UNIVERSITY OF CALGARY

The Social Nature of Issue Tracking in Software Engineering

by

Dane Bertram

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA

DECEMBER, 2009

UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled "The Social Nature of Issue Tracking in Software Engineering" submitted by Dane Bertram in partial fulfillment of the requirements for the degree of Master of Science.

Supervisor, Dr. Saul Greenberg
Department of Computer Science

Co-supervisor, Dr. Robert Walker
Department of Computer Science

Dr. Jonathan Sillito
Department of Computer Science

Dr. Yaoping Hu
Department of Electrical & Computer Engineering

Date

Abstract

Issue tracking systems help organizations manage issue reporting, assignment, tracking, resolution, and archiving. Traditionally, issue tracking systems have been largely viewed as simple data stores where software defects are reported and tracked as “bug reports” within an archival database. Yet, as issue tracking is fundamentally a social process, it is important to understand the design and use of issue tracking systems from that perspective. Consequently, I (and my colleagues) conducted a qualitative study of issue tracking systems as used 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 immediate 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. These results were all the more striking because in spite of the teams being collocated—and the frequent, face-to-face communication afforded by such collocation—the issue tracker was *still* used as a fundamental communication channel. Through my analysis of the interviews conducted during the study, I identified five roles that issue tracking systems play within such teams, six areas where stakeholder perceptions of various aspects of the issue tracker varied, and—as a direct result of those differing perceptions—seven considerations for the design of future issue tracking and software development coordination tools. I also presented a set of three interface design sketches to a subset of my research participants and solicited their feedback.

Publications

Materials, ideas, and tables in this thesis have previously appeared in the following publication (see Appendix B for co-author permission forms):

Bertram, D., Voida, A., Greenberg, S., and Walker, R. (2010). **Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams**. To appear in Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work (Savannah, GA, February 6 - 10, 2010). CSCW 2010. ACM, New York, NY.

Acknowledgements

I would like to thank my supervisors Dr. Saul Greenberg and Dr. Robert Walker for their guidance, patience, and encouragement. They allowed me the freedom to pursue my interests as a professional developer while also providing me with the necessary resources to translate them into this thesis. I would also like to thank Dr. Amy Voida for her guidance and mentoring throughout the design, execution, analysis, and writing-up of my research study. Without her help, that study—and as a result, this thesis—would simply not exist. I would also like to thank the software companies and their employees that participated in my research study for their time. 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.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

In this thesis, I address the use of issue (or defect, or modification request) tracking systems in small, collocated, software development teams. Specifically, I: investigate how these systems fundamentally support the communicative and collaborative aspects of software development; articulate the various roles played by these systems within such small, collocated teams; and identify how these systems might be redesigned to better support those identified roles in the future. To set the scene, I begin this chapter by motivating my interest in the area and providing a brief overview of existing research that further clarifies the niche addressed by this thesis. I then present the specific problem addressed by my research and how I will solve it in this thesis. I conclude with an organizational overview of the remainder of this document.

1.1 Background & Motivation

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, even within small, collocated teams.

In addition to requiring the efforts of many, the creation of quality software also requires the careful management of many small, interrelated issues relating to the software, where these issues appear and are (usually) resolved over the course of time. While features and implementation tasks are borne out of specifications, they rarely result in perfect software. Defects or “bugs” are inevitably found. 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 is used by numerous stakeholders throughout the lifecycle of the software and also serves as an archive of completed work [Reis and de Mattos Fortes 2002].

Existing research examining the software development process in general has repeatedly identified a deeply rooted social component. For example, LaToza et al. [2006] found developers relied heavily on their social network for determining the rationale behind a given piece of code, while Perry et al. [1994] found developers spent *over half of their time* interacting with co-workers during day-to-day development tasks. Given how frequently the social component of software development has been noted in the literature, it suggests that the "tools of the trade" should attempt to directly support this social aspect of creating software. Although not as well explored, the same argument applies to issue trackers. A small body of work has examined the use of issue tracking systems within distributed software development teams and has noted their use as a general purpose communication and collaboration device [Carstensen et al. 1995, Halverson et al. 2006, Sandusky and Gasser 2005].

However, none of this work has specifically examined *small, collocated* teams. This is the niche explored by this thesis, i.e., how small, collocated teams use issue tracking systems. This is important, for small development teams are still commonplace within many small organizations today. As we shall see, despite the lower barrier to face-to-face communication associated with these collocated teams, our study—described in detail throughout the remainder of this thesis—uncovered that the issue tracker is still fundamental in meeting these teams' communication, collaboration, and coordination needs.

To foreshadow what is to come, I will 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. I go on to identify and characterize several aspects of issue tracking about which its many stakeholders hold differing and sometimes conflicting perspectives. Finally, I provide considerations for the design of issue tracking and software development coordination tools. While some of the considerations presented in later chapters may already be addressed by existing commercial tools, it is the interplay and tradeoffs that must be made among them that I argue also need to be carefully explored and scrutinized.

1.2 Problem Statement

This thesis concerns the real-world usage of issue tracking systems in small, collocated development teams and how these tracking systems fulfill the communicative, collaborative, and coordinative needs of such teams. I address the following problem in this thesis:

> **We do not know how small, collocated teams make use of issue tracking systems in real-world, industrial settings from the perspective of supporting communication, collaboration, and coordination both inside and outside the team.** Previous research (Sandusky and Gasser 2005, Halverson et al. 2006, Carstensen et al. 1995, etc.) has examined or noted these aspects of issue tracking systems, but *only* within the context of *large, distributed* teams. Examining how issue tracking systems support these core components of the software development process within small, collocated teams *in spite* of their intrinsic benefits of frequent, low-cost face-to-face communication bears exploration.

1.3 Thesis Goals

In this thesis, I address the aforementioned problem with the following goals:

1. **I investigate how small, collocated teams use issue tracking systems in day-to-day software development tasks for the purpose of communication, collaboration, and coordination both inside and outside the immediate development team.** I have designed, conducted, and evaluated the results of a qualitative study aimed at exploring how issue tracking systems are currently used in the above mentioned teams with a specific focus on how these tools fit into the larger role of coordinating the development process, facilitating

cooperation within the development team, and supporting the necessary communication among its members. Through this exploration I have identified five key roles fulfilled by the issue tracking system within these teams. Six areas in which stakeholder perceptions of the issue tracker varied were also identified and used as the direct basis for the development of seven design considerations.

2. **I attempt to design new interfaces as well as enhancements to existing systems with the intent of incorporating better communication and awareness facilities into the issue tracking and overall software development processes.** To achieve this goal, I presented three interface design sketches to real-world development teams and solicited their feedback. These designs do not attempt to remedy all the limitations of current issue tracking systems nor directly address the design considerations identified via the previous thesis goal. Rather, these designs illustrate possible enhancements to the communication tools used by developers so that they can better interoperate with issue tracking systems, as well as enhancements to the issue trackers themselves to better support and make use of the development team's communication surrounding issues.

1.4 Organizational Overview

This thesis is divided into seven chapters:

In Chapter 2, I contextualize the area examined in this thesis by providing background information explaining what comprises an issue tracking system as well as some of the usual characteristics of these systems. Next, I present a deeper discussion of the existing literature as it pertains to the social aspects of software development and coordination within the development team. This section also describes previous work that has examined issue tracking systems specifically, and ends by describing the overlapping area where issue tracking and software development coordination meet. This discussion helps elucidate the specific niche addressed by the remainder of this thesis: the study of the communicative and collaborative aspects of issue tracking systems within small, collocated development teams.

In Chapter 3, I describe the qualitative empirical study conducted as my primary means of attaining the first goal set out previously in Section 1.3. This description begins with the original high-level questions I set out to answer through my study and carries through their evolution into the full-fledged qualitative study that was conducted with 15 participants spread across 4 different organizations. Details of my sampling method, participant population stratification, development team profiles, participant demographics, study materials and limitations, as well as the data collection process and analysis methodology are also presented.

In Chapter 4, I present the results of my study pertaining to the many different communication and collaboration roles taken on by the issue tracking system within the small, collocated, industrial software teams interviewed. I begin by revisiting conventional definitions of issue tracking systems. I then go on to articulate how these systems are simultaneously seen as a knowledge repository, a boundary object, a communication and coordination hub, a communication channel in and of itself, and as a contextualization repository.

In Chapter 5, I describe the differing views that team members had of various issue tracking facilities, as well as shortcomings and other incompletely met needs of the software as relayed by study participants. The areas examined include the differences in how the issue tracker itself was viewed by its clients (e.g., as a bug list, task list, or to-do list); when an issue comes into being and when this should be reflected in the issue tracker; the contention between many well-compartmentalized fields and the ease with which new cases can be created; the different perceptions and origins of an issue’s priority; the various shades of issue ownership and their related embodiments; and finally the tricky problem of balancing the privacy of sensitive issue information with its customer-facing transparency. Based on the frustrations expressed by our participants, I then formulate seven design considerations, summarized in Table 5.1.

In Chapter 6, I attempt to translate some of the aforementioned design considerations into actual designs. This is done via interface design sketches that stress the incorporation of communication and awareness within issue tracking. In particular, they illustrate possible enhancements to communication tools so that they can better interoperate with issue trackers; and to the issue tracking systems themselves so that they can better incorporate the communication surrounding the issues they contain. These designs were presented to a subset of our research participants with whom I discussed the designs' envisaged functionality and solicited their feedback.

In Chapter 7, I review the primary conclusions drawn from thesis as a whole, and then summarize the primary and secondary contributions of this research. I then briefly present potential future directions suggested by this research.

Chapter 2. Background & Related Work

This chapter sets the scene for the rest of this thesis and articulates how my thesis fits into the larger research space. I present background information on issue tracking systems and their associated workflows. This is followed by: a discussion of related work covering the social aspects of software development and development team coordination that have previously been explored; work examining issue tracking systems specifically; and a closing discussion of where issue tracking systems and development coordination meet. Through the exploration of this existing literature, this chapter will introduce the niche addressed by the remainder of this thesis: the examination of the communicative and collaborative aspects of issue tracking systems within small, collocated development teams.

2.1 Issue Tracking 101

A bug-tracking database is not just a memory aid or a scheduling tool. It doesn't make it ***easier*** *to produce great software, it makes it* ***possible*** *to create great software.*

—Joel Spolsky, as quoted in the forward of Gunderloy [2007]

2.1.1 What is issue tracking?

There is a plethora of issue tracking software in existence today. While all of these systems vary considerably in their interfaces and features, they all share the same core purposes and functionality as described below.

If we look to textbooks for a definition of issue tracking, we find descriptions such as the following:

Issue tracking, often called bug tracking (and sometimes request tracking), is the process of keeping track of your open development issues. Bug tracking is a misleading term in many ways and obviously depends on your *definition of bug. "Issue" is a broad enough term to describe most of the kinds of tasks you might need to track when developing* [*software*]*, and so drives our choice of terminology here.*

—Henderson [2006]

And if we look further for a more technical description, we find definitions such as this:

A bug tracking system is some program or application that allows the project team to report, manage, and analyze bug reports and bug trends. Functionally, most bug tracking systems provide a form that allows us to report and manage specific bugs, a set of stored reports and graphs that allow us to analyze, manipulate, and output this bug data in various ways, and a customizable workflow or life cycle that provides for orderly bug management.

—Black [2002]

Issue trackers, bug trackers, modification request control systems—regardless of what specific terminology is used—all roughly refer to the same class of software systems. In the most general of terms, these systems are designed to manage electronic artefacts that: (a) move from a starting state to an end state (and possibly visit several in-between states along the way), and (b) accumulate information during their transitions between these states. More specifically, *issue tracking systems* are databases that keep track of bodies of information—outstanding issues—such as software defects or "bugs," feature requests, customer inquiries, etc. The variety in the types of information stored in issue tracking systems has been acknowledged since their earliest incarnations. The following definition by Knudsen et al. still applies over 30 years after it was coined.

A Modification Request (MR) is a request to those in charge of a system to modify that system. The request might be to add, modify, or delete capabilities, to fix a bug, to issue a new version of a system, or even to create a system. Thus our definition of an MR includes trouble reports, design change requests, enhancement requests, etc. A request may result in changes to system hardware, software, or documentation or it may result in no changes at all.

—Knudsen et al. [1976]

All this information represents tasks that move from an opening state to a closed state over time. Along the way, these information artefacts often accumulate additional information as various people within the software team work on them. For example, a bug might gain instructions on how to reproduce the problem, or a feature request might be refined with additional specifications or sketches. Each of these items (or "modification requests" or "issues") can also have certain attributes associated with them. A bug, for example, might include things like the time the issue was filed, who filed it, what version of the software the bug applied to, and the severity of the bug (e.g., a *critical* bug that causes loss of data versus a *minor* one such as an aesthetic imperfection).

My primary interest lies in issue tracking systems as applied to software development. Given the relatively free and unconstrained usage of the term "issue tracking," there are also similar systems that I will *not* be examining in this thesis—in particular, helpdesk trouble ticket trackers, and business issue management systems. While these systems *do* represent a common manifestation of "issue tracking," their primary purpose is not targeted toward software development and are thus outside my focus.

2.1.2 The lifecycle of an issue

Within an issue tracking system, each item (or issue or case) generally follows one of a few predefined paths from when it is first opened until it is eventually closed. Depending on the type of issue, there may be various intermediate states as well as cycles within its path.

> *Each stage in the life of an* [*issue*] *is characterized by a status, or state. The number of states may vary from system to system. In addition, within a single* [*issue tracking*] *system, different* [*issues*] *will go through a different succession of states.*
>
> —Knudsen et al. [1976]

This set of paths is often referred to as the "workflow" supported by an issue tracking system [Bugzilla 2009, Atlassian 2009]. Workflows can vary from very simple and open-ended to very detailed and complex. The state diagram in Figure 2.1 illustrates a minimal set of states required for an issue tracking system's workflow, namely, the "open" and "closed" states and the transitions between them. When a new bug or feature request is created it starts in the “open” state. After work has been completed on this item it follows the “resolve” transition to end up in the “closed” state. If the issue is later found to be incomplete (e.g., a bug reappears or a feature is not fully implemented) it can follow the “reopen” transition back to the “open” state.

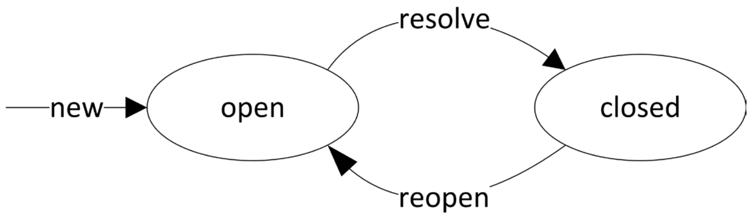


Figure 2.1 – Simplified issue lifecycle

As eloquent and simple as this set of states is, there are a number of situations in which such a simplified view of issue tracking falls apart. For example, many software development teams require issues to be verified before officially being marked as closed:

> *When a bug is resolved, it gets assigned back to the person who opened it. This is a crucial point. It does not go away just because a programmer thinks it should. The golden rule is that only the person who opened the bug can close the bug. The programmer can resolve the bug, meaning, “hey, I think this is done,” but to actually close the bug and get it off the books, the original person who opened it needs to confirm that it was actually fixed or agree that it shouldn't be fixed for some reason.*
>
> —FogBugz Documentation [Fog Creek Software 2009b]

The state diagram illustrated in Figure 2.2 was reproduced from Bugzilla [Bugzilla 2009]—an open-source issue tracking system—and shows a more complicated, albeit more realistic, set of potential paths an issue may traverse on its journey toward being closed.

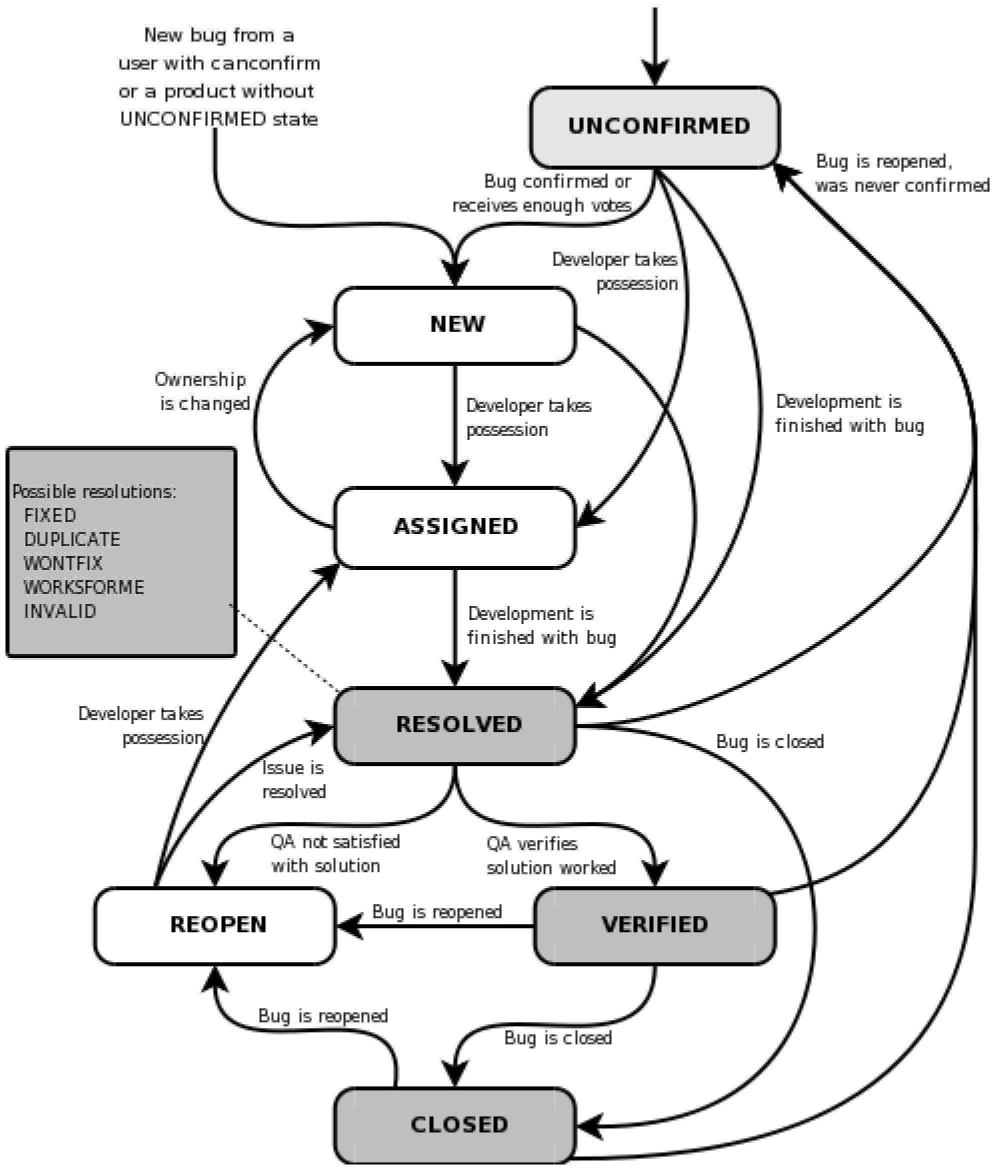

New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug is reopened, was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

Figure 2.2 – Issue lifecycle supported by Bugzilla [Bugzilla 2009]

As we can see, the workflow supported by an issue tracking system can quickly escalate to something much more complicated than the rudimentary workflow illustrated earlier in Figure 2.1. In addition to supporting a larger number of states and transition options, the Bugzilla workflow shown in Figure 2.2 also reveals that some states may comprise of multiple sub-states or "statuses." The most common state to exhibit this is the "resolved" state. This is because an issue can make the transition into the resolved state for a variety of reasons—most of which are useful to record for future reference. For example, an issue can be resolved as “duplicate” if another issue exists representing the same problem or feature request. Usually in such an instance the unique identification number of the duplicate case would be stored along with the resolved issue. Alternatively, a bug could be resolved as “worksforme” (i.e., works for me) to indicate that the person resolving it could not reproduce the problem: he or she either needs more information detailing how to reproduce it or perhaps the bug has already been fixed.

