallible depiction of what problems were still outstanding and which had been addressed. Schedules of non-bug tasks could "show you your perception of how things will go, but in [the] issue tracker you want to know the reality of the situation" (P04).

Project managers and members of the QA team felt the issue tracker also served a major function as a vertical communication channel, cutting across colleagues at different levels of the organizational hierarchy; that is, it served as their "method of communicating to production and management of what the outstanding quality-impacting work left in the project [was]" (P05). In a similar vein, lead developers recognized the importance of integrating scheduling alongside the tracker's list of outstanding work:

> What I've found now is that we need something more than just bug tracking. We actually need a way to match that to the schedule as well and have some sort of time tracking built-in (P15).

While many of the opinions expressed by our participants focused on how the issue tracker was used from a personal, within-the-development-team perspective, one project manager also spoke from the customer's viewpoint: "I'm using it as an outboard brain to keep track of all of the things that, from a customer perspective, are important" (P10). From their standpoint, the issue tracker served not only as a database for tracking bugs and features, but also as an interface between the company and their customers.

These distinct viewpoints suggest that the issue tracking system serves different, yet interrelated purposes for each of the different roles within the development team. While the overarching purpose of the issue tracker remains clear, the finer details regarding how individuals with different roles use the system suggests the need for custom-tailored views for each of its various stakeholders.

Design implication #1: *The issue tracker represents different things to different people. These small but significant distinctions should be acknowledged and exposed in the issue tracker through features catering to each of the stakeholder's individual needs. Customizable, role-oriented interfaces that emphasize certain aspects of the tracker's data while abstracting away others may provide a better fit for the multitude of stakeholders that make up the issue tracking system's audience.*

Full-fledged Bugs and Almost Bugs

Ideally, our participants felt that every bug found in a system should find its way into the issue tracking system to serve as a record of its existence and eventual resolution. In realty however, the pragmatics of day-to-day development sometimes resulted in a less-than-complete record. Ephemeral, even if critical, bugs did not always get entered after the fact, and others failed to be created due to a fuzzy understanding of their root cause. We also found there were varying *degrees* of existence for bugs as they moved from being first reported, to confirmed, to completely fleshed-out entities within the issue tracking system.

Most participants commented indirectly about when an issue distinctly crossed the necessary threshold to qualify as a bug. This, at times blurry, line sometimes depended upon

the accuracy of its repro steps or the level of detail known about its underlying cause:

> In this case I knew that we were monitoring it, so I didn't create the issue for it right at that point. … As soon as I'd figured out what it was I logged the issue because I knew which program was causing the problem (P02).

The consensus among our participants was that the issue tracker should have as complete a record as possible of all the bugs in the system, however there were still certain circumstances under which bugs—sometimes important bugs—might be missed: "often bugs that are that critical don't even get filed because by the time that you would go to file it, the bug is already resolved" (P05).

To help avoid this problem, some of the participants' issue tracking systems integrated with their organization's public-facing email addresses: "any customer who sends email to [our company], it goes immediately into our issue tracking system. It automatically creates a case every single time" (P07). By automatically generating a case for each incoming email, more cases ended up in the tracker, and the consolidated view of bugs, features, and customer inquiries further contributed to the wealth and robustness of knowledge available to the development team.

***Design implication #2**: The starting point for a bug was not always a well-defined point in time and an issue's visibility to others on the team could place an onus of completeness on the filer that may prevent them from filing an issue as soon as possible. Providing functionality for stakeholders to enter semi-private, lightweight "pre-bugs" may increase the number of bugs that eventually end up being recorded.*

A "Flurry of Fields" vs. Ease of Entry

Certain members of the development team, such as QA, team leads, and management, often turned to the sorting and filtering capabilities of the issue tracker to gain insight into the state of its contents. Having a large collection of small, well-compartmentalized fields gave these stakeholders the ability to "slice and dice" the tracker's data in the ways they needed. Those who filed bugs, however, felt somewhat compelled to *not* file bugs due to the never-ending lists of fields to be completed and required fields on new issue entry forms. This tension turned out to be the number one complaint from our participants who used issue trackers with customizable fields. Balancing the number of fields associated with an issue and the ease of creating new cases was often an ongoing battle:

> Generally, everyone does feel that there's too many [laughs]. … So, it does get frustrating. And we've tried at different times to pare it down, but then it always explodes back out because someone feels they need to indicate something different about the issue and there's no other appropriate way to track that property (P01).

In contrast, those in QA saw "a value behind every single field because it [would help them] narrow a query down" in the future (P04). This contention primarily existed between highly technical personnel with frequent data-mining tasks, and the often less reporting-centric role of bug filers. Although QA was the predominant source of new bug reports, a large portion of cases in our participants' issue trackers also originated from customers and other potentially non-technical users of their software. These stakeholders would "tend to skip over a lot of these fields" (P02) because they were more interested in simply getting an issue recorded in the first place, and returning later to fill in the rest of the details.

This tension often resulted in an ever waxing and waning number of fields associated with cases in the issue tracker, as illustrated by P01 above. Some participants managed to find stopgaps that would give them the sorting and filtering capabilities they needed, without adding addition fields:

> So we try to, in the title, go through it…just because there's so many [bugs]…I don't know what the size is now, but there's 120 people logging bugs sometimes, so you just try to go through once you get the bug, try to put something in the front that helps you query for them and sort by the title (P03).

This makeshift tagging functionality was a wish reflected by others on the development team as well, and was used as an alternative to adding more fields, even if it was a less than ideal one: "so what we ended up doing is, look [referring to the title field]…I'm doing my own tags…'testpilot:' that's a tag" (P04).

Part of the desire for a tagging feature stemmed from the way tags could address the all-or-nothing problem of adding a custom field; because "unless *everyone's* using [the fields], they're not useful to anybody [and] that's the really big problem" (P04). By using tags (or an ad hoc facsimile), some of the benefits of having custom fields could still be attained. Additionally, makeshift tagging also supported the need for transient labels and searchable fields, such as tagging bugs that needed to be fixed for a rapidly approaching beta release. Such temporary labels served as useful search terms during the limited time span leading up to the associated deadline.

The need to minimize the number of fields presented to issue filers was well recognized and even resulted in Team B going so far as to write their own custom abstraction that sat on top of their issue tracker for creating bugs:

> Because you kind of have to be an expert with the bug tracking system to file good bugs with this system, we wrote a very simple interface to encourage non-experts to file bugs. … We found that when we put this in, people in other departments who have never filed a bug before ever, were filing bugs (P05).

This sentiment was also reflected in Team C, the issue tracking software vendor, by not having a single required field when filing a new bug report. Getting bugs into the issue tracker in the first place, even if incomplete, was a primary concern for many on the QA team: "It's to get information disseminated as quickly as possible and then

you can go back to the person to get more information if you need to" (P07).

The tension between having a rich set of compartmentalized data attributes associated with each bug and the ease with which they could be created was a common one.

Design implication #3: *Providing lightweight tagging capabilities to issue tracking would likely help ease some of the tensions between wanting detailed fields associated with issues, and maintaining a simplified, hassle-free interface for creating cases. It would be important for these tags to leverage the issue tracker's existing robust infrastructure for searching, sorting, and filtering issues.*

Different Perceptions of Priority

One of the cardinal services provided by the issue tracker, as described by our participants, was its ability to prioritize and order their team's outstanding work. This prioritization allowed developers and those on the QA team to organize their personal work and maintain an accurate idea of which issues should be addressed first with the limited time available for any given milestone. These categories or levels of priority also served as a gauge of where the project stood in terms of both completeness and relative quality. Having many high-priority or "showstopper" bugs open presented a very different potential timeline to management than having only lower priority bugs left to resolve.

We found that the priority assigned to an issue could come from a number of different sources depending on the development team's workflow and authority structure. Sometimes a bug's priority initially came from the filer of the bug or feature request. Often, this priority would be redefined at a later time by project managers, team leads, or individual developers later in the issue's lifecycle.

For example, Team A worked directly in support of a separate department on campus and, as such, that department's manager had a well-defined influence over the prioritization of projects for the development team:

> It's up to them to decide, you know, "Hey, we've got 2 projects of seemingly equal priority, what do you want us to do?" And they'll prioritize "Ok, that one's higher" or "That'll solve more of our problems" (P01).