The key purpose of issue tracking software, therefore, is to support these detailed workflows and catalogue information as it is appended to each issue during its traversal of these paths. In addition to tracking a specific state of an issue within the workflow, other relevant attributes are also recorded. For example, each issue is “owned” or “assigned” to a specific person within the software team. Tracking such ownership of who is working on what helps provide accountability for ensuring each issue in the system has someone looking after it (this is discussed further in Section 5.5). The underlying database structure of the issue tracker keeps track of all of these various attributes in such a way as to also allow those who use the issue tracking system to search for issues and group them according to various criteria (e.g., by priority, by who initially created the case, by current owner, by date created, etc.).

2.2 Related Work

I now present a brief summary of existing research relating to the social aspects of software development and its inherent communication and coordination challenges. Prior work specifically examining issue tracking systems is also discussed as well as research looking at the how specific software development tools address the communication and coordination needs of a development team. This section’s goal is to frame the rest of this thesis and elucidate the niche this work fills within the existing body of literature. While an overview is presented here, specific theoretical literature as relevant to particular sub-topics will be introduced in later chapters.

My thesis is that issue tracking systems fundamentally support the communicative and collaborative aspects of software development, even within small, collocated teams. They do this by providing a robust, canonical, and well compartmentalized database of the work items completed by the development team over time and additionally by storing the discussions and decisions surrounding the issues they contain. In the first section below, I review work that has observed, identified, or examined the social aspects of software development coordination and communication in general and that hint toward the importance of exploring the role of issue tracking systems. The following section then presents the relatively scant literature that has addressed issue tracking systems specifically, and is followed by an investigation of where these two areas overlap; that is, where issue tracking and software development coordination meet. Through this process I will reveal the niche that is explored by the remainder of my thesis: the communication, coordination, and collaboration patterns, practices, and tools of small, collocated teams as they relate to issue tracking systems.

2.2.1 The social aspect of software development coordination

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 (CSCW) and software engineering to examine how software teams communicate with each other and coordinate their work. Perry et al. [1994], for example, studied individual developers' perceived and actual time allocation for various activities throughout their day-to-day work. Their studies found that *over half* of each developer's time was spent interacting with co-workers.

In a similar vein, Kraut and Streeter's [1995] examination of the formal and informal aspects of the software development process found that while

> *technical problems continually arise in the process of creating software, and while people can solve some of these problems themselves or by examining relevant documents, other problems demand information or cooperation from other people.*
>
> —Kraut and Streeter [1995]

This led them to the realization that while personal communication is pivotal to coordination, it may be "too expensive, too ephemeral, or too local to be an effective communication mechanism" if not assisted via technological means. As will be shown in later chapters, issue tracking systems often provide precisely the assistance necessary to overcome these challenges, even when dealing with relatively small, collocated teams.

More recently, Ye [2006] argued that software development is both knowledge-intensive and collaborative. His 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 et al. [2006]: in addition to noting developers' reliance on their social network for determining code rationale, their study found that developers at Microsoft "reported spending nearly half their time fixing bugs." If developers in a corporation as large and established as Microsoft are spending such a significant portion of their time dealing with bugs, it stands to reason that a thorough examination of issue tracking systems and their communicative properties may be warranted. Unfortunately, this has not been the case within the fields of CSCW and software engineering until recently.

2.2.2 Examining issue trackers specifically

Although much analysis 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, especially within the context of small, collocated teams that are still the norm in many organizations today. Most existing work touching on this aspect of software development has focused primarily on distributed teams within an organization, or distributed communities that work on open-source development. For example, Sandusky and Gasser [2005] found that the issue tracker used by an open-source development team was often a primary and logical location for much of the distributed negotiation involved in resolving bugs. Similarly, Halverson et al. [2006], while developing two prototype visualizations, found issue trackers provided a locus for negotiation, "often through extended interactions that involve debate among developers, reaching consensus, or soliciting management input" within the teams they studied.

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 [Bettenburg et al. 2008], identifying who should work on a given issue [Anvik et al. 2006, Canfora and Cerulo 2006], visualizing various aspects of the issue tracker's data [Halverson et al. 2006, Ellis et al. 2007, D'Ambros et al. 2007], and improving developers' ability to detect defects in their systems [Fenton and Neil 1999]. 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. Examples of earlier work looking specifically at issue tracking systems include that of Carstensen et al. [1995], where they observed and examined the coordination aspects of issue tracking. However, this was using earlier, paper-based workflow systems and again involved a distributed team.

2.2.3 Where issue tracking and development coordination meet

When looking for prior work that has specifically examined the coordination dynamics of issue tracking systems, we unfortunately do not find much. If, however, we slightly broaden our search to include other, closely-related development tools such as configuration management systems, we have slightly better luck. Configuration management tools aim to control developers' ability to alter code. By doing so, these systems provide (at least some) support for the collaborative work of software developers by mitigating the inherent complications of multiple people making changes to the source code of a project at the same time. By providing mechanisms that help regulate this process and prevent one developer from unknowingly overwriting the work of another, they help coordinate the team's efforts in a productive direction. Grinter [1995], for example, examined the use of configuration management tools for the coordination of software development and found that while such tools helped, and "despite well-defined policies surrounding tool usage and a good cognitive understanding of the tool, coordinating software development remains difficult."

Tools such as configuration management systems help coordinate development efforts at a low level—that of individual source code files. By regulating access to these files, such tools make developers on a team immediately aware of the work of their colleagues, especially if they both attempt to modify the same file at the same time. One of the conclusions drawn by Grinter [1995] was that despite this assistance in coordinating developers' efforts, "the developers still use other communicative solutions to overcome challenges of coordinating work".

If we raise ourselves one level higher in the software development process from the physical manipulation of source code files, we can see where issue tracking systems come to play a role in developer coordination and communication. Most of the times when a legitimate issue (i.e., not a duplicate) is filed in the issue tracking database, the likely outcome will involve modifying the source code of the system under construction. In a way, the issue tracker serves as an early warning system that code changes are likely to follow, and by capturing changes at this earlier stage (in addition to the communication tools provided by the tracker itself) can further improve the coordination and awareness of software developers on the team.

In a somewhat similar vein, Fitzpatrick et al. [2006] attempted to help fill this apparent gap between the coordination mechanisms provided by configuration management systems and the additional communication means needed by developers to coordinate their work. They developed a lightweight notification and chat system that they integrated with a configuration management tool. Their tool consisted of a tickertape-style notification window that scrolled the commit log messages submitted by developers along with their code changes. Those using the system could open a chat dialogue in response to such messages as they saw fit. After observing a team's long-term use of this tool they found that by augmenting the configuration management system with this tool the developers were *able to maintain an overview of what code changes are happening, engage in social chit-chat, have a timely work discussion, engage in collaborative problem solving, negotiate the flow of work, ... and promote team culture.*

—Fitzpatrick et al. [2006]

Thus, this work also suggests that tools providing heightened awareness and the ability to discuss the changes being made to the underlying source code are likely to be beneficial to the development process.

If we look more recently, Aranda and Venolia's work [2009] has finally started to bridge the gap between broader CSCW research and the specific use of issue tracking systems as a software engineering tool. Aranda and 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 detailed examination of these bug histories they found that the assumption regarding the completeness of histories stored within issue tracking databases was not founded as "electronic repositories hold incomplete or incorrect data more often than not." In addition, they issued 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.

—Aranda and Venolia [2009]

As we shall see, this challenge is taken up in the remainder of this thesis.

2.3 Discussion & Framing of the Thesis

The chapter so far, has revealed several aspects of issue trackers. First, it is portrayed as a technical system that manages and records transitions—sometimes complex transitions—about the state of an issue as it moves from an open to an ultimately closed state. Second, we saw that software development in general, and issue tracking specifically, can be seen as incorporating a social process, albeit one that is not explicitly supported by development tools directly. This creates the following challenge: how do we understand and improve coordination dynamics as part of our issue tracking systems?

It is precisely these hidden coordination dynamics that I set out to uncover in the empirical study described in the following chapters of this thesis. I and my collaborators conducted a qualitative study of issue tracking systems as used by small, collocated software development teams. We found that an issue tracker is not *just* a database for tracking bugs, features, and inquiries as described in Section 2.1, but also a focal point for communication and coordination for many stakeholders within and beyond the immediate software development 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. These results were all the more striking to us because in spite of the teams examined being collocated—which afforded them frequent, low-cost, face-to-face communication—the issue tracker was still used as a *fundamental* communication channel.

In this chapter, I introduced the general terminology of issue tracking systems and explained their basic functionality. Issue trackers generally serve as a database of records that support the aggregation of data surrounding the contained records over time. These records may be defects or "bugs," feature requests, or customer inquiries and the data they accumulate can include simple attributes (e.g., creation date, priority, owner, etc.) as well as developer conversations in the form of comment histories. In addition to storing these records and their respective attributes, issue trackers also facilitate a lifecycle workflow which these issue records traverse on their way from being initially opened to eventually being closed. Both simple and complex workflows were presented to illustrate how the intuitive and perhaps ideal differs from the practical and realistic. Finally, this chapter presented a brief overview of literature related to this thesis and identified the niche within the field that it addresses: the in-depth exploration and analysis of issue tracking systems usage within small, collocated, industrial development teams.

Chapter 3. Research Study

The next three chapters detail a qualitative study that explores the communication and collaboration aspects of professionals who perform issue tracking as part of their day-to-day work. In this chapter, I describe the study methodology, the data collected, and how the data was analyzed. The subsequent chapters detail and discuss the results.

3.1 Description of the Experiment

3.1.1 High-level goals/questions

As described in the previous chapter, issue tracking systems have matured over time, where they have leveraged advances in technology and interface design to better support the tracking of bugs, feature requests, and customer inquiries. While such systems are well suited for tracking bugs within a database, we also know that development team members communicate, coordinate, and collaborate with one another as they collectively delegate, manage, and attempt to comprehend particular issues. Such team interaction—especially for collocated teams—has not been rigorously nor directly examined. We do not have a good understanding of how issue tracking system facilities are currently used to support—directly or indirectly—the interactions of small, industrial, collocated software development teams.

Consequently, I designed and executed a study to better understand the role of such communication, collaboration, and coordination in the issue tracking systems used by such teams. While I expected to uncover how professional developers used their particular tools, my primary goal was *to see how these tools fit into the larger role of coordinating the development process, facilitating cooperation within the development team, and supporting the necessary communication among its members*. For each team examined in my study, I aimed to gain a holistic understanding of their software development and release cycle, while also drilling down into the day-to-day practicalities of using issue trackers and how they fit into each team's custom-tailored processes and practices. Consequently, I studied not only the use of the issue tracking systems themselves, but also the integration between such systems and other development tools (source code repositories, code editors, email systems, wikis, etc.). Usage frequency and motivation for a wide variety of communication channels (email, chat, instant messaging, wikis, micro-blogging, verbal conversation, etc.) were also investigated, with a specific interest in how these various channels poured into, sprouted from, and interoperated with the issue tracking system and the team's development cycle.

3.1.2 Subjects

Target audience

Small, collocated, industrial software development teams made up the primary audience for our study. This subset of the larger software development community was chosen for a number of reasons. *Small development teams* consisting of 5–30 members were targeted due to their increased likelihood for frequent, intimate communication episodes. Small teams also allowed for a more holistic sampling of the development team and its composite roles while keeping the necessary number of sampled participants low. *Collocated teams* were selected to allow for the inclusion of questions pertaining to face-to-face conversations including casual, serendipitous communication episodes such as those that might occur around the lunch table, at the water cooler, or in the hallway. Because such teams are readily able to communicate face-to-face, a collocated team's use of technology *over* face-to-face suggests a positive and/or significant role for that technology. Finally, *industrial development teams* were selected, as these teams exhibited professional cultures whose processes and practices had been developed and refined over time. An "industrial" development team in this context refers to one which works directly on the production of a software product that is used by a collection of clients in a professional or commercial setting. For instance, this distinction includes roles such as technical staff at a university whose job is the maintenance and customization of a commercial software package used by others within that university, but excludes laboratory settings such as researchers or students producing prototype software as part of their research.

Recruiting process

Participants were recruited using a largely convenience-based snowball sampling method. Existing relationships with software development teams established by me and my colleagues within the Interactions Lab at the University of Calgary were used to identify potential participant teams. For example, after speaking with various members of the IT staff at the University of Calgary, I then broadened my search based on their recommendations and referrals. Through my own work experience and that of other students in the Interactions Lab (past and present), several additional potential development teams were identified.

After contacting a variety of teams through these means, careful stratification of the pool of development teams was considered to ensure as broad a coverage as possible given the constraints of my study (time, travel expense, etc.). Each of the development teams chosen (described below) belonged to a different organization and wrote software targeting a different market within the broader software development field. Each team made use of at least one formal issue tracking system in support of their day-to-day development tasks.

After selecting which teams to pursue, I then described the type of participants I was hoping to interview (outlined in the following paragraph). Many of the participants in our study volunteered to be interviewed, while others were asked by our contact person (often a manager within the organization) to participate.

Study participants from each of the selected teams were recruited, with the assistance of our primary contact person, in order to sample three primary roles within a typical professional software team:

- *Developer or Team Lead.* This role consisted primarily of programmers, but included those involved in the design and specification of the system under development. This role included those acting as a "team lead" or "senior developer," but excluded those in strictly management-type positions (see the PM role below). All participants in this category wrote code as a significant portion of their day-to-day activities.

- *Quality Assurance or Tester.* The quality assurance (QA) role included those whose primary function was to test the software under development. This included not only the physical act of testing the software itself, but also the design, refinement, and maintenance of test scripts. Some participants in this category also served as front-line usability test participants (e.g., thinking aloud while being asked to exercise new program functionality in order to uncover usability and implementation flaws). Additionally, some QA team members also wrote code as part of their job responsibilities, but such coding was done to facilitate or automate their testing work rather than to contribute directly to the customer-facing functionality of the system under development.
- *Project or Program Manager.* The project or program manager (PM) role included team members that either did not write code at all, or did so only on rare occasion (e.g., email auto-filtering scripts). Such people primarily managed those in the roles mentioned above. They also typically defined the high-level design of features, and shaped the direction of the system under development. The majority of our PM participants also spent a significant portion of their time dealing with customer support issues either directly, or with the assistance of dedicated customer support staff.

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. 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 (see Participant demographics below).

Development team profiles

In total, four North American software development teams were selected for our study, each belonging to a different organization. Three of the four teams studied produced commercial software while the other worked primarily on the customization and maintenance of a commercial software product used by other departments within their organization. A brief description of each of the development teams studied follows:

Team A: University IT Department

Team A was a development team within a large university's IT department. At the time of the study, the university had over 27,000 students enrolled including undergraduate, graduate, and professional degree programs and over 5,000 academic and support staff. The development team itself was composed of five developers, seven QA staff, and two PMs. All members of the team interviewed were collocated on the same floor within a single building. The participant team's responsibilities included the maintenance, customization, and support of a third-party commercial human resource management system used by a number of other departments on campus.

Team B: Gaming Company

Team B was a product development team within a large gaming company. At the time of the study, the company employed over 400 people across its multiple North American offices. The participant team was responsible for the production of portions of an upcoming role-playing game and was composed of approximately 20 developers, five QA staff, and three PMs. All members of the team interviewed were collocated in the same building spread across several floors.

Team C: Issue Tracking System Vendor

Team C was part of a small commercial software company that produced issue tracking and project management software as well as a remote desktop application. At the time of the study, the company employed approximately 25 employees. The development team interviewed was responsible for the production of the company's commercial issue tracking and project management product. The issue tracker used by this team was the same issue tracking system they were developing. Six developers, two QA staff, and one PM made up the participant development team with all members of the team collocated on the same floor within a single building.

Team D: Large Interactive Display Vendor

Team D formed a product development team writing software to accompany a large interactive display hardware product. Team D's organization employed over 1,000 people worldwide across multiple offices at the time of the study. The specific development team interviewed was composed of eighteen developers, eight QA staff, and six PMs. All members of the team were collocated in the same building spread across two floors.

Participant demographics

In total, 15 individuals from the above described software development teams took part in our research study (Table 3.1). The participants' ages ranged from 25 to 51 years with a mean of 33.4 and a median of 33. Their experience with issue tracking software ranged from 2 to 16 years (including any gaps) with a mean of 5.6 years and a median of 4. As mentioned earlier, gender balancing did not play a role in the recruiting process; however, our participant pool did include at least one female participant in each of the roles of interest. Table 3.1 reveals that the number of participants interviewed from each team varied from two to five (two from Team A, four from Team B, five from Team C, and four from Team D). It also reveals the diversity of issue trackers used: Team A used Microsoft Excel [Microsoft Corporation 2009a] and Microsoft Sharepoint [Microsoft Corporation 2009b] (systems not designed explicitly for issue tracking), Team B used ExtraView [ExtraView 2009] as well as its own proprietary software (an abstraction built on top of ExtraView that is described in further detail in Section 5.3), while Teams C and D both used the web-based FogBugz [Fog Creek Software 2009a] issue tracking system.

Software Team	Participant Demographics						Issue Tracker Used
	Developer/Lead		Quality Assurance/Tester		Project/Program Manager		
	Male	Female	Male	Female	Male	Female	
Team A University IT Dept.	P01			P02			SharePoint & Excel
Team B Gaming Company	P06		P04, P05			P03	ExtraView & proprietary
Team C Issue Tracker Vendor	P08, P09, P11			P07	P10		FogBugz
Team D Large Display Vendor	P15	P13		P12	P14		FogBugz & Excel

Table 3.1 – Overview of participant population

Table 3.1 also reveals that we were reasonably successful in recruiting a cross-section of the various roles within the development team (excepting Team A, where we did not interview a PM). This was important in helping us understand the interplay among the three primary roles of interest (developer, QA, and PM), and in particular between the developers and QA staff working "in the trenches."

3.1.3 Methods and materials

We primarily used qualitative methods for our study, specifically, semi-structured interviews. The interviews were conducted in-situ (i.e., in each participant's regular office) whenever possible to both preserve work context and facilitate the contextual interview aspects of each study session (see Section 3.1.4) [Beyer and Holtzblatt 1997]. A variation of the grounded theory approach to data analysis was later applied to our interview data (described further in Section 3.2) [Corbin and Strauss 2008].

Initial study formulation

We began designing our study by thinking about the following research questions:

1. What role does lightweight communication (email, chat, instant messaging, wikis, twitter, verbal conversation, etc.) play in the day-to-day completion of software development tasks?
2. How does such communication apply to, interoperate with, and sprout from issue tracking tasks (filing software bugs, working on feature requests, handling customer support problems, etc.)?
3. How might this communication be better leveraged and/or supported by software development tools?

From these initial questions we iteratively expanded (and then later refined) the set of questions we were interested in exploring. This list of questions later became the interview protocol discussed below.

A questionnaire targeted at capturing participant experiences and practices with lightweight communication in respect to issue tracking tasks—as well as basic experience and demographic information—was also created. This questionnaire was administered prior to the semi-structured interview portion of each participant's session and served to help frame each participant's mind within the area(s) of their work we would later be interviewing them about.

Some initial sketches of potential development and communication tools were also drafted throughout the course of our study based on my own personal experience and intuition as well as participant comments, ideas, and opinions obtained through interview sessions.