Even though this external manager had the ability to specify the priority of certain projects, the development team retained its ability to govern its own projects depending on their scope and size. Those projects requiring less than "one man-week" to be completed did not require approval from outside the development team. This hierarchical prioritization structure also trickled down to influence individual developers' perception of prioritization factors: "developers set the priority, but then, you know, there are certain ones that are probably more important to the product owner than other ones" (P13).

When examining a single developer's list of issues, our participants expressed the need for finer granularity when prioritizing their work:

> We have a pretty simple priority system in [our issue tracker]. There's low, medium, high, or blocker, but when you have 3 high bugs "Uh, which one's the first one?" (P06).

To work around this, some of our participants repurposed other features of their issue tracker to address the lack of fine-grained control over the ordering of their bugs: "this [favorites] menu has a sense of order to it, so I use it as a task list...especially if I'm afraid I'm going to forget to work on something" (P11).

Finally, the "ripple effect" of changing code in the project also played a significant role in the prioritization of bugs for some of our participants, such as the project manager for Team B:

> If we're going to fix this, is this a low-risk, meaning we can fix this, it's not going to impact anybody else, or is this a high-risk, meaning that if we fix this one model potentially a cut scene could break? (P03).

Although encapsulated within a single "priority" field, often the true priority of an issue proved to be multi-layered. Project managers, quality assurance, and developers all often played a role in determining the priority for a given issue and sometimes these priorities conflicted or left room for personal discretion.

Design implication #4*: Acknowledging the multitude of factors that compose an issue's priority is important to serving the needs of its stakeholders. It may prove useful to present users of the issue tracker with a personal, persistent, and easily re-orderable list that is separated from those strictly ordered based on issue fields such as priority. Giving users the means to articulate their personal ordering of issues without explicitly affecting its priority may help them to better organize their day-to-day work.*

Shades of Ownership: Yours, Mine, or Ours?

During our study, issue ownership was repeatedly reported as being an essential component of the software development process. Having the ability to assign an issue to a single person via the issue tracker—and having a record of that assignment—provided the accountability needed by the team in order to complete their work effectively. However, this ownership of an issue was not always a clear case of "mine vs. yours" throughout its lifetime.

Most teams in our study had a workflow that reflected the idea of having a single, high-level owner for each bug, feature, or inquiry. In the case of bugs, this ownership would often start and end with whoever initially filed the issue: "When I log an issue I close it because it's really for me to take it from top to bottom" (P02). This full-circle approach to issue ownership was supported, sometimes indirectly, by all of the issue trackers examined. The process was usually facilitated by a workflow where a case (once it was resolved) would be automatically re-assigned to the person that originally opened it. When not enforced directly by the issue tracker's workflow rules, this behavior

was instilled in the development team via their standard practices and procedures over time.

In the issue trackers we examined, an issue could generally only be assigned to a single person (or sometimes a placeholder representing a group) at any given time. At first glance, this may appear to be a limitation, but in reality this was the main reason accountability was preserved:

> [Issues are] always assigned to an individual—someone's always responsible for them. None of this diffusion of responsibility stuff. So at any point your boss can come and say "You're responsible for this. What's going on?" (P08).

The restriction of having only a single active owner at any point in time did result in some participants monitoring bugs even when they were no longer assigned to them personally: "In this case, [the bug would] still be active and it wasn't assigned to me, but I was monitoring it because it was a showstopper…like it was a really big deal" (P12). This monitoring behavior was also sometimes supported via the issue subscription and email notification functionality described earlier. Other times, monitoring would take the form of flagging or bookmarking the issue by its original owner before assigning it to someone else to work on: "I start by starring the bug so that I remember that I still kind of own it and then assigning it to them" (P11).

A related ownership issue was brought up by a member of Team C, who described the distinction between the ownership of a customer support inquiry and its underlying bug or feature: "by design we try to separate the inquiry and ultimately the actual bug case" (P10). This behavior allowed for a relatively clean separation between the technical discussion around a fix and the potential workaround instructions and public-facing discussion with the customer. Hyperlinking between these cases was used in an attempt to maintain a unified view of both aspects of the issue. Using the linking functionality and/or the other methods mentioned previously, the contact person responding to the customer might also monitor the underlying technical bug in order to notify the customer of its status and progress over time.