Ethics approval and research activities

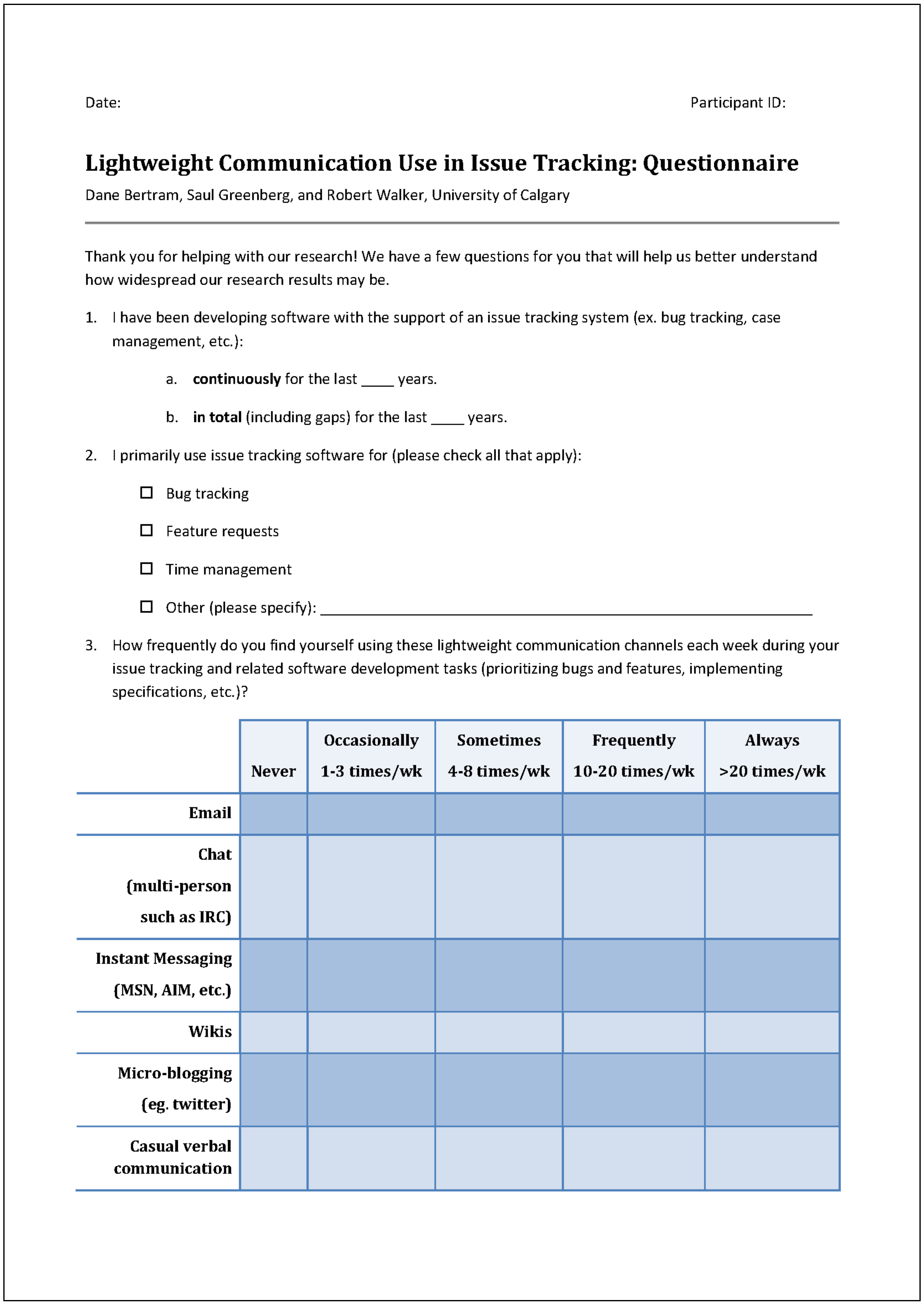
After developing the initial drafts of the questionnaire and interview protocol, we submitted an ethics application to the University of Calgary's Conjoint Faculties Research Ethics Board (CFREB), which was approved after minor revisions (see Appendix A). This application outlined the following set of research activities each participant could be asked to partake in, where participants would be compensated $20 for their time.

1. A questionnaire about the participant and their experiences and practices with lightweight communication in respect to issue tracking tasks (see below).
2. A contextual interview during issue tracking tasks and related software development tasks (whichever issue tracking tasks the participant may be working on or preparing to work on as part of their regular work routine) for a period of time not to exceed two hours. During these tasks a researcher will observe and make notes. The participant may be asked to think aloud and clarifying questions may also be asked by the researcher.
3. A semi-structured interview about their software development habits related to issue tracking and their usage of lightweight communication channels. A series of open-ended questions will be used to guide the general direction of the interview, but emergent issues and ideas will be explored as they come up (both that arise during the interview and that arise during the observations).
4. Sketching of their ideas for potential user interfaces and tools that would make use of such communication (including design suggestions and potential requirements for addressing their needs and wants).
5. Feedback on prototype systems and software development tools (sketches, computer software, textual and verbal descriptions, etc). This task may be a recurring one in order to facilitate an iterative design process.

I conducted a dry-run of my study with a post doctoral researcher at the University of Calgary who had software development experience, and through this process, I improved various aspects of the protocol and mechanics, (e.g., audio recording quality), and made refinements and clarifications to both the questionnaire and interview questions. In practice, due to time constraints most interview sessions followed an abbreviated agenda consisting of the questionnaire and a combined contextual/semi-structured interview as described in Activities 1–3. Only participants from Team C were explicitly involved in the sketching feedback and brainstorming activities of Activities 4–5 (see below).

Questionnaire

The questionnaire administered to participants (Figure 3.1 and Figure 3.2) was designed to gather information about their experience and usage practices with issue tracking and lightweight communication tools. As shown in Figure 3.1, we asked each participant about the length of their experience developing software with the support of an issue tracking system both continuously as well as in total (i.e., including any gaps). Primary uses for their issue tracker were solicited along with approximate frequencies of use for various communication channels (email, chat, instant messaging, verbal communication, etc.). Basic demographic information such as age and sex were also requested (Figure 3.2).



Date: Participant ID:

Lightweight Communication Use in Issue Tracking: Questionnaire

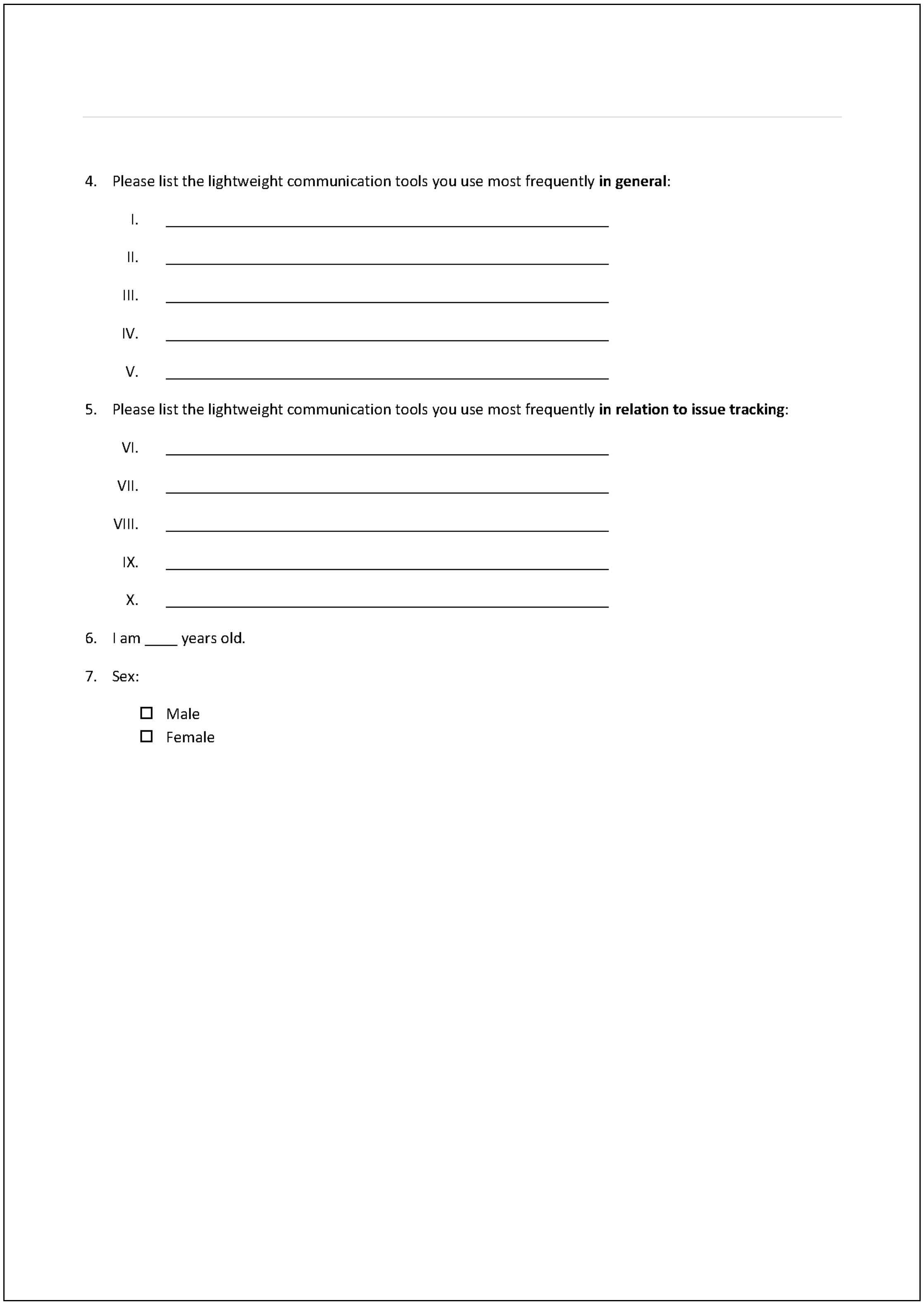
Dane Bertram, Saul Greenberg, and Robert Walker, University of Calgary

Thank you for helping with our research! We have a few questions for you that will help us better understand how widespread our research results may be.

1. I have been developing software with the support of an issue tracking system (ex. bug tracking, case management, etc.):
   a. **continuously** for the last ____ years.
   b. **in total** (including gaps) for the last ____ years.
2. I primarily use issue tracking software for (please check all that apply):
   - ☐ Bug tracking
   - ☐ Feature requests
   - ☐ Time management
   - ☐ Other (please specify): ______________________________
3. How frequently do you find yourself using these lightweight communication channels each week during your issue tracking and related software development tasks (prioritizing bugs and features, implementing specifications, etc.)?

	Never	Occasionally 1-3 times/wk	Sometimes 4-8 times/wk	Frequently 10-20 times/wk	Always >20 times/wk
Email					
Chat (multi-person such as IRC)					
Instant Messaging (MSN, AIM, etc.)					
Wikis					
Micro-blogging (eg. twitter)					
Casual verbal communication					

Figure 3.1 – Questionnaire (page 1 of 2)



4. Please list the lightweight communication tools you use most frequently **in general**:

    I. ______________________________

    II. ______________________________

    III. ______________________________

    IV. ______________________________

    V. ______________________________

5. Please list the lightweight communication tools you use most frequently **in relation to issue tracking**:

    VI. ______________________________

    VII. ______________________________

    VIII. ______________________________

    IX. ______________________________

    X. ______________________________

6. I am ____ years old.

7. Sex:

    ☐ Male
    ☐ Female

Figure 3.2 – Questionnaire (page 2 of 2)

The questionnaire was administered after each participant completed the informed consent process, but prior to the contextual/semi-structured interview. In addition to following the procedure outlined in our ethics application, this ordering also primed our participants for their interview by helping to both frame the context of our study and also put them into the mindset of the types of stories, experiences, and problems we were looking to explore.

Interview protocol

The contextual/semi-structured interview activity, which was audio-recorded, roughly followed a list of fourteen guiding interview questions and requests (see below). Some of these fell under a more contextual interview style (e.g., “Please walk me through the process of adding a new bug into the issue tracker.”), while others followed a more traditional question-and-answer format (e.g., “When working with bugs in the issue tracker, what are the other software tools you use most frequently?”). Throughout this activity the observer took notes of interesting, surprising, and/or unclear comments or actions made by the participant to be followed up with later during the interview. Emergent issues were explored in greater depth as time, relevance, and significance permitted.

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 as well as selecting among multiple available channels. We also explored shortcomings and workarounds related to their existing tools—both those related to communications as well as those targeting general software development.

The basic interview protocol was as follows, with additional questions inserted opportunistically based on observations made and participant responses:

1. Can you please give me a tour of the issue tracking system you use?
2. What issue tracking system do you use? Have you always used this system in your current position, or have you switched recently?

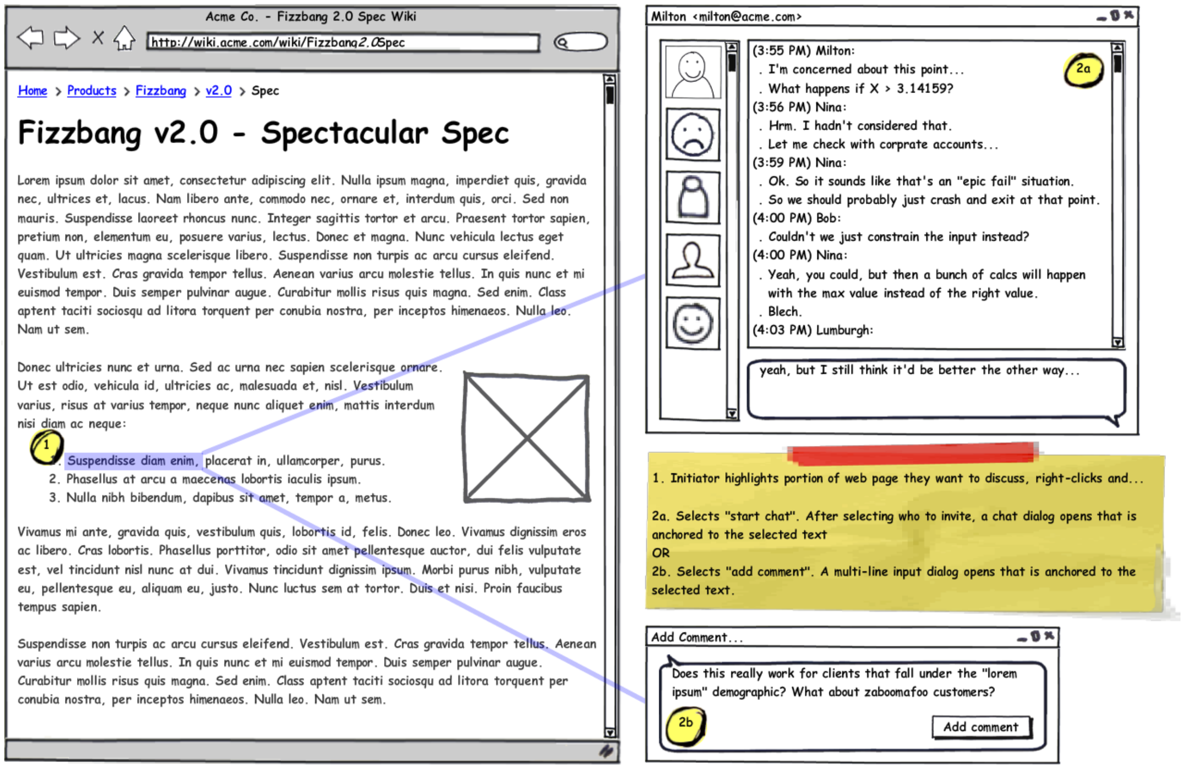
3. Can you please describe the primary purpose of issue tracking systems from your perspective?
4. What are the different roles and responsibilities of the people working with the issue tracking system in your department (including yourself)? Do certain people have different levels of access to the issue tracking database than others?
    a. Is there a QA process?
    b. Do outside stakeholders (users) have the ability to enter bugs and/or feature requests directly/indirectly into the issue tracker?
    c. Does management make use of any reports or metrics from the issue tracker?
5. What is your workflow for managing and resolving bugs/features? What are the steps in the process? Who is/are responsible for each step?
6. What information is usually tracked with each issue?
7. Please walk me through the process of: ("Tell me about the last time you did X...")
    a. adding a new feature or bug into the issue tracker.
    b. starting to work on resolving a bug
    c. starting to work on implementing a new feature
8. When working with bugs/features in the issue tracker, what are the other software tools you use most frequently? Why do you use these particular tools?
    a. Integration with source code repositories, IDE, email?
9. What types of information do you frequently require from others in order to complete your day-to-day issue tracking and related software development tasks? How do you usually obtain this information? How frequently do you find yourself looking for these kinds of information?
10. What communication channels do you use to communicate with your colleagues and other project stakeholders? How frequently do you use each communication channel?
    a. Micro-blogging (e.g., twitter)
    b. Instant messaging
    c. Chat (e.g., IRC)
    d. Email
    e. Wiki
    f. Mailing lists/bulletin boards
    g. Verbal conversation (face-to-face, phone/Skype)
    h. Physical notebook

11. Do you have a preference as to which communication channel you use? If so, under what conditions and why?
    a. Try to discern motivation for use of one channel over another
    b. Synchronous vs. asynchronous component
    c. Normally archived vs. not normally archived (i.e., availability of communication history to the rest of the team, management, etc.)
12. How frequently is the information obtained via lightweight communication channels used? Are the outcomes of these communication episodes ever transposed into another location (e.g., issue tracker, wiki, email, notebook, etc.)?
13. What shortcomings have you experienced with the tools you currently use from a communication/collaboration perspective? What workarounds have you come up with? Are there any shortcomings that stop you from completing your issue tracking or related software development tasks?
14. Is there anything else you'd like to tell me about that I haven't asked you about?

During explicit "walkthrough" questions (e.g., Questions 1 and 7) as well as those that often spurred specific examples that could be demonstrated or relived (e.g., Questions 4b, 4c, 5, 6, 8, 10, 12, and 13), participants would often—and were also encouraged to—tell complete stories of how a given bug or feature (for example) evolved over time from its initial conception through to its final or current state.

Design sketch roundtable

Throughout the research study, I created a number of design sketches that illustrated enhancements to development and communication tools as well as issue tracking systems (see Chapter 6 for greater detail and examination). Most of these mock-ups were created between the week leading up to our first participant session and the end of our study some weeks later. After completing all of the contextual/semi-structured interviews, a subset of the participants from Team C were asked to give feedback on a set of these potential software interfaces. The interfaces were presented as annotated paper-based sketches (e.g., Figure 3.3) developed using the Balsamiq Mockups [Balsamiq Studios 2009] rapid interface prototyping tool. Pros and cons of the presented designs as well as ideas for other potential interfaces were solicited.


Acme Co. - Fizzbang 2.0 Spec Wiki

http://wiki.acme.com/wiki/Fizzbang2.0Spec

Home > Products > Fizzbang > v2.0 > Spec

Fizzbang v2.0 - Spectacular Spec

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ipsum magna, imperdiet quis, gravida nec, ultrices et, lacus. Nam libero ante, commodo nec, ornare et, interdum quis, orci. Sed non mauris. Suspendisse laoreet rhoncus nunc. Integer sagittis tortor et arcu. Praesent tortor sapien, pretium non, elementum eu, posuere varius, lectus. Donec et magna. Nunc vehicula lectus eget quam. Ut ultricies magna scelerisque libero. Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Donec ultricies nunc et urna. Sed ac urna nec sapien scelerisque ornare. Ut est odio, vehicula id, ultricies ac, malesuada et, nisl. Vestibulum varius, risus at varius tempor, neque nunc aliquet enim, mattis interdum nisi diam ac neque:

1. Suspendisse diam enim, placerat in, ullamcorper, purus.
2. Phasellus at arcu a maecenas lobortis iaculis ipsum.
3. Nulla nibh bibendum, dapibus sit amet, tempor a, metus.