Design implication #5*: Multiple levels of issue ownership exist throughout the software development team. By supporting this pattern in the issue tracker itself, stakeholders can eliminate their need to manually monitor issues that they still partially own.*

Design implication #6*: Allowing for distinct facets of an issue (e.g., customer communication and its technical discussion) to be contained within a single entity while remaining easily distinguishable may also prove useful in acknowledging the multiple identities of certain issues.*

Privacy, Transparency, and the 'In' Crowd

Just as there is a "vital tension between privacy and visibility" in supporting general communication [8], a similar tension also exists with respect to the contents of the issue tracker. Having stakeholders both inside and outside the company wall created tension in whether issues should be accessible by various audiences and which portions of those issues should be visible. Although developers and project managers alike agreed that "you want your customers to feel like they're part of forming the features [of your product]" (P11), in some cases, this desire conflicted with the inherent sensitivity of much of the information being stored within the issue tracking system.

Getting customer feedback on which features to implement and keeping them in the loop as bugs were being resolved were both important aspects of the organization-customer relationship. Exposing appropriate information to customers while protecting proprietary or vulnerability-related details was a difficult line to walk: "a lot of that stuff is not accessible in [our issue tracker] and would take a lot of work to make it accessible without exposing private information" (P09).

This dichotomy between "insider" and "outsider" views of the issue tracker was a palpable source of frustration to those on the development team who frequently interacted with customers:

> The benefit of [our issue tracker] is that it's really rich and you can see the entire issue history and a whole bunch of other stuff. But the downside is that it's a fairly closed system in terms of there's a clear sense of who's in and who's out, and customers are sort of out. Even though we can collect information from them, they can't interact with the system directly (P10).

An issue's representation as seen by those internal to the company was often accompanied by a public-facing counterpart in a manner much like the customer communication and technical discussion facets described in the previous section. Unfortunately, in commercial development teams, striking the necessary balance between customer transparency and the security of sensitive information often resulted in an entirely closed system with no public presence outside of a bug reporting interface.

Design implication #7*: Careful consideration of the multiple potential audiences for an issue and its attributes is necessary when developing external-facing interfaces and views. Graduation between fully-private and fully-public data may be needed and a clear distinction among these levels of privacy should be maintained in order to gain transparency without risking undue exposure of sensitive information.*

DISCUSSION & CONCLUSION

One of our participants, P14, had been working with issue trackers of various sorts for over 28 years:

> My first 8 were actually a paper tracking system. A call would come in from a customer and the call would be time-stamped, it would indicate on the call what the problem was and it would go into some box—physically some little box—and then you'd filter through and look for your name and pick it up if it was assigned to you (P14).

Issue trackers have come a long way from this initial physical form, but at their core, they still remain primarily focused on simply tracking and archiving issues. What has changed over time however—in addition to the transition from physical to digital media—is the increasing use of issue tracking systems as an essential form of communication within the development team.

Issues still frequently come in from outside customers, but instead of taking the form of a phone call that is transcribed onto a physical card, new issues may now be automatic crash reports submitted by the customer's software, online entry forms completed by customers, or generated from incoming email. Instead of assigning bugs to one another and indicating their status by placing them into physical boxes, these operations are now done electronically.

As issue trackers have evolved to become more central to the software development process, they have also begun to service many of the conversational, archival, and organizational needs of that process. Addressing the vastly varying needs of the software team has not come without its challenges, however. Supporting customer inquiries while maintaining the highly technical conversation surrounding its resolution has sometimes resulted in a fragmented representation of what is essentially a single issue. The issue itself now ties together a wide variety of resources and is leveraged by an increasing number of stakeholders in varying ways. In order to support these stakeholders, properly faceted views of this wealth of underlying data within the issue tracker need to be presented. These may take the form of per-developer to-do lists, filtered public-facing views, or lists that distinguish between the perception of future work and the reality of identified bugs.

From a CSCW perspective, issue tracking systems now play a significant supporting role in the communication and collaboration within software development teams. In addition to providing developers with prioritized to-do lists, issue trackers have come to embody a massive amount of organizational knowledge.