Vivamus mi ante, gravida quis, vestibulum quis, lobortis id, felis. Donec leo. Vivamus dignissim eros ac libero. Cras lobortis. Phasellus porttitor, odio sit amet pellentesque auctor, dui felis vulputate est, vel tincidunt nisl nunc at dui. Vivamus tincidunt dignissim ipsum. Morbi purus nibh, vulputate eu, pellentesque eu, aliquam eu, justo. Nunc luctus sem at tortor. Duis et nisi. Proin faucibus tempus sapien.

Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Milton <milton@acme.com>

(3:55 PM) Milton:
. I'm concerned about this point...
. What happens if X > 3.14159?
(3:56 PM) Nina:
. Hrm. I hadn't considered that.
. Let me check with corprate accounts...
(3:59 PM) Nina:
. Ok. So it sounds like that's an "epic fail" situation.
. So we should probably just crash and exit at that point.
(4:00 PM) Bob:
. Couldn't we just constrain the input instead?
(4:00 PM) Nina:
. Yeah, you could, but then a bunch of calcs will happen with the max value instead of the right value.
. Blech.
(4:03 PM) Lumburgh:

yeah, but I still think it'd be better the other way...

1. Initiator highlights portion of web page they want to discuss, right-clicks and...

2a. Selects "start chat". After selecting who to invite, a chat dialog opens that is anchored to the selected text
OR
2b. Selects "add comment". A multi-line input dialog opens that is anchored to the selected text.

Add Comment...

Does this really work for clients that fall under the "lorem ipsum" demographic? What about zaboomafoo customers?

Figure 3.3 – Sample design sketch

During this session each interface was explained verbally and any questions regarding the envisioned functionality were answered. Feedback ranged from technical feasibility concerns, to alternative interactions, to entirely different communication and/or development tools. The sketches themselves along with their pros and cons and the feedback received are examined in Chapter 6.

3.1.4 Problems and limitations

Participant interviews were carried out in each participant's regular workspace whenever possible. As is typical with contextual interviews, particular problems did arise. For example, while story-telling, participants sometimes moved far away from the topic of issue tracking. In these cases, we had to re-orient the discussion to bring it back "on track." Some participants also had their own constraints that interrupted the interview. For example, P04 had an abbreviated interview due to conflicting meetings. The logistics of meeting with particular teams also affected what we could do. For example, Team B's members 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. This occasionally added some extra work for participants in accessing their software (e.g., some pages normally displayed automatically had to be loaded manually, some people worked with dual monitors while only a single monitor was available in the office used, etc.). All problems experienced were relatively minor, as participants were always able to explain and show what they were doing and access the requisite applications and websites.

Across the four teams studied, only three major commercial issue tracking software packages were observed (Microsoft SharePoint [Microsoft Corporation 2009b], ExtraView [ExtraView 2009], and FogBugz [Fog Creek Software 2009a]), plus an additional proprietary abstraction/integration layer used by Team B. Given the large number of both commercial and open-source issue tracking systems currently available, our study's observations may not generalize to the broader use of issue tracking systems as a whole. To help address this, many of our results (see Chapters 4, 5, and 6) focus on higher-level, crosscutting concerns rather than product-specific shortcomings (unless noted otherwise). Similarly, our study focused on the use of issue tracking and communication tools by small (<30), collocated, non-open-source development teams. As such, our observations likely differ from those seen in larger and/or primarily distributed teams, such as those seen in many popular open-source projects [Anvik et al. 2006, Bettenburg et al. 2008, Reis and de Mattos Fortes 2002].

Finally, many of our participants self-identified as being interested in our study and, as such, may have been biased. For example, after hearing about a study being done on communication and issue tracking, developers with "an axe to grind" about their current toolset may have been more likely to volunteer or otherwise participate (although we saw no evidence of this in our data).

3.2 Data Collection & Analysis

Our raw data comprised field notes, audio recordings of the interviews, and questionnaire results. After completing our research study, a multi-staged analysis process then began. This process was made up of two primary phases: transcription of the interview audio recordings followed by a grounded theory-inspired approach to the analysis of the resultant interview data [Corbin and Strauss 2008]. Full transcription of interview audio recordings was chosen as part of our bottom-up analysis strategy. A top-down coding approach, whereby a selected set of concepts or "codes" would be listened for, tagged, and extracted from the interview data [Corbin and Strauss 2008], was considered, but discarded due to the open-ended, exploratory nature of our initial research questions and interview protocol. Since we did not set out to reject a specific hypothesis—and despite the initial concepts that appeared to surface during the interviews themselves—we chose instead to let the concepts "bubble up" from the data itself. To support this strategy, a full transcription approach to the raw interview recordings was taken.

3.2.1 Interview transcription

In total, over 21 hours of audio recordings were produced from the 15 participants interviewed, all of whom consented to the use of a digital voice recorder during their respective interviews. Using foot pedals and the associated transcription control software [NCH Software 2009], over 330 pages (when formatted with 1" margins and single-spaced) of transcripts were generated. The transcription process was done by me, where it took approximately 130 hours to complete spread out over the course of several weeks following the fieldwork portion of the study. This intensive transcription task meant I became intimately familiar with the data.

During interviews, participants often referred to visuals (e.g., screen contents) and context-specific items as they performed walkthroughs and/or described examples. These were captured somewhat in the field notes and occasionally supplemented by screen captures obtained with the participant's consent. To augment the audio record during transcription, I consulted my field notes and added annotations to the transcripts when necessary in order to paint an accurate and as-complete-as-possible picture of what each participant was relating to the interviewer.

3.2.2 Data analysis

Once the interviews were transcribed, I took a multi-pass approach to analyse our data from the bottom up. This process was inspired by and largely mimicked the grounded theory approach described by Corbin and Strauss [2008]. The first pass consisted of reading through all the transcripts in detail and highlighting interesting, surprising, or seemingly important or recurring phrases and ideas. To facilitate this note taking process the transcripts were printed using half-page right margins which provided ample space for annotation and note making An example working environment is shown in Figure 3.4, where we see an annotated and marked up transcript page (right side), the accompanying field notes (left side), and various office supplies to assist in the marking-up process.

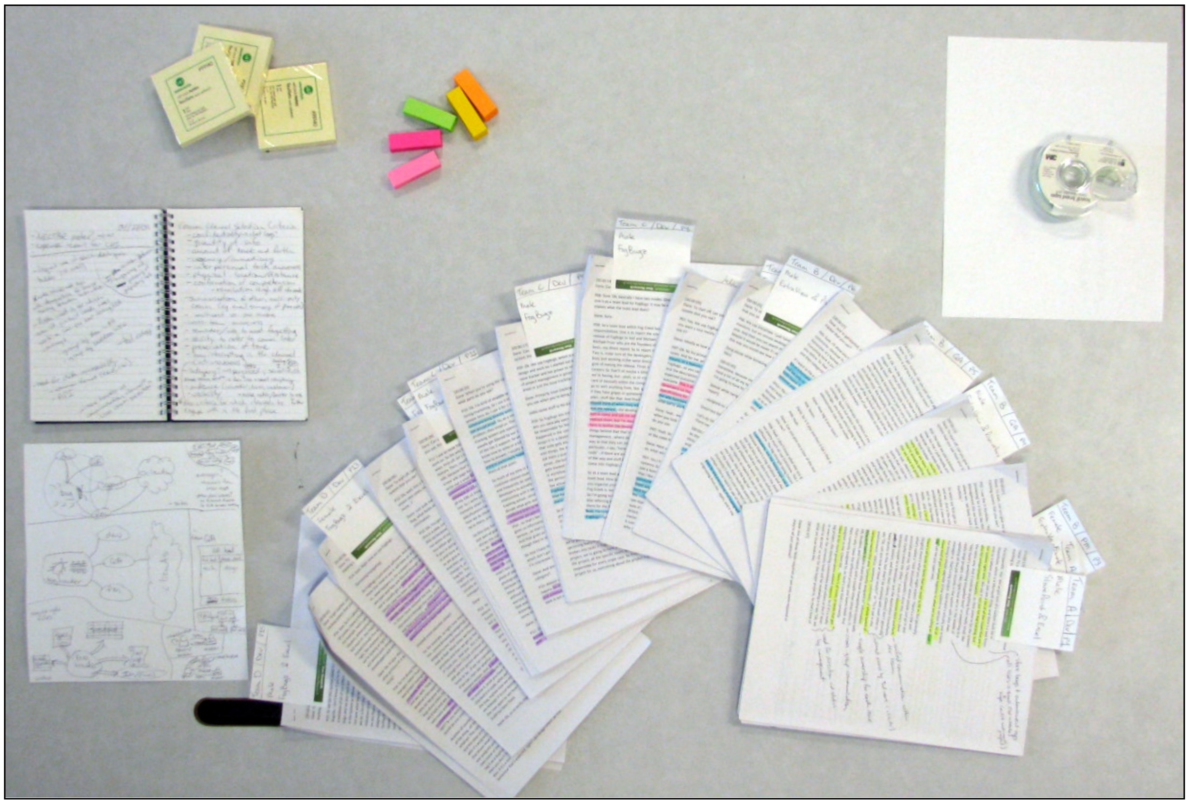


Figure 3.4 – First-pass data analysis artefacts

After completing this first pass, I began the second pass of analysis using Affinity Diagram construction [Beyer and Holtzblatt 1997], i.e., where items are grouped and re-grouped into related clusters, and then labelled as themes emerged. Here, I re-read the transcripts along with my annotations and wrote down salient points and direct quotes on sticky notes and loosely organized them on a large whiteboard. Figure 3.5 illustrates the working environment, where it shows a multitude of notes spread across multiple whiteboard surfaces, spatially aggregated into tentative affinity groups. Intermittently throughout this round of analysis—and usually after a large enough pile of new sticky notes had accumulated—I would stop and organize these notes into the existing groupings or form new ones. After completing the task of generating all the sticky notes I then re-read them and carefully reconsidered their tentative groupings. This resulted in the re-shuffling of existing collections of notes to better reflect the concepts emerging from the data, as seen in Figure 3.6, and the close-up in Figure 3.7. Of course, these groupings of concepts and points were not exhaustive. During affinity diagramming, I identified other possible cross-cutting themes and concepts, but decided to not explicitly group these within the affinity diagram due to their constituent notes already being grouped elsewhere. After multiple passes of reviewing the groupings and reading the sticky notes, I refined and structured the concepts to arrive at the higher-level themes that make up the outline for much of Chapters 4 and 5.

Figure 3.5 – Affinity diagram construction

Figure 3.6 – Theme and concept groupings extracted from interview data

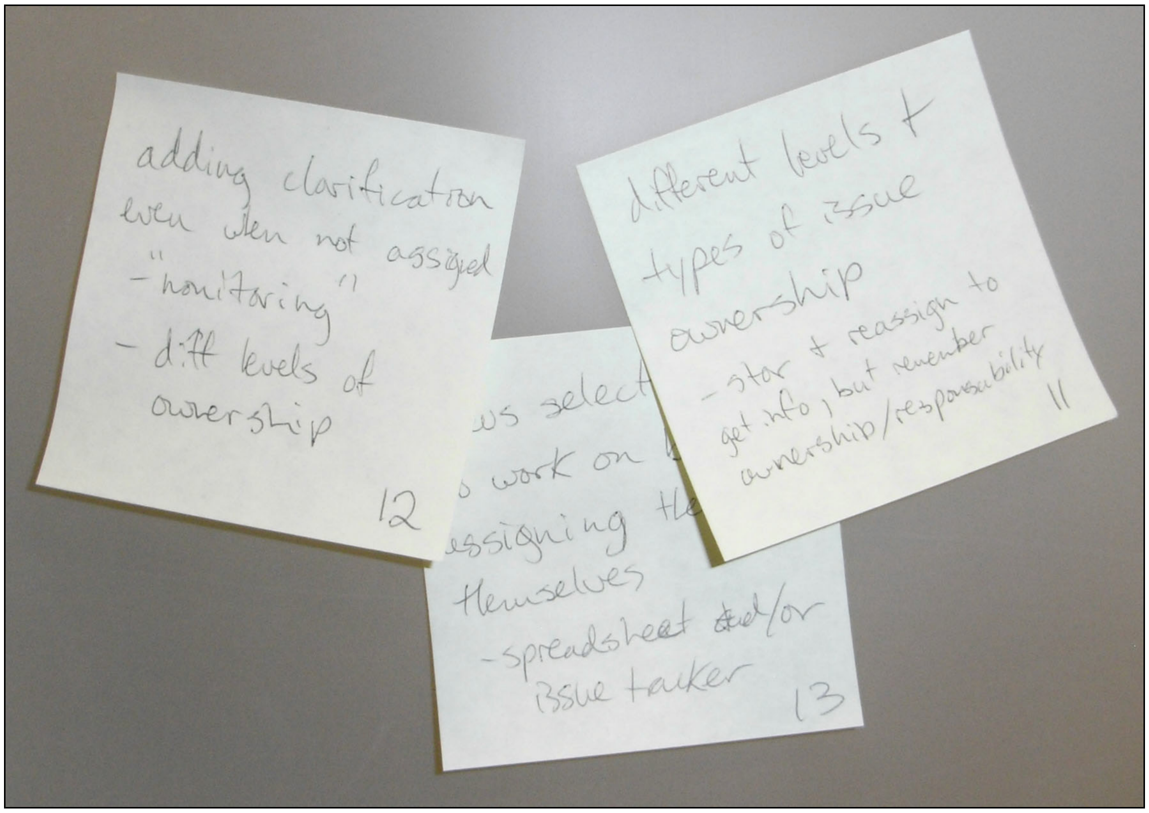

adding clarification even when not assigned
- "monitoring"
- diff levels of ownership
12
ws select
work on
assigning the
themselves
- spreadsheet ~~and~~/or issue tracker
13
different levels + types of issue ownership
- star + reassign to get info, but remember ownership/responsibility
11

Figure 3.7 – Close-up of in-progress concept clustering

Once the concepts had solidified, I performed another pass over the interview transcripts, where I extracted representative quotes that illustrated the themes and concepts identified through the affinity diagramming analysis phase. These quotes are used in the following chapters.

3.3 Summary

This chapter described the process used to design and execute our research study. It began by outlining the high-level goals of my study, its target audience, and the recruiting process used to obtain our participants. Next, the roles within the development team that we chose to investigate were described (i.e., developer, quality assurance, and project manager), along with brief descriptions of the organizations interviewed and a breakdown of our study participant demographics. The materials used in our study (e.g., ethics application, questionnaire, interview protocol, etc.) were presented and the methodology used in our analysis of the resultant data was also described. The next three chapters detail the results obtained from our analysis.

Chapter 4. Roles of the Issue Tracker

In the next two chapters, I present the results from the study outlined in Chapter 3. This chapter covers the results pertaining to the multitude of communication and collaboration roles taken on by the issue tracking system in small, collocated, industrial software teams. Chapter 5 will concentrate on the differing views held by those on the development team who make use of the issue tracker, the functionality breakdowns of such systems, and the incompletely met needs of these development teams in relation to their tools.

As described in Section 3.2.2, I analysed the interview transcripts, identified themes, and extracted representative quotes for each of those themes. I then interpreted these themes, where I took those that reflected existing CSCW theory and tied them back into the literature.

This chapter begins by revisiting the conventional definitions of issue trackers. As we will see in the remainder of the chapter, the conventional definitions do not tell the whole story revealed by our interview data analysis. Subsequent sections show how the issue tracking system satisfied a number of different social, communicative, and collaborative roles for our participant teams. We will see how the issue tracker serves as a knowledge repository, a boundary object, a communication and coordination hub, a communication channel, and as a contextualization repository.

4.1 Conventional Definitions

Most definitions of an *issue tracker* (a.k.a. a defect or bug tracker) approximate Pressman's [2004] description of it as a software tool "that enables the [development] team to record and track the status of all outstanding issues associated with each configuration object."

While a reasonably accurate surface description, in practice most issue trackers also track a variety of other artefacts, e.g., feature requests and inquiries.

In its simplest form, 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—usually software defects—that need to be addressed by the development team. Bugs may be initially reported by customers, the QA team, or anyone else with access to the tracker. *Features* represent new functionality or enhancements to existing code. They usually originate from project managers or team leads. *Inquiries* include everything from sales questions to technical support cases and largely originate from customers.

Issues progress through various states over time (open, resolved, closed, etc.). For each state change or update, various annotations are often stored along with the issue. These annotations typically include attributes, such as the title or due date for resolving a bug, the history of ownership, and discussion about the issue.

Yet, this database view of issues does not tell the whole story. As alluded to in Chapter 2, there are a number of social dimensions to issue trackers that extend well beyond the conventional definitions found in the literature. Through our interviews and the analysis of their transcripts (as described in Chapter 3), we uncovered a number of other *social* dimensions of issue tracking systems and identified a number of interrelated roles served by the tracker. These are described in the following sections.

4.2 Issue Tracker as Knowledge Repository

A *knowledge repository* can be defined as an "on-line computer-based storehouse of expertise, knowledge, experiences, and documentation about a particular domain of expertise" [Liebowitz and Beckman 1998] and that by "creating a knowledge repository, knowledge is collected, summarized, and integrated across sources." In this regard, an issue tracking system is also a knowledge repository. Expertise and knowledge are stored in the form of bug resolution descriptions, troubleshooting steps taken when working on customer support cases, and even in the assignment of bugs and feature requests to the appropriate team member by a PM or team lead based on their knowledge of expertise. The domain of expertise being described is that of the software system being developed. Knowledge is accumulated across many sources, including both human (developers, PMs, QA, customers, etc.) and digital (source code repositories, screen captures, customer email, etc.). The knowledge repository becomes increasingly critical to its users, especially as the volume of knowledge continues to grow over time:

> *Having* [*the issue tracker*] *able to page things back into my brain is definitely the largest impetus for doing everything in* [*the tracker*].
>
> —P09

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* [Ackerman and Halverson 1998]. Ackerman and Halverson clarify that it is often more fruitful to think of organizational memory as "both object and process," and this applies in the realm of issue tracking as well. The underlying database of bugs, features, and inquiries serves as the *object* representing the development team's memory, while the practices established by that team (entering issues into the database, keeping communication about issues with the issues themselves, and searching through previous issues before filing new ones) serve as the *process* that keeps the team's memory accurate, robust, and useful.

> *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, some of which have been described by Dingsøyr and Røyrvik [2003].

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 cannot 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 [Hollan et al. 2000]. 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.

4.3 Issue Tracker as Stakeholders' Boundary Objects

In this section, we describe how issue trackers involve a large social network of stakeholders, where the issue tracker and its issues serve as *boundary objects*, whereby stakeholders in different roles have different views of the underlying issues represented, and leverage the data stored in the issue tracker in different ways [Star and Griesemer 1989].

We discuss stakeholders first. We talked at length with project managers, developers, and those on QA teams about how they use their issue trackers. Although primarily aimed at software developers and quality assurance, we discovered that the issue tracker's social network extends to include a wide range of roles and stakeholders within the broader software ecology, all involved in the process of entering, tracking, and resolving issues.

For example, 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 answer 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 (the step-by-step instructions on how to recreate or "repro" a 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. This is where *boundary objects* come in. 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 [Star and Griesemer 1989]. Several examples of the different ways in which stakeholders viewed the issue tracker as boundary objects are provided below.

Our first example is reflected in the way development team 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.

Our second example includes the collection of people who interacted with the issue tracker that are outside the development team boundary or even outside the company boundary. The sales team, management higher up the organizational hierarchy, and contract workers (often testers) periodically worked with the issue tracking system. Customers external to the organization would interact with the tracker, though usually in an indirect manner such as via automated crash reports or email integration (see below).

Given the wide range of people working with and influencing the contents of the issue tracking system, it becomes clear how an issue-tracking system can act as a boundary object between relatively disparate social worlds [Fitzpatrick, 2003], where the fundamental purpose of (and usage for) the issues stored within can vary widely. Customers are made aware of the issue tracker via the case number automatically appended to their support emails or through the “support ticket” form they filled out after encountering a problem with a piece of software. They view the underlying issue tracker as a means to the resolution of their problem and also as a confirmation of acknowledgement that their concerns are being addressed by the company and not just funnelled into an inbox that is never checked.

Sales and customer support staff view the issue tracker as a tool for assisting them in acquiring new clients and keeping existing clients happy. The issue tracking system provides them both with the detailed client history they need to ensure that existing customers feel well taken care of, and also helps to keep pertinent information readily available so that the customer need not be asked the same questions repeatedly whenever they contact the company. Furthermore, as a sales tool, the issue tracker provides a resource for answering technical inquiries from potential customers in a timely and easily traceable manner.

This multitude of purposes, usages, and understandings of the issue tracking system speaks to the malleability of its identity depending on where a stakeholder lies in proximity to the tracker and with whom such a stakeholder interacts with across the company or development team boundary. The wide range of people interacting with and relying on the issue tracking system on a regular basis also begins to illustrate the broad communicative power it provides.

4.4 Issue Tracker as Communication & Coordination Hub

Given the large role that issue trackers often play in the development process and the number of stakeholders that may be involved in it, it is not surprising that a significant body of communication surrounds the data stored within these systems. Examples include priority discussions during triage meetings, reproduction step clarifications between the person who created an issue and its current owner, fix verification and regression testing, and so on. 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. That is, the issue tracker serves as a communication and coordination hub, where it stores information collected and disseminated not only within the issue tracker, but also via other communication channels (face-to-face meetings, instant messenger contents, email, wikis, etc.).

Tracking all the communication related to issue tracking systems can be challenging, but proves to be crucial in providing the team with 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 study data from all of the teams interviewed. 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 issue tracker also served as both the starting and ending point for various links between development artefacts for our participants. 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. 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.

It is through this interweaving of communication and the centralization of decisions made and available resources that the issue tracker helped senior developers and project managers coordinate their teams:

> *From a team lead perspective, this is where all the communication and list making and giving tasks to people happens.*
>
> —P08

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

4.5 Issue Tracker as Communication Channel

Issue trackers also support direct—albeit sometimes rudimentary—communication via their own facilities. 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 would be appended onto the end of the existing list of comments. In other systems, there was a single "task discussions" field that was edited collectively and repeatedly by the various stakeholders interacting with the bug. Both variations of this practice were often augmented by the ability to attach or associate email messages, screenshots, or other artefacts to an issue.

The comment history or issue discussion (as described in the previous paragraph) 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. This consolidation aspect also resulted in a significant displacement of email usage in favour of the issue tracker (excluding the notification emails described below) for some of our teams:

> *So* [*the issue tracker*] *is really the primary form of communication and that's why I think we don't use email as much. It's pretty consolidated.*
>
> —P07

The communication channel provided by each issue's comment history was often favoured over other channels that were viewed to be more interrupting for less urgent and timely communication. For example, instant messaging systems, while used, were not necessarily preferred due to their propensity to alert, flash, and otherwise grab their users' attention [Nardi et al. 2000]. The issue tracker's more subtle support of back-and-forth communication was often favoured. In this way, the case itself served as a persistent, asynchronous, and oftentimes multicast communication channel.

> *If it's going to actually add any piece of information that is relevant to the case...if that talk's going to add...then I'll usually try to do it in the case.*
>
> —P09

Typically, 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. Of course, each of these messages had a link back to the actual issue, thus integrating the communication with the issue and its related information.

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* [Churchill et al. 2000] that kept relevant communication embedded within the context of the issue to which it referred:

> *Okay, 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, much like the Babble system described by Erickson et al. [1999].

4.6 Issue Tracker as Contextualization Repository

As touched on in the previous sections, we found that the issue tracker also served as a significant source of context for the issues being worked on by our participant teams. Because of how "once a case gets opened, everything that happens to it is tracked in the system" (P09), the issue—and its associated comment and assignment history—provided a single point of contact for understanding the lineage and work history for a given bug, feature, or customer inquiry. This sentiment was expressed by each and every team we interviewed in one way or another. In addition, by having a long-term archive of previously resolved bugs and answered customer questions, the issue tracker became the de facto information source—as well as the first logical destination for most new information—for many on the development team:

> *That's why I use* [*our issue tracker*] *for almost everything. Because then it is all archived and I can easily find stuff in it.*
>
> —P07

Having a complete and readily available history of previous communication also proved to be essential to our participants in creating and maintaining quality relationships with customers over time:

> *This is actually an example of a case where having the complete history of the conversation with this customer is pretty important because they've clearly put a lot of time and effort into trying to talk to us about their ideas and that means ... they've clearly put a lot of value onto the relationship that they've developed with us and so that's a good indicator to us that we should be putting time back into that relationship.*
>
> —P10

In a similar manner, having the detailed history of how a given bug or feature came to be in its current state was key in assisting PMs and lead developers in both the initial triage process as well as the ongoing, iterative prioritization of remaining issues in a project as it progressed throughout its development lifecycle:

> *You can just see a history and figure out what the decision pattern was. So at a triage ... often in a meeting we made a decision, like two months ago, on this looking at the bug and then it comes back to us and we're wondering what's going on and we can see the new discussion pattern and realize that, oh, things are quite different, we need to make a new decision based on this new information. And just having this record is invaluable.*
>
> —P04

In addition to providing relevant context for issues directly through its own fields and comment history, the issue tracker also provided *linked context* in a variety of ways. For example, the issue tracker used by Team C provided the ability to promote a topic from an electronic discussion forum into an issue within the tracker while at the same time preserving its original forum thread:

> *If you created a case from a discussion group, then that case will always have a link back to the original discussion topic so you can see the full context of the discussion.*
>
> —P09

Other forms of context preservation were frequently facilitated via alternate means such as the simple attachment of files or via copy-and-paste mechanisms:

> *We will attach, like the whole email. ...* [*That way*] *they can see the whole history instead of having to explain it all, you know?*
>
> —P12

Analogous to the way various information sources were linked *to* from the issue tracker, so too were there a number of other mediums used by our participants that would link *back* into the issue tracking system and the additional context contained there. For example, as described briefly in Section 4.4, the notification emails generated whenever a case was assigned to an individual served as not only a high-level summary of an underlying issue, but also as an in-bound link to the context and history of that issue stored within the issue tracker. In fact, the importance of reviewing the complete context of an issue prior to beginning new work on it frequently resulted in these notification emails serving as little more than stepping stones back into the rich environment of the issue tracker:

> *The majority of the time, other than I might scroll through and see who did change it, I'll then just use the link to go into* [*our issue tracker*] *directly and to look at it there.*
>
> —P01

This in-bound linking was also commonly used as a quick and unambiguous means for synchronizing the views between team members before asking a question or getting assistance with an open issue:

> *If someone has a problem with something, they should start their sentence in IM with an ID. ...* [*laughs*] *... I'll ping someone with this ID and say, you know, "Hey, do you know what's up with this thing?" So it's like 3 words and they can look it up and get back to me when they know.*
>
> —P04

The patterns described above resulted in the issue tracker serving a context and contextualization repository for the development teams interviewed. Each underlying issue within the tracker compartmentalized and consolidated much of the pertinent information related to it, while also often providing links to additional context either through attachments or hyperlinks to other electronic systems. Having a canonical address to each case within the issue tracking system provided participants with lightweight context handles that could be effortlessly passed among team members in order to quickly synchronize their respective view of an issue before proceeding with detailed, context-specific questions or instructions.

4.7 Summary

In this chapter, I explained how issue trackers are far more than a centralized database of issues being tracked over time. I presented the various roles an issue tracking system takes on within a development team that were revealed by the participants or our research study.

These roles included: repository-type functions for both issue context and organizational knowledge pertaining to the software under development; a communication and coordination role where the issue tracker served as a consolidation point (e.g., a communication and coordination hub); a communication channel in and of itself; and as a means for bridging the disparate worlds of various stakeholders (e.g., as a boundary object). While some of these roles are supported somewhat by the issue tracker, most are a consequence of the way the social network of stakeholders perceive, appropriate, and use the issue tracking system. In later chapters, I will suggest ways that an issue tracker can directly support some of these stakeholders' requirements.

Chapter 5. Differing Views & Incompletely Met Needs

This chapter presents the remainder of the qualitative results from the study described in Chapter 3. Here, I describe the differing views that team members had of various issue tracking facilities, as well as shortcomings and other incompletely met needs of the software as relayed by study participants. I then formulate seven design considerations (summarized in Table 5.1) based upon the above participant frustrations as well as further analysis of our interview data.

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

As discussed in Chapter 4, issue trackers and the issues contained within them form a type of boundary object 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" [Star and Griesemer 1989]. 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. Several of these distinctions are described below.

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 already 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 addition to communicating an idea of the outstanding work remaining, the issue tracker also provided project managers with the fine-grained work items needed to manage their team under the pressure of rapidly approaching deadlines:

> *I mostly go in there on a daily basis to see how many bugs we have—how many highs/blockers/etc.—making sure they're actually getting assigned out to the team.*
>
> —P03

In a similar vein, lead developers also 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 its customers.

In addition to serving as an "outboard brain" for project managers and customer-critical information, the issue tracker also served as a catch-all for potentially important, but easily misplaced or forgotten-over-time details for the development team as a whole:

> *The primary purpose* [*of an issue tracker*] *would be to have a system that you trust that can serve as a way to page out all of the information which you know about a product and you know that you can't remember all at once. ... The issue tracker serves as a way to safely offload little bits of information—little bugs, little feature requests—and know that they're going into a place that we'll be able to recall later and not worry about being forgotten.*
>
> —P09

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. This leads us to our first design consideration.

Design consideration #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.

5.2 Full-fledged Bugs & Almost Bugs

Our participants felt that every bug found in a system should ideally find its way into the issue tracking system, which would serve as a record of its existence and eventual resolution. In reality 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 reproduction 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

And although the practices established at each of our participants' companies were generally ones of entering issues as soon as possible, this was not always the case, and sometimes resulted in a form of "bug filing guilt":

> *So when things go awry like this it's a lot harder to identify. So I might not log the issue until I have something to go on and something to actually work with. Though maybe I should log the issue...I should probably log the issue...that it's been identified and basically took a first look and couldn't.... So I'll take a first look right away and see if there's something obvious, because you don't want to let something like that go because something's wrong, but if after looking at it for the first glance you can't see anything that's the cause, you should probably mark it and at least say that it's happening.*
>
> —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). Participants reported that 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.

It is this blurriness and uncertainty that often surrounds the creation of new bugs that is the premise of our second design consideration.

Design consideration #2:

> The starting point for a bug is not always a well-defined point in time. In particular, an issue's immediate visibility to other team members creates an onus of 'completeness' that may not actually be present; thus the filer may defer or neglect to file an incomplete issue. Providing functionality for stakeholders to enter semi-private, lightweight "pre-bugs" may increase the number of bugs that eventually end up being recorded.

5.3 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, sometimes did not file bugs due to the amount of work required to complete the seemingly never-ending lists of required and optional fields on new issue entry forms. This tension was 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.

The tension between getting the bug in quickly versus having a bug structured in a way that would provide later value to others is typical of many collaborative systems. Grudin [1994] describes this as "the disparity between who does the work and who gets the benefit."

> *A groupware application never provides precisely the same benefit to every group member. Costs and benefits depend on preferences, prior experience, roles, and assignments. A groupware application is expected to provide a collective benefit, but some people must adjust more than others. Ideally, everyone benefits individually, even if some benefit more; however, this ideal is rarely realized. Most groupware requires some people to do additional work to enter or process information that the application requires or produces.*
>
> —Grudin [1994]

This tension often resulted in an ever waxing and waning number of fields associated with cases in the issue tracker, as described by Participant 1 (P01) above. However, a working solution was described via *ad hoc* tagging. In particular, some participants managed to find stopgaps that would give them the sorting and filtering capabilities they needed, without adding additional 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 [Storey et al. 2008] 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 their tracker 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. Grudin [1994] suggests a solution:

> *Demonstrating an application's collective and indirect benefits can help. Reducing the work required of non-beneficiaries seems to be an obvious priority, but it is very difficult to do in practice, because pleasing the principle beneficiary is critically important and the natural focus of attention. One promising approach is to design, along with the technology, processes for using it that create benefits for all group members. This has been stressed in several new meeting management applications. For example, a key element of the process in one is a specific commitment delivered by the meeting convenor to act on the contributions of the participants.*

Grudin attempts to improve the process, as a way to provide all with a reasonable benefit. However, our participants created a technical solution of tagging that reduces one person's work, while still giving reasonable benefit to others, as articulated in the following design consideration.

Design consideration #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. These tags should leverage the issue tracker's existing robust infrastructure for searching, sorting, and filtering issues.

5.4 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 [favourites] 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, the true priority of an issue often 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. The tension around properly prioritizing issues based on its many influencing factors lead us to the following design consideration.

Design consideration #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.

5.5 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, although 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 behaviour was instilled in the development team via their standard practices and procedures over time:

> *Each person has to remember what they're supposed to do on that system because otherwise if they've taken some work and they've done some work and they don't assign it back to me, I don't know that I'm ready to work on it.*
>
> —P02

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 behaviour was also sometimes supported via the issue subscription and email notification functionality described earlier (i.e., small text summaries of actions taken on an issue can be emailed to anyone who "subscribes" to the issue of interest regardless of who that issue is currently assigned to). 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 behaviour 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. Hyper-linking 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.

The variety and nuance in how issue ownership is viewed by the various stakeholders of the tracking system, along with the multiple intimately related facets of issues as defined by their multiple, distinct audiences, prompted us to define the following two design considerations.

Design consideration #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 consideration #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 prove useful in acknowledging the multiple identities of certain issues.

5.6 Privacy, Transparency, and the 'In' Crowd

Just as there is a "vital tension between privacy and visibility" in supporting general communication [Erickson et al. 1999], 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, as well as which portions of those issues should be visible, if any. 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 transparency for customers and maintaining the security of sensitive information often resulted in an entirely closed system with no public presence outside of a bug reporting interface:

> *Our bugs are not generally publicly visible. Like, open-source projects probably wouldn't do it this way, right? They'd probably say, "Oh, that's a bug, so let's open a bug and resume this communication there." Or maybe do it both places. But here, we've got the internal stuff happening in the bug and anything where we need to talk to the person directly, we email them using the* [*email functionality provided directly by the issue tracker*].
>
> —P11

Thus, the last of our design considerations is suggested by the distinctions and nuances between—as well as the content contained within—each of the fully-private, fully-public, and apparent "no-man's-land" areas of the issues stored within the tracker.

Design consideration #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.

5.7 Summary

Addressing the vastly varying needs of the software development team along with other potentially company-external audiences has left issue tracker vendors with a number of significant challenges. 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 need to be presented of this wealth of underlying data held by the issue tracker. 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. And although no current tools currently address all of our design considerations collectively, future tools cannot simply combine them in concert and expect success. Such a system would very likely suffer from unintended side-effects and excessive complexity rather than successfully addressing the needs of its various audiences. For example, some of the considerations suggest adding even more fields to structure the views of an issue tracker, which contradicts the tagging requirement that eschews a multitude of fields. Rather, it is important for system designers to carefully consider these tools from the perspective of each potential audience and actively consider the tradeoffs to be made among the considerations suggested here, as well as other factors and constraints placed upon their product.

This chapter described a number of the differing views held by various members of the development team. Areas examined included the differences in how the issue tracker itself was viewed by its clients (e.g., as a bug list, task list, or to-do list); when an issue comes into being and when this should be reflected in the issue tracker; the contention between many well-compartmentalized fields and the ease with which new cases can be created; the different perceptions and origins of an issue's priority; the various shades of issue ownership and their related embodiments; and finally the tricky problem of balancing the privacy of sensitive issue information with its customer-facing transparency. Through these discussions I suggested a number of design considerations, summarized below in Table 5.1.

The question still remains of how these design considerations can be translated into actual design. In the next chapter, I suggest such designs via interface design sketches that illustrate enhancements to issue tracking systems as well as development and communication tools. I showed these designs to a subset of our participants, and then summarize their feedback.

5.1 A Bug List, a Task List, a To-do List, or a...?   ***Design consideration #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.
5.2 Full-fledged Bugs & Almost Bugs   ***Design consideration #2:*** The starting point for a bug is not always a well-defined point in time. In particular, an issue's immediate visibility to other team members creates an onus of 'completeness' that may not actually be present; thus the filer may defer or neglect to file an incomplete issue. Providing functionality for stakeholders to enter semi-private, lightweight "pre-bugs" may increase the number of bugs that eventually end up being recorded.
5.3 A "Flurry of Fields" vs. Ease of Entry   ***Design consideration #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. These tags should leverage the issue tracker's existing robust infrastructure for searching, sorting, and filtering issues.

5.4 Different Perceptions of Priority ***Design consideration #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.
5.5 Shades of Ownership: Yours, Mine, or Ours? ***Design consideration #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 consideration #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 prove useful in acknowledging the multiple identities of certain issues.
5.6 Privacy, Transparency, and the 'In' Crowd ***Design consideration #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.

Table 5.1 - Summary of design considerations

Chapter 6. Design Sketches

This chapter presents a series of interface design sketches, first introduced in Chapter 3 (Section 3.1.3), that I developed throughout the course of my research study. These interfaces were created in order to directly address the second goal of this thesis: *to design new interfaces as well as enhancements to existing systems with the intent of incorporating better communication and awareness facilities into the issue tracking and overall software development processes*. They do not attempt to address all limitations of current issue tracking systems as described in Chapters 4 and 5. Rather, these three designs stress the incorporation of communication and awareness within issue tracking. In particular, they illustrate possible enhancements to communication tools so they can better interoperate with issue tracking, and to issue tracking systems themselves so that they can better incorporate the communication surrounding the issues they contain. I presented these designs to a subset of my research participants from Team C, discussed their envisaged functionality with them, and solicited their feedback.

6.1 Setting the Scene

The three designs were created as annotated paper-like sketches, where I developed them using the Balsamiq Mockups [Balsamiq Studios 2009] rapid interface prototyping tool. These sketches are not formally designed or well-polished systems. Their scope of implication firmly reflects their presentation as *idea sketches* rather than *well-formed prototypes* [Buxton 2007]. Their intent was to generate conversation and ideas for new or refined tools centred on communication, collaboration, and awareness within primarily small development teams. Some of the designs were derived by modifying or extending existing systems or prototypes previously described in the literature, such as Erickson et al.'s Babble system [Erickson et al. 1999]. Others were inspired by social networking websites such as Facebook.com [Facebook 2009], while still others were based directly on feedback received from participants during our roundtable sessions.

The interface sketches were then provided to selected participants as follows. Each design was explored together at a group roundtable. I presented each interface sketch to the participants, and then verbally explained them. Participants could ask any questions they had concerning the envisioned functionality, which I then answered. I solicited a critique—pros and cons—of the presented designs, as well as ideas for other potential interfaces from my participants. Participant feedback ranged from the technical feasibility of each design as presented, to alternative interaction methods, to suggestions of entirely different communication and/or development tools. Participants often related personal use cases, and suggested use cases for other potential audiences of the envisioned tools. I recorded the roundtable session through field notes taken during the session; it was not audio recorded. The discussions below are based on these field notes.