To conclude, we have presented results from a qualitative study of small, collocated software development teams and have responded to Aranda and Venolia's call to uncover the "hidden coordination dynamics" surrounding issue tracking systems [3]. Our contributions include:

- Articulating the many roles of the issue tracker (knowledge repository, boundary object, coordination hub, and communication channel);
- Describing the many areas where stakeholder perceptions vary (types of lists, degrees of issue existence, field congestion, priority perceptions, shades of ownership, and the tension between transparency and privacy);
- Identifying seven implications for the design of issue tracking and software development coordination tools.

ACKNOWLEDGEMENTS

We would like to thank the software companies and their employees that participated in our study for their time. We would also like to thank Stephen Voida for his input and feedback on early drafts of this paper. This research was supported in part by the National Sciences and Engineering Research Council of Canada, the Informatics Circle of Research Excellence, Alberta Ingenuity, the NSERC/iCORE/Smart Technologies Chair in Interactive Technologies, and by NSERC's NECTAR Strategic Networks Grants Program.

REFERENCES

1. Ackerman, M.S. and Halverson, C. Considering an organization's memory. In *Proc. CSCW 1998*, ACM Press (1998), 39–48.
2. Anvik, J., Hiew, L., and Murphy, G.C. Who should fix this bug? In *Proc. ICSE 2006*, ACM Press (2006), 361–370.
3. Aranda, J. & Venolia, G. The secret life of bugs: Going past the errors and omissions in software repositories. In *Proc. ICSE 2009*, ACM Press (2009).
4. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., and Zimmermann, T. What makes a good bug report? In *Proc. FSE 2008*, ACM Press (2008), 308–318.
5. Churchill, E.F., Trevor, J., Bly, S., Nelson, L., and Cubranic, D. Anchored conversations: Chatting in the context of a document, In *Proc. CHI 2000*, ACM Press (2000), 454–461.
6. Corbin, J. and Strauss, A. *Basics of qualitative research: Techniques and procedures for developing grounded theory* (3rd Ed.). Sage Publications, Los Angeles, 2008.
7. Dingsøyr, T. and Røyrvik, E. An empirical study of an informal knowledge repository in a medium-sized software consulting company. In *Proc. ICSE 2003*, IEEE Computer Society (2003), 84–92.
8. Erickson, T., Smith, D.N., Kellogg, W.A., Laff, M., Richards, J.T., and Bradner, E. Socially translucent conversations: Social proxies, persistent conversation, and the design of "Babble." In *Proc. CHI 1999*, ACM Press (1999), 72–79.
9. Fenton, N. and Neil, M. A critique of software defect prediction models. *IEEE Transactions on Software Engineering, 25*, 5 (1999), 675–689.
10. Grinter, R. E. Using a configuration management tool to coordinate software development. In *Proc. COCS 1995*, ACM Press (1995), 168–177.
11. Hollan, J., Hutchins, E., and Kirsh, D. Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM TOCHI, 7*(2), (2000), 174–196.
12. LaToza, T.D., Venolia, G., and DeLine, R. Maintaining mental models: A study of developer work habits. In *Proc. ICSE 2006*, ACM Press (2006), 492–501.
13. Nardi, B.A., Whittaker, S., and Bradner, E. Interaction and outeraction: Instant messaging in action. In *Proc. CSCW 2000*, ACM Press (2000), 79–88.
14. Perry, D.E., Staudenmayer, N., and Votta, L.G. People, organizations, and process improvement. *IEEE Software, 11*, 4 (1994), 36–45.

15. Pressman, R.S., *Software engineering: A practitioner's approach* (6th Ed.), New York, McGraw-Hill, 2004.
16. Reis, C.R. and de Mattos Fortes, R.P. An overview of the software engineering process and tools in the Mozilla project. In *Proc. Workshop Open Source Software Development*, University of Newcastle upon Tyne (2002), 155–175.
17. Star, S.L. and Griesemer, J.R. Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science 19*, 3 (1989), 387–420.
18. Ye, Y. Supporting software development as knowledge-intensive and collaborative activity. In *Proc. WISER 2006*, ACM Press (2006), 15–22.