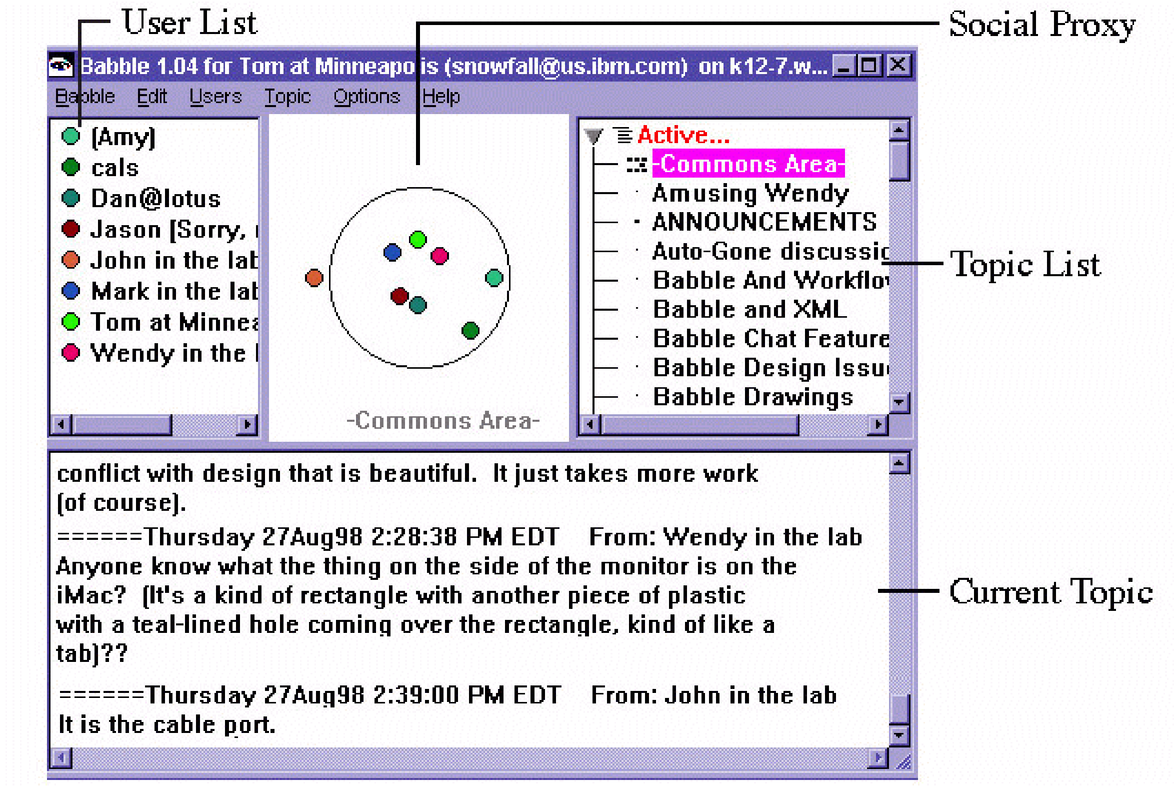
In the following sections, I present each design sketch, describe its functionality, and discuss the feedback received.

6.2 Issue Tracking meets Internet Relay Chat

The first design sketch presented to our participants was largely inspired by Erickson et al.'s Babble interface [Erickson et al. 1999], illustrated in Figure 6.1, which was itself largely inspired by various Internet Relay Chat (IRC) systems [Oikarinen and Reed 1993]. As we will see, Babble is an advanced chat-like system based around topics and users. We extend the Babble concept by considering how it could interact with the information held by an issue tracking system, where issues serve as topics.

I begin by explaining Babble. As seen in Figure 6.1, the Babble interface is comprised of a *user list* (upper left), *social proxy* (top middle), *topic list* (upper right), and *current topic* (bottom) window. The *user list* window lists the names of all the people currently logged into the Babble system. Each user is assigned a coloured "marble" to the left of its label, where this marble is used elsewhere in the interface to signify that user's presence. The *social proxy* window displays a visualization leveraging these marbles. It shows which logged in users are taking part in the topic that is presently open in the interface, where more active users are shown at the centre. That is, if a user does not interact with the Babble interface over time, their marble drifts toward the periphery of the social proxy circle to indicate this lack of activity. This design reflects the idea of *locales* and the *center–periphery relationship* as described in the Locales Framework theory [Fitzpatrick 2003].

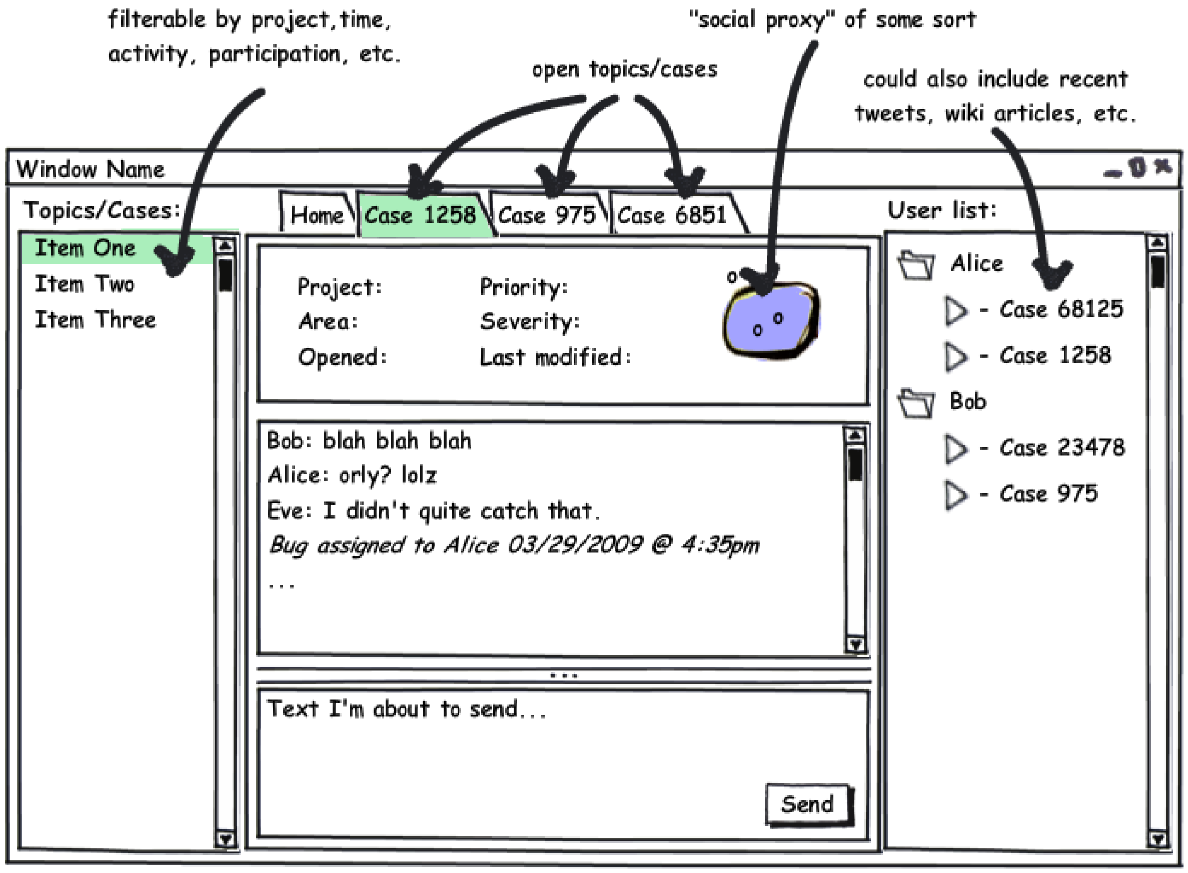
The *topic list* window displays all of the available topics—or in terms of IRC, "channels"—that are available for users to participate in. These topics can be created or deleted by anyone using the Babble system. The selected topic is reflected in the social proxy and current topic window. The *current topic* window displays a persistent, time-stamped log of that topic's conversation. Whenever a users wishes to add something to a given topic, they enter their message in an entry window that then appends their comment (along with a time-stamp) to the end of the topic's conversation log. Users switch between topics simply by clicking on different topics in the topic list window, which then updates the contents of the other windows.



conflict with design that is beautiful. It just takes more work
(of course).
======Thursday 27Aug98 2:28:38 PM EDT From: Wendy in the lab
Anyone know what the thing on the side of the monitor is on the
iMac? (It's a kind of rectangle with another piece of plastic
with a teal-lined hole coming over the rectangle, kind of like a
tab)??
======Thursday 27Aug98 2:39:00 PM EDT From: John in the lab
It is the cable port.

Figure 6.1 – The Babble interface presented by Erickson et al. 1999

The interface sketch shown in Figure 6.2 applies the Babble design toward discussions centred primarily on issues held by an issue tracking system. The user list, topic list, social proxy, and current topic windows are visually similar to that of the Babble system. However, instead of generating topics at will, each topic listed in the envisioned system is tied to an underlying case held by the issue tracking system. Unlike Babble, the proposed system also understands that a person may open and be interested in multiple issue topics at the same time. This is done via a tabbed middle section at the upper part of the main window. Each tab represents an issue topic, where the tab groups a particular topic's social proxy and topic window. Using tabs, users can readily switch between the multiple open issues they are involved with or interested in.



Bob: blah blah blah
Alice: orly? lolz
Eve: I didn't quite catch that.
Bug assigned to Alice 03/29/2009 @ 4:35pm
...

Text I'm about to send...

Figure 6.2 – First design sketch (Babble + IRC)

The upper section of each open tab also contains some basic information pulled directly from the underlying case (e.g., project, priority, severity, last modified date, etc. as seen in Figure 6.2), as well as a social proxy visualization. The bottom section contains a text box that appends a person's typed-in message to the particular topic's conversation log. This conversation log is then stored as the comment history of the underlying issue. In addition to showing the comments entered by users currently active in an issue's conversation, the topic window also displays status messages as other events related to the issue being viewed occur. For example, as shown in the active topic in Figure 6.2, the bug being discussed was assigned to Alice at 4:35pm on March 29th. Similar notifications would also appear for other issue-related events; for example, source code check-ins with an underlying source code management system that reference this issue's case number.

Finally, the user list in our sketch (Figure 6.2, right) displays more information about each user than what is shown in Babble. Each logged-in user's name is expandable to show a list of all the issues that person is currently participating in; that is, all the topic tabs that user currently has open on their computer. This display could be easily augmented to indicate which topic the user is currently viewing (e.g., an asterisk next to their active topic), and other awareness information such as recent documentation changes or code check-ins.

6.2.1 Participant feedback

The goal of the interface design was to provide a real-time conversation mechanism that is tightly integrated with, and grounded in, the issues within the issue tracking system. Participant feedback for this design was positive in regards to this goal, where the idea of incorporating a "chat room" style interaction with the underlying issue was well received. Yet they were concerned about acquiring the necessary critical mass to make such a tool successful and useful for the team. First, participants described critical mass uptake as problematic for almost any new tool being introduced to the team, especially when that tool required one to use an additional application rather than integrating directly with existing tools or interfaces. Second, participants reported that many tools provide the greatest benefit to their users only when there is a sufficiently large portion of their team also using the tool. Thus they saw this as a form of "chicken and egg" problem, where a tool is most useful to an audience only after gaining enough participants, but participants are hesitant to invest in learning or using a new tool without the guarantee of an up-front payoff.

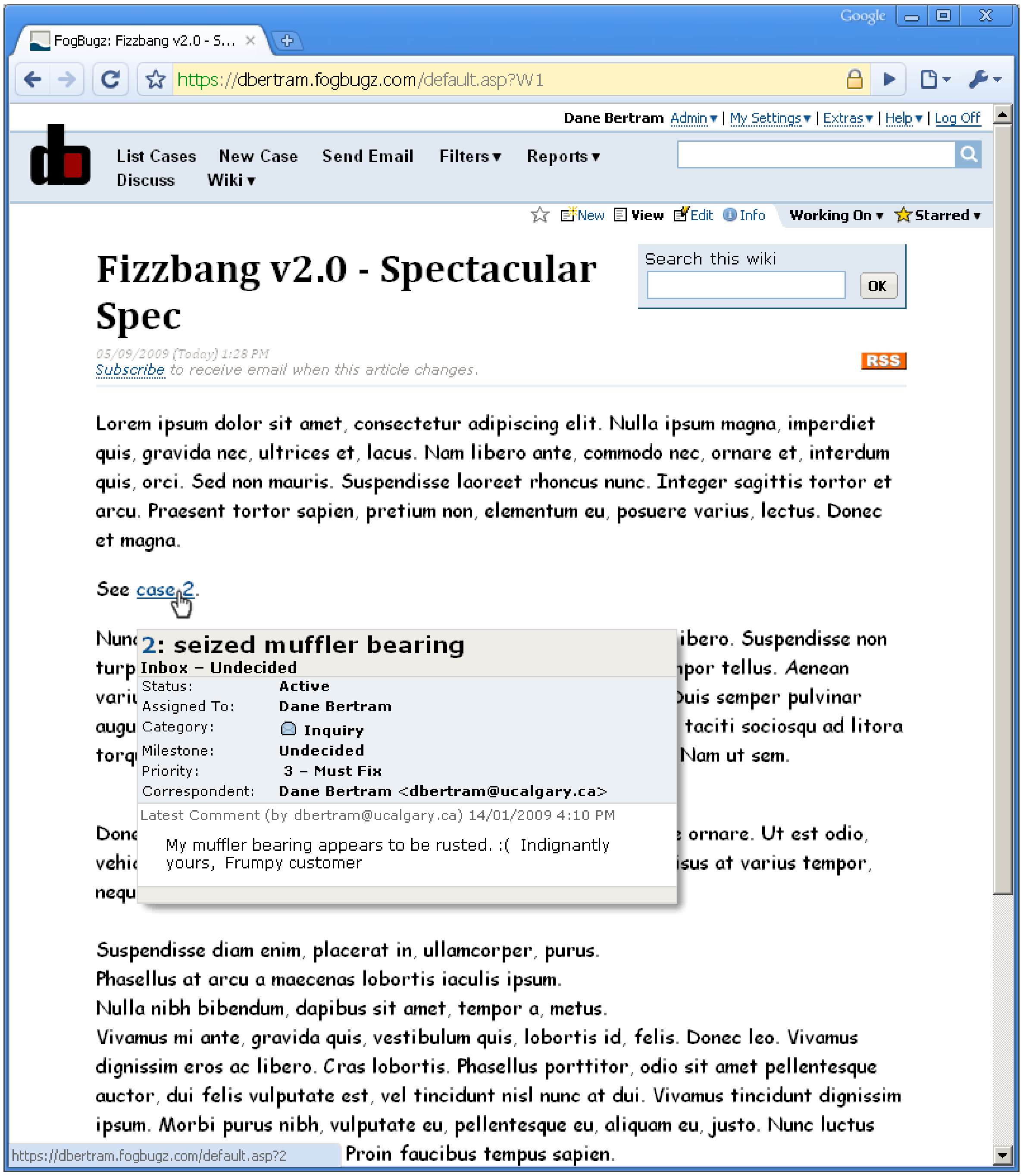
Participants were also hesitant about the viability of the social proxy and user list awareness features. Many of the developers on Team C strongly favoured the "assign the issue back-and-forth and wait for email notifications" communication pattern that occurred around bugs being discussed rather than the more immediate chat or instant messenger style communication pattern. As described in Section 4.5, the asynchronous conversation pattern provided by auto-generated email notifications was viewed to be less interrupting, even if this was at the expense of being less efficient than more synchronous forms of communication. Real-time versus asynchronous awareness and communication is, of course, a trade-off. Sometimes issue discussions would boil down to what could have been accomplished via a brief instant messenger-like conversation that were instead spread out over the course of an entire day due to the latency that often occurred when relying on the email notification pattern. Problems with asynchronous interaction were often exacerbated by the need for participants in such temporally extended conversations to re-familiarize themselves with the context of the case under discussion after each notification, much like what was described in the *contextualization repository* discussion of Section 4.6.

In summary, participants generally acknowledged that a system like the one proposed would likely be useful once established within a development team. However, they generally viewed the barriers to entry to be too high, where they did not see sufficient benefit to motivate changing established communication and working patterns. This does not imply this design would always fail; it could work well with new teams who do not have an existing culture of use, or may succeed by changing an existing culture over time.

6.3 Inline Wiki Conversations

Our second design focused primarily on integrating the capabilities of the wiki component of an existing issue tracker being used by Team C with an anchored, real-time chat system.

I begin by explaining the wiki as provided by Team C's issue tracker, FogBugz [Fog Creek Software 2009a]. This wiki was tightly integrated with the rest of the issue tracking system. For example, hyperlinks could be easily inserted into a wiki document simply by inserting a link and specifying an issue's case number as the URL. This would be automatically translated into a link to the related case. After the link had been created, a small floating preview of the case's details, as depicted in Figure 6.3, would appear whenever a user rested their mouse over the case's hyperlink from the containing wiki page. This integration made it easy for bugs and feature implementation requests to tie themselves back to their respective specifications and also allowed the QA team to keep lists of test cases in the wiki that linked directly to their individual underlying cases.


FogBugz: Fizzbang v2.0 - S...

https://dbertram.fogbugz.com/default.asp?W1

Dane Bertram Admin ▾ | My Settings ▾ | Extras ▾ | Help ▾ | Log Off

List Cases New Case Send Email Filters ▾ Reports ▾
Discuss Wiki ▾

New View Edit Info Working On ▾ Starred ▾

Fizzbang v2.0 - Spectacular Spec

Search this wiki OK

05/09/2009 (Today) 1:28 PM
Subscribe to receive email when this article changes.

RSS

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ipsum magna, imperdiet quis, gravida nec, ultrices et, lacus. Nam libero ante, commodo nec, ornare et, interdum quis, orci. Sed non mauris. Suspendisse laoreet rhoncus nunc. Integer sagittis tortor et arcu. Praesent tortor sapien, pretium non, elementum eu, posuere varius, lectus. Donec et magna.

See case 2.

Nunc ... ibero. Suspendisse non turp ... mpor tellus. Aenean vari ... Duis semper pulvinar augu ... taciti sociosqu ad litora torq ... Nam ut sem.

2: seized muffler bearing
Inbox – Undecided

Status:	Active
Assigned To:	Dane Bertram
Category:	Inquiry
Milestone:	Undecided
Priority:	3 – Must Fix
Correspondent:	Dane Bertram <dbertram@ucalgary.ca>

Latest Comment (by dbertram@ucalgary.ca) 14/01/2009 4:10 PM

My muffler bearing appears to be rusted. :(Indignantly yours, Frumpy customer

Donc ... ornare. Ut est odio, vehic ... isus at varius tempor, nequ

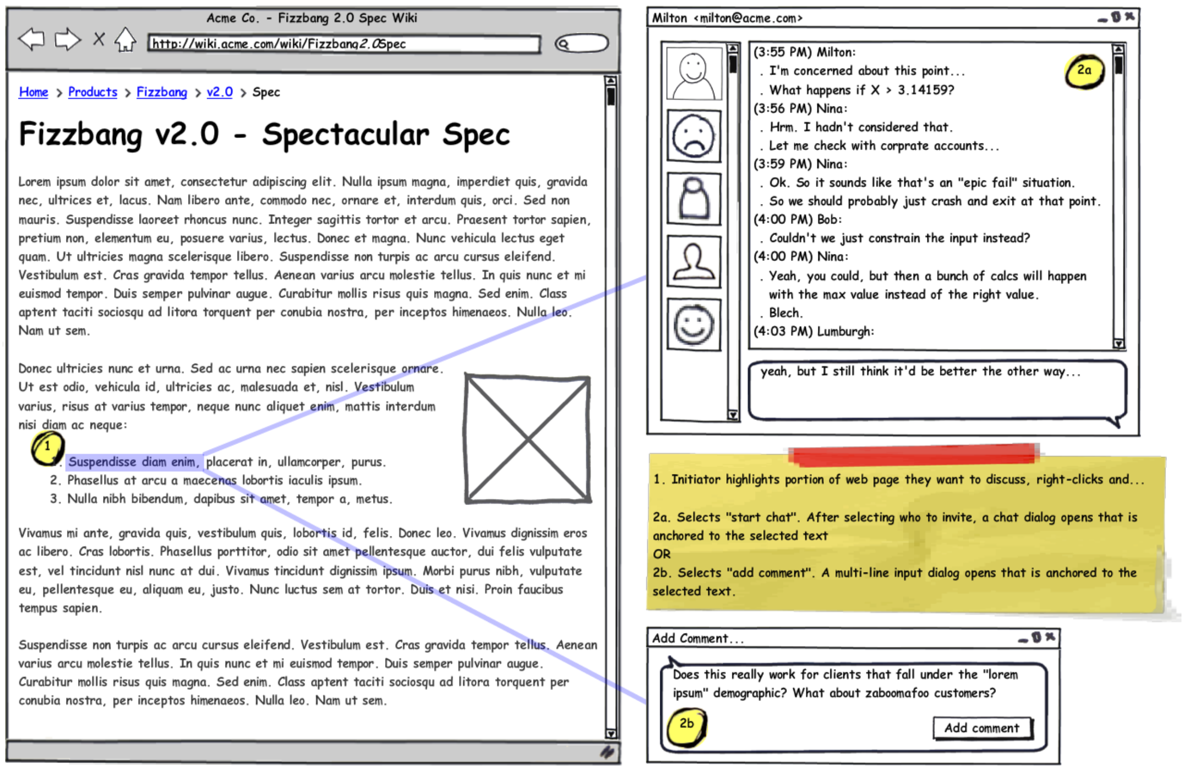
Suspendisse diam enim, placerat in, ullamcorper, purus.
Phasellus at arcu a maecenas lobortis iaculis ipsum.
Nulla nibh bibendum, dapibus sit amet, tempor a, metus.
Vivamus mi ante, gravida quis, vestibulum quis, lobortis id, felis. Donec leo. Vivamus dignissim eros ac libero. Cras lobortis. Phasellus porttitor, odio sit amet pellentesque auctor, dui felis vulputate est, vel tincidunt nisl nunc at dui. Vivamus tincidunt dignissim ipsum. Morbi purus nibh, vulputate eu, pellentesque eu, aliquam eu, justo. Nunc luctus Proin faucibus tempus sapien.

https://dbertram.fogbugz.com/default.asp?2

Figure 6.3 – Case link mouseover preview

Due to this tight integration, Team C had developed a practice of writing most—if not all—of their feature and product specifications within the wiki portion of their issue tracker. After creating the initial version of a feature specification it was a common practice to then solicit feedback from other members of the development team. This was usually done via a weekly status report wiki article that each member of the team contributed a small paragraph to, where they outlined what they had been working on that week. As other members of the team reviewed the specification linked to by this wiki article, they would generally provide feedback by editing the specification's wiki page, adding a heading with their name at the bottom of the article, and then listing questions and concerns in a bulleted list at the bottom of the document. Over time, the original author of the specification (as well as others) would often reply to these comments by prefixing their response with their name or changing the colour of the font used to indicate that a different author was replying. In this way there was a small bulletin board- or discussion group-style conversation happening directly within the wiki article.

This use of the FogBugz wiki, as well as the concept of "sticky conversations" [Churchill et al. 2000], inspired the following design sketch. Figure 6.4 envisions a system and interface that integrates an anchored, real-time chat system directly with the underlying wiki system. That is, the chat conversation concerning an issue is incorporated into the wiki as a sticky conversation. If desired, such chat sessions could remain accessible after the fact as annotations to the underlying wiki article. The persisted portion of the conversation log could be filtered prior to being embedded into the page if needed to remove irrelevant or potentially sensitive information.


Acme Co. - Fizzbang 2.0 Spec Wiki

http://wiki.acme.com/wiki/Fizzbang2.0Spec

Home > Products > Fizzbang > v2.0 > Spec

Fizzbang v2.0 - Spectacular Spec

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ipsum magna, imperdiet quis, gravida nec, ultrices et, lacus. Nam libero ante, commodo nec, ornare et, interdum quis, orci. Sed non mauris. Suspendisse laoreet rhoncus nunc. Integer sagittis tortor et arcu. Praesent tortor sapien, pretium non, elementum eu, posuere varius, lectus. Donec et magna. Nunc vehicula lectus eget quam. Ut ultricies magna scelerisque libero. Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Donec ultricies nunc et urna. Sed ac urna nec sapien scelerisque ornare. Ut est odio, vehicula id, ultricies ac, malesuada et, nisl. Vestibulum varius, risus at varius tempor, neque nunc aliquet enim, mattis interdum nisi diam ac neque:

1. Suspendisse diam enim, placerat in, ullamcorper, purus.
2. Phasellus at arcu a maecenas lobortis iaculis ipsum.
3. Nulla nibh bibendum, dapibus sit amet, tempor a, metus.

Vivamus mi ante, gravida quis, vestibulum quis, lobortis id, felis. Donec leo. Vivamus dignissim eros ac libero. Cras lobortis. Phasellus porttitor, odio sit amet pellentesque auctor, dui felis vulputate est, vel tincidunt nisl nunc at dui. Vivamus tincidunt dignissim ipsum. Morbi purus nibh, vulputate eu, pellentesque eu, aliquam eu, justo. Nunc luctus sem at tortor. Duis et nisi. Proin faucibus tempus sapien.

Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Milton <milton@acme.com>

(3:55 PM) Milton:
. I'm concerned about this point...
. What happens if X > 3.14159?
(3:56 PM) Nina:
. Hrm. I hadn't considered that.
. Let me check with corprate accounts...
(3:59 PM) Nina:
. Ok. So it sounds like that's an "epic fail" situation.
. So we should probably just crash and exit at that point.
(4:00 PM) Bob:
. Couldn't we just constrain the input instead?
(4:00 PM) Nina:
. Yeah, you could, but then a bunch of calcs will happen with the max value instead of the right value.
. Blech.
(4:03 PM) Lumburgh:

yeah, but I still think it'd be better the other way...

1. Initiator highlights portion of web page they want to discuss, right-clicks and...

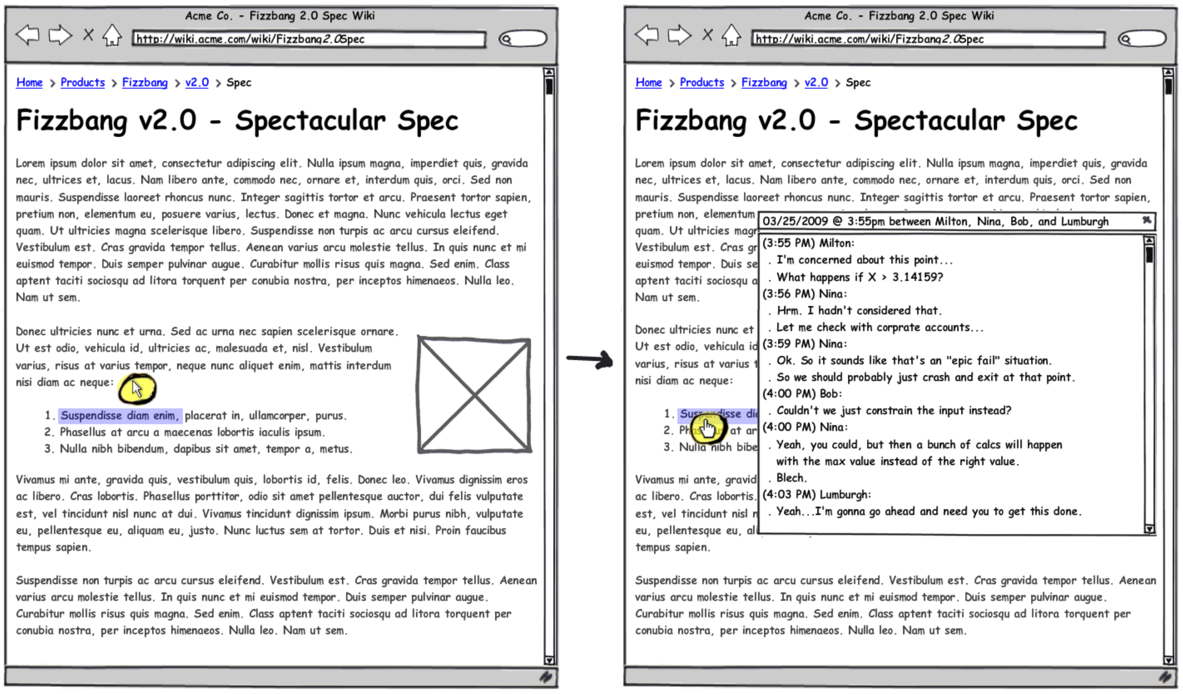
2a. Selects "start chat". After selecting who to invite, a chat dialog opens that is anchored to the selected text
OR
2b. Selects "add comment". A multi-line input dialog opens that is anchored to the selected text.

Add Comment...

Does this really work for clients that fall under the "lorem ipsum" demographic? What about zaboomafoo customers?

Figure 6.4 – Second design sketch (inline wiki conversations)

The interaction with the proposed system begins by simply highlighting the portion of the wiki article that the user wishes to discuss. After right-clicking on the selected content they could then decide to either initiate a chat session with others or to simply add their own comment in isolation. If they chose to start a chat session they would be prompted to select from a list of potential participants (i.e. users of the underlying issue tracking system or wiki system). After the list of participants had been selected a small floating chat dialog would open (Figure 6.4 upper-right) where the highlighted content could be discussed in real-time. If the user instead wished to simply add their own comment without discussion, they would be presented with a multi-line input box to enter their comment into (Figure 6.4 lower-right). After the chat session had finished or the user had entered their standalone comment, the selected page content would be augmented to indicate an associated annotation. In both cases, initiating and anchoring a conversation or comment over a wiki segment is easy and lightweight. Figure 6.5 shows a sample of what the resultant annotation and mouseover effect might look like.


Acme Co. - Fizzbang 2.0 Spec Wiki

http://wiki.acme.com/wiki/Fizzbang2.0Spec

Home > Products > Fizzbang > v2.0 > Spec

Fizzbang v2.0 - Spectacular Spec

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ipsum magna, imperdiet quis, gravida nec, ultrices et, lacus. Nam libero ante, commodo nec, ornare et, interdum quis, orci. Sed non mauris. Suspendisse laoreet rhoncus nunc. Integer sagittis tortor et arcu. Praesent tortor sapien, pretium non, elementum eu, posuere varius, lectus. Donec et magna. Nunc vehicula lectus eget quam. Ut ultricies magna scelerisque libero. Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Donec ultricies nunc et urna. Sed ac urna nec sapien scelerisque ornare. Ut est odio, vehicula id, ultricies ac, malesuada et, nisl. Vestibulum varius, risus at varius tempor, neque nunc aliquet enim, mattis interdum nisi diam ac neque:

1. Suspendisse diam enim, placerat in, ullamcorper, purus.
2. Phasellus at arcu a maecenas lobortis iaculis ipsum.
3. Nulla nibh bibendum, dapibus sit amet, tempor a, metus.

Vivamus mi ante, gravida quis, vestibulum quis, lobortis id, felis. Donec leo. Vivamus dignissim eros ac libero. Cras lobortis. Phasellus porttitor, odio sit amet pellentesque auctor, dui felis vulputate est, vel tincidunt nisl nunc at dui. Vivamus tincidunt dignissim ipsum. Morbi purus nibh, vulputate eu, pellentesque eu, aliquam eu, justo. Nunc luctus sem at tortor. Duis et nisi. Proin faucibus tempus sapien.

Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

03/25/2009 @ 3:55pm between Milton, Nina, Bob, and Lumburgh

(3:55 PM) Milton:
. I'm concerned about this point...
. What happens if X > 3.14159?
(3:56 PM) Nina:
. Hrm. I hadn't considered that.
. Let me check with corprate accounts...
(3:59 PM) Nina:
. Ok. So it sounds like that's an "epic fail" situation.
. So we should probably just crash and exit at that point.
(4:00 PM) Bob:
. Couldn't we just constrain the input instead?
(4:00 PM) Nina:
. Yeah, you could, but then a bunch of calcs will happen with the max value instead of the right value.
. Blech.
(4:03 PM) Lumburgh:
. Yeah...I'm gonna go ahead and need you to get this done.

Figure 6.5 – Annotation mouseover behaviour of second design sketch

6.3.1 Participant feedback

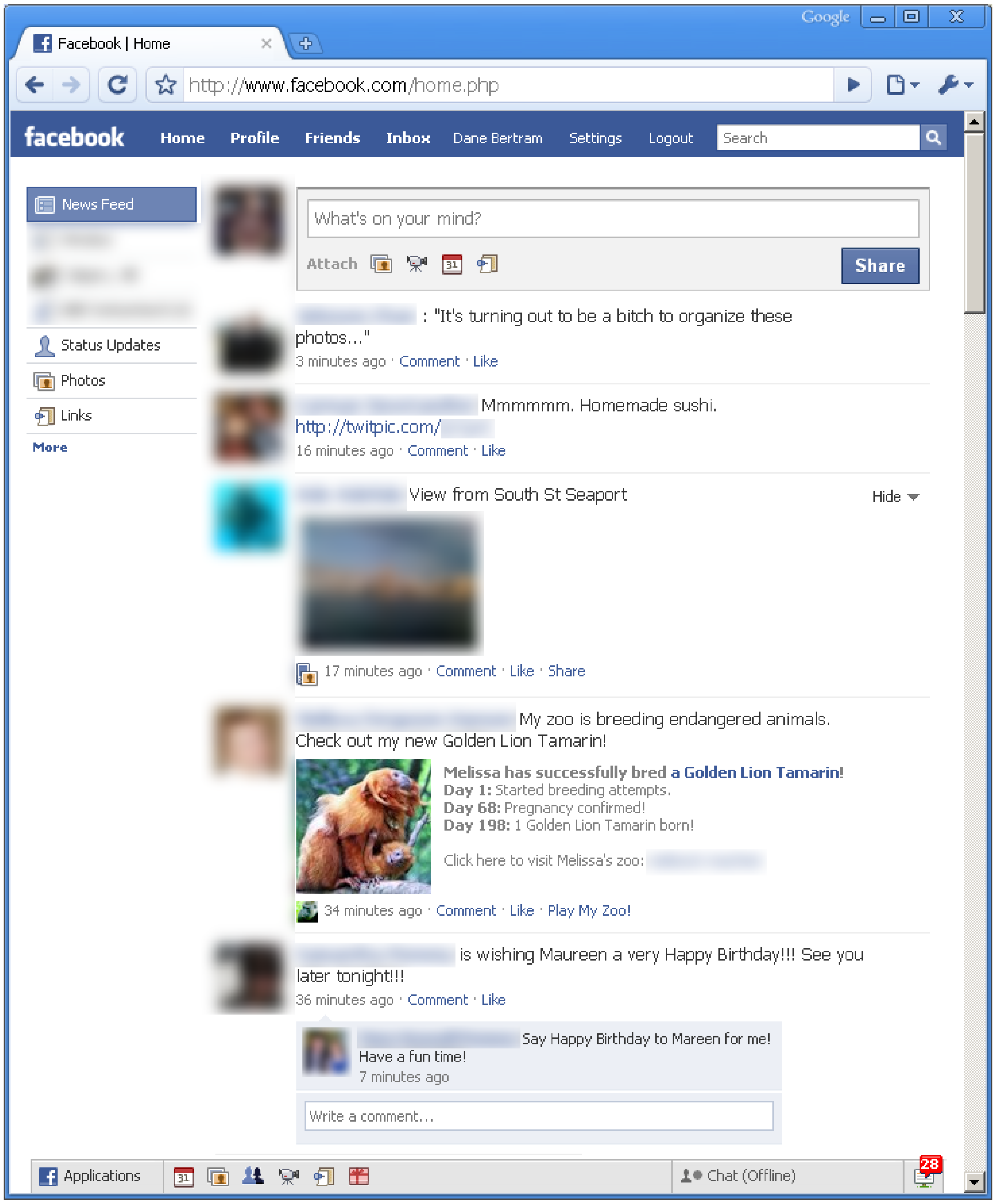
Most participant feedback for this design sketch centred on the technical feasibility and the implementation difficulties that would need to be considered for such a system. One of the participants had recently worked on a similar system that needed to keep track of precise selection ranges within wiki articles, much like those that would be used in the proposed system. He suggested that such a fine level of granularity would be very difficult to implement correctly and consistently across the many popular browser and operating system combinations likely to be in use.

This participant also suggested that most users of their system did not actually need or even desire such a fine level of granularity to begin with. Instead, he suggested that the granularity could be limited to the lowest block-level items within the wiki article's Document Object Model (DOM) without losing the desired locality or specificity for anchoring the resultant annotations. For example, rather than allowing a user to select only a few characters or words as suggested in Figure 6.4 and Figure 6.5, the selection process could be made less granular and be limited to, say, the entire sentence that makes up bullet point number one in the same figures. He also suggested that a system with a coarser-grained selection mechanism would still anchor the resultant annotation to a specific-enough subsection of the surrounding document to provide sufficient context for the ensuing chat dialog or comment.

In summary, our participants—who already had a culture of using wikis—were positive about the design. Concerns tended to be centered on implementation practicality issues, and interface refinement in terms of selecting block-level attributes of wiki text rather than arbitrary regions for anchoring annotations.

6.4 Recent Activity Feed

Our final design uses various activity feeds to supply team awareness by reviewing recent personal and co-worker activity over time. This interface was largely inspired by the activity feed found on the popular social networking site, Facebook.com (see Figure 6.6) [Facebook 2009]. Facebook's activity feed provides a reverse-chronological, time-stamped list of various tidbits of information pertaining to people you've identified as your friends. These tidbits range from brief messages written on you and your friend's publicly-visible "walls," to thumbnails of photo albums that have been uploaded and shared, to status messages posted just to let others know what they're thinking or have been up to recently. The activity feed updates continuously and, over time, can provide a centralized view of "what's happening" within your social network.


Facebook | Home

http://www.facebook.com/home.php

[redacted] : "It's turning out to be a bitch to organize these photos..."
3 minutes ago · Comment · Like

[redacted] Mmmmmm. Homemade sushi. http://twitpic.com/[redacted]
16 minutes ago · Comment · Like

[redacted] View from South St Seaport
Hide
17 minutes ago · Comment · Like · Share

[redacted] My zoo is breeding endangered animals. Check out my new Golden Lion Tamarin!
Melissa has successfully bred a Golden Lion Tamarin!
Day 1: Started breeding attempts.
Day 68: Pregnancy confirmed!
Day 198: 1 Golden Lion Tamarin born!
Click here to visit Melissa's zoo: [redacted]
34 minutes ago · Comment · Like · Play My Zoo!

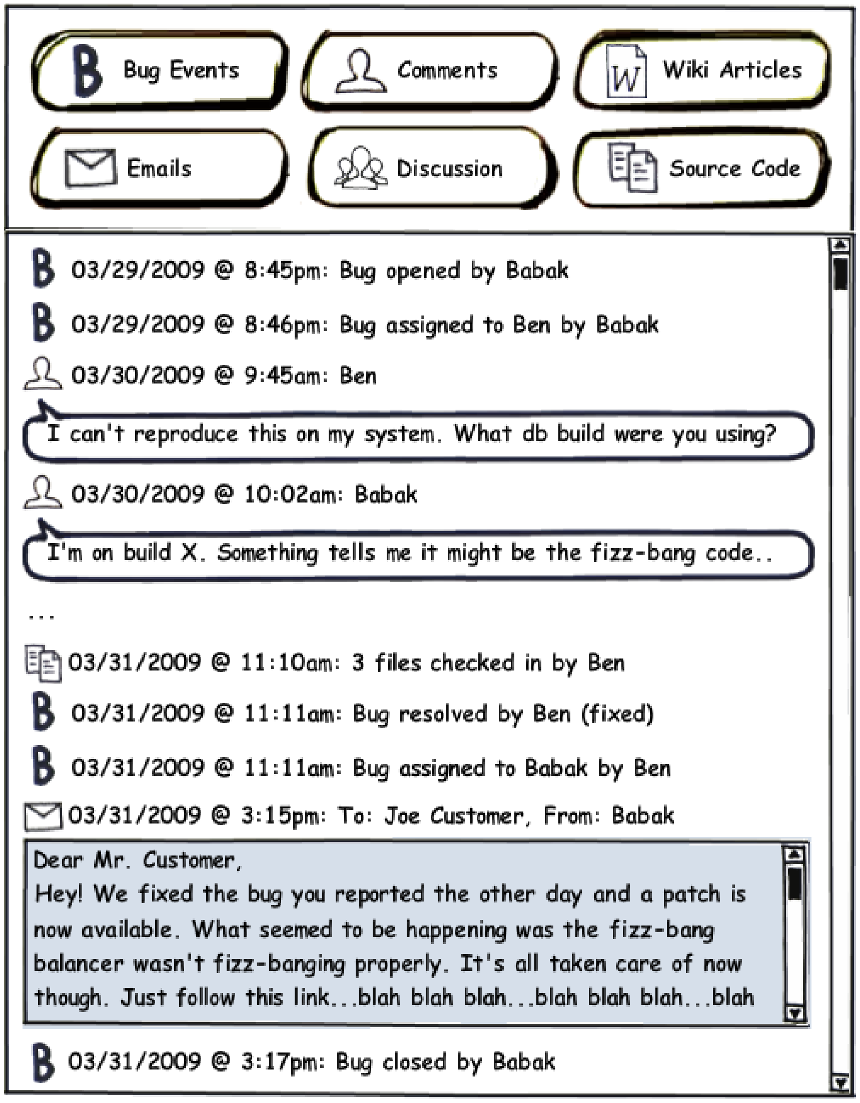
[redacted] is wishing Maureen a very Happy Birthday!!! See you later tonight!!!
36 minutes ago · Comment · Like

[redacted] Say Happy Birthday to Mareen for me! Have a fun time!
7 minutes ago

Write a comment...

Figure 6.6 – Facebook.com recent activity feed

Our final design, illustrated in Figure 6.7, extends the Facebook.com activity feed concept by integrating activity feeds across systems already in use by the team, such as their issue tracker, wiki, discussion board, and/or source code management system. The idea is that as events occur in each of these respective systems, short summaries would appear in the proposed activity feed. Each event would be time-stamped, identify who generated or caused the event, and would include a link back to the underlying system to allow an interested person to obtain more information.



03/29/2009 @ 8:45pm: Bug opened by Babak
03/29/2009 @ 8:46pm: Bug assigned to Ben by Babak
03/30/2009 @ 9:45am: Ben
I can't reproduce this on my system. What db build were you using?
03/30/2009 @ 10:02am: Babak
I'm on build X. Something tells me it might be the fizz-bang code..
...
03/31/2009 @ 11:10am: 3 files checked in by Ben
03/31/2009 @ 11:11am: Bug resolved by Ben (fixed)
03/31/2009 @ 11:11am: Bug assigned to Babak by Ben
03/31/2009 @ 3:15pm: To: Joe Customer, From: Babak
Dear Mr. Customer,
Hey! We fixed the bug you reported the other day and a patch is now available. What seemed to be happening was the fizz-bang balancer wasn't fizz-banging properly. It's all taken care of now though. Just follow this link...blah blah blah...blah blah blah...blah
03/31/2009 @ 3:17pm: Bug closed by Babak

Figure 6.7 – Third design sketch (recent activity feed)

As with Facebook.com activity feeds, this developer activity feed would be filterable based on the types of events of interest. The various categories for events would take the form of toggle buttons like those depicted at the top of Figure 6.7. Some events could also be expandable inline to provide additional information without requiring users to navigate away from the activity feed and into a separate application. For example, the email event shown near the bottom of Figure 6.7 could be displayed initially as a single line summarizing to whom the email was sent and by whom. Next to this line would be a small plus sign that, when clicked, would expand to display a scrollable panel including the body of the email message. Similar drill-down operations could also be incorporated into the other types of events displayed in the activity feed.

Much like the previous design, the activity feed interface was not envisioned to serve as a standalone application, although such a standalone activity feed client would be a possibility. Ideally, this feed would be embedded within an existing tool already frequently used by members of the development team throughout the day. For example, the proposed activity feed could be inserted as a sidebar on the user's homepage or the default homepage of their issue tracking system. In this way, the activity feed would serve as a flexible, centralized portal into the various underlying systems it drew events from.

6.4.1 Participant feedback

Of all the design sketches presented, this interface received the most enthusiastically positive feedback. Participants were visibly excited about having such an interface, and many expressed that they had previously thought about how useful such an awareness system would be for them personally. The proposed system was also considered to be fairly straightforward to implement due to it serving primarily as a data sink for existing tools already in use. Many of these underlying systems (issue trackers, source code repositories, wiki publishing systems, etc.) already provide public APIs for accessing the types of events that would be used by the proposed system, and those that don't are often backed by a relational database that can be easily queried by external tools (if given sufficient access to the database). The participants involved in this roundtable agreed that this design provided the best "bang for the buck" from the perspective of the development time required and the overall value gained by the resulting system.

Participants did, however, express concern about the potential information overload that could result from such a system. Many of the events shown in the design sketch were very fine-grained (e.g., events for the opening of individual bugs) and could result in a flood of entries if used with even a moderately large development team. To help deal with this, participants suggested tabbing the interface into a number of various scopes. Each tab in the interface would roughly duplicate the interface presented above, but would be scoped to include only a subset of all the available events. These tabs would comprise scopes ranging from one that displayed only the events pertaining to "me," to all the events for a single project area, to an entire product, and all the way up to a global feed showing events being generated by all the members of the various development teams within an organization. They also suggested filtering out small events or grouping similar small events together in order to cut down on the overall quantity of event entries that would need to be displayed. For example, rather than showing a separate event entry for each of six cases a co-worker opened in rapid succession, instead show a single entry summarizing this (i.e., "Bob opened 6 new cases") and provide the details of each case in the expanded detail view.

6.5 Summary

Overall, participant feedback raises several points relevant to these and other designs.

First, people generally thought that designs that extended, modified, or integrated with existing systems were better than designs of entirely new applications. Their concerns were two-fold: new applications could disrupt existing communication and/or working habits and toolsets of the team, and would inhibit critical mass. The "critical mass" problem is not a new one, and remains a primary obstacle in getting new tools off the ground and adopted by development teams. Fortunately, today's software systems are increasingly being designed with integration and interoperation in mind rather than being totally proprietary and closed off to the outside world. Many now make use of open data storage and exchange formats, provide plug-in architectures, or expose public data access APIs.

Conveniently, most popular issue tracking systems today are also either entirely web-based or provide a web interface to their underlying database. Indeed, all of the teams interviewed during our study used completely web-based issue tracking systems. Through the advent of systems like Greasemonkey[1] [Greasemonkey 2009], which allow users to readily customize both the appearance and (to a certain degree) the functionality of existing web applications, such integrative prototypes could potentially be rapidly layered over top of existing development tools. I posit that integrating new functionality with existing tools can help avoid the adoption inertia that frequently prevents a promising tool from taking hold within an existing development team's arsenal. However, this does not mean that totally new interfaces are impractical, for they may be adopted by new teams or even by niches within existing teams as a disruptive technology.

Second, our participants all showed interest in having the issue tracker provide a more real-time communication component for use in issue discussion, such as the "issue tracking meets IRC" design sketch presented above (Section 6.2). However, they did not wish to be forced into using a system requiring their constant presence in a chat room-like environment. This suggests that new designs should incorporate both real-time and asynchronous mechanisms, where one should not obviate the other.

Third, being able to discuss an issue or feature specification at its logical source—the issue or specification document itself—appealed to all interviewed participants. An example is our idea of an "inline wiki conversations" interface Figure 6.4 (Section 6.3). The cautionary note here is technical: we may not be able to integrate such a feature within existing systems, or do so in a way that could easily outweigh the implementation cost incurred.

Fourth, participants championed the need for awareness among team members. In particular, the idea of having a "recent activity feed" that would bring together activity notifications from various development tools already in use, while at the same time being

[1] Greasemonkey, and other systems like it, allow users to install customized scripts (JavaScript in the case of Greasemonkey) in their browser that make on-the-fly changes to web page content entirely on the client (i.e., without any explicit involvement of the server generating the content). Each time a web page with an associated script is loaded, that script is executed on the client immediately after the page has finished loading in the browser. Since these scripts are both site-specific and persistent, the changes made by these scripts are effectively permanent for the user running the script. Changes can include modifying the appearance of the underlying web page, modifying or extending its functionality, or combining one site's data or functionality with content provided by another (such hybrid web sites are often called "mashups").

integrated within an existing web-based interface, was very well received. The awareness and development details presented by such a system were seen as striking an ideal balance between the development investment required and the overall functionality gained, provided that the potential flood of activity events could be sufficiently mitigated through either filtering or the collapsing of multiple similar events into shorter, summarizing ones.

This chapter has only touched upon a few changes to issue tracking systems and their companions, and in particular, how communication and awareness can be integrated as part of these systems. In the previous chapters, I described a variety of other factors critical to issue tracking, and each of those could also be used to generate alternate designs. While it is beyond the scope of this thesis to do so, I believe that the design of future issue tracking systems should consider these factors collectively.

Chapter 7. Conclusions, Contributions, & Future Work

In this chapter, I present a discussion of the conclusions drawn from this thesis, a summary of its primary and secondary contributions, as well as a brief discussion of the potential future directions this research could be taken in.

7.1 Discussion & Conclusions

One of our participants, P14, has 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. A tool traditionally viewed as a relatively straightforward engineering tool has revealed itself to be a complex facilitator of communication and coordination within what is a *fundamentally social process*: developing software. Even within small teams that enjoy the benefit of frequent, readily available face-to-face communication, still the issue tracker remains a core component in the successful cooperation of the team's members.

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 need to be presented of this wealth of underlying data held within the issue tracker. 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. And although no existing tools currently address all of our design considerations collectively, future tools cannot simply combine them in concert and expect success. Such a system would very likely suffer from unintended side-effects and excessive complexity rather than succeed in addressing the needs of its various audiences. Rather, it is important for system designers to carefully consider their tools from the perspective of each potential audience and actively consider the tradeoffs to be made among them.

From a CSCW perspective, issue tracking systems now play a significant supporting—and sometimes pivotal—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.

7.2 Contributions

At the beginning of this thesis, I identified the following problem:

> *We do not know how small, collocated teams make use of issue tracking systems in real-world, industrial settings from the perspective of supporting communication, collaboration, and coordination both inside and outside the team.*

With this research I set out to address that problem through the following two goals:

1. *Investigate how small, collocated teams use issue tracking systems in day-to-day software development tasks for the purpose of communication, collaboration, and coordination both inside and outside the immediate development team.*
2. *Attempt to design new interfaces as well as enhancements to existing systems with the intent of incorporating better communication and awareness facilities into the issue tracking and overall software development processes.*

Through the course of an in-depth qualitative study of 15 participants spread across three primary roles and four organizations as well as the later analysis of that study and the feedback received on three interface design sketches, two primary and three secondary contributions have been made:

Primary contributions:

1. A detailed exploration of the communication, coordination, and collaboration patterns, practices, and tools of small, collocated teams as they relate to issue tracking systems.
2. The articulation of five roles issue tracking systems play within these teams:
    i. The issue tracker as a repository for organizational knowledge and facilitator of distributed cognition within the development team
    ii. The issue tracker and its issues as boundary objects that bridge a number of diverse and distinct stakeholder worlds

iii. The issue tracker as a communication and coordination hub that consolidates a variety of interconnected communication channels within the development process

iv. The issue tracker as a persistent, asynchronous, and oftentimes multicast communication channel in and of itself

v. The issue tracker as a contextualization repository providing a canonical address for all things related to each of the issues contained within

Secondary contributions:

1. The description of six areas where stakeholder perceptions vary in relation to various aspects of the issue tracker:

    i. Various types of lists as seen by different roles leveraging the issue tracker

    ii. Degrees of issue existence as described by those filing and fixing bugs

    iii. Contention over field congestion and workarounds that side-step this problem

    iv. Differing multi-faceted opinions of what comprises an issue's priority

    v. Shades of issue ownership and related monitoring patterns

    vi. Tension between transparency, privacy, and the internal-external wall

and as a direct result of these differing perceptions,

2. The identification of seven considerations for the design of issue tracking and software development coordination tools:

    i. 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.

ii. 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.

iii. 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.

iv. 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.

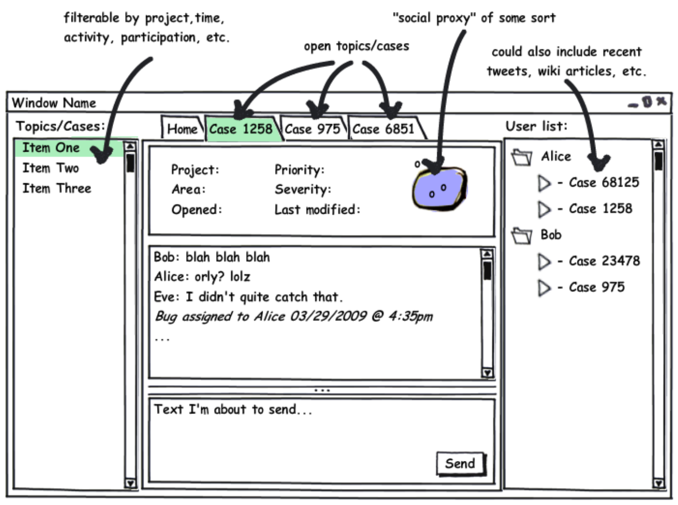
v. Multiple levels of issue ownership exist throughout the software development team. By supporting this pattern in the issue tracker itself, stakeholders can eliminate their need to manually monitor issues that they still partially own.

vi. 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.

vii. 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.

3. The presentation and critique of three prototype systems in the form of early design sketches:

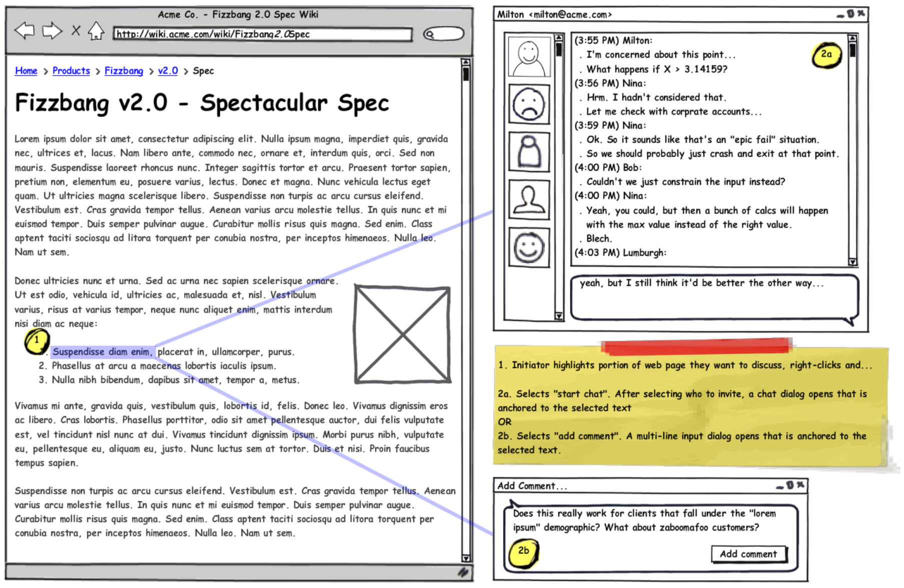
   i. Issue tracking meets Internet Relay Chat



Bob: blah blah blah
Alice: orly? lolz
Eve: I didn't quite catch that.
Bug assigned to Alice 03/29/2009 @ 4:35pm
...

Text I'm about to send...

ii. Inline wiki conversations


Acme Co. - Fizzbang 2.0 Spec Wiki
http://wiki.acme.com/wiki/Fizzbang2.0Spec
Home > Products > Fizzbang > v2.0 > Spec
1
2a
2b
Add comment

Fizzbang v2.0 - Spectacular Spec

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla ipsum magna, imperdiet quis, gravida nec, ultrices et, lacus. Nam libero ante, commodo nec, ornare et, interdum quis, orci. Sed non mauris. Suspendisse laoreet rhoncus nunc. Integer sagittis tortor et arcu. Praesent tortor sapien, pretium non, elementum eu, posuere varius, lectus. Donec et magna. Nunc vehicula lectus eget quam. Ut ultricies magna scelerisque libero. Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Donec ultricies nunc et urna. Sed ac urna nec sapien scelerisque ornare. Ut est odio, vehicula id, ultricies ac, malesuada et, nisl. Vestibulum varius, risus at varius tempor, neque nunc aliquet enim, mattis interdum nisi diam ac neque:

1. Suspendisse diam enim, placerat in, ullamcorper, purus.
2. Phasellus at arcu a maecenas lobortis iaculis ipsum.
3. Nulla nibh bibendum, dapibus sit amet, tempor a, metus.

Vivamus mi ante, gravida quis, vestibulum quis, lobortis id, felis. Donec leo. Vivamus dignissim eros ac libero. Cras lobortis. Phasellus porttitor, odio sit amet pellentesque auctor, dui felis vulputate est, vel tincidunt nisl nunc at dui. Vivamus tincidunt dignissim ipsum. Morbi purus nibh, vulputate eu, pellentesque eu, aliquam eu, justo. Nunc luctus sem at tortor. Duis et nisi. Proin faucibus tempus sapien.

Suspendisse non turpis ac arcu cursus eleifend. Vestibulum est. Cras gravida tempor tellus. Aenean varius arcu molestie tellus. In quis nunc et mi euismod tempor. Duis semper pulvinar augue. Curabitur mollis risus quis magna. Sed enim. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Nulla leo. Nam ut sem.

Milton <milton@acme.com>

(3:55 PM) Milton:
. I'm concerned about this point...
. What happens if X > 3.14159?
(3:56 PM) Nina:
. Hrm. I hadn't considered that.
. Let me check with corprate accounts...
(3:59 PM) Nina:
. Ok. So it sounds like that's an "epic fail" situation.
. So we should probably just crash and exit at that point.
(4:00 PM) Bob:
. Couldn't we just constrain the input instead?
(4:00 PM) Nina:
. Yeah, you could, but then a bunch of calcs will happen with the max value instead of the right value.
. Blech.
(4:03 PM) Lumburgh:

yeah, but I still think it'd be better the other way...

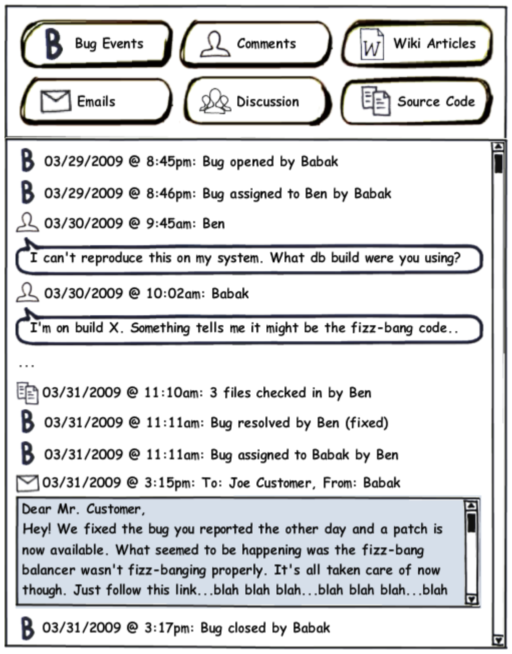
1. Initiator highlights portion of web page they want to discuss, right-clicks and...

2a. Selects "start chat". After selecting who to invite, a chat dialog opens that is anchored to the selected text
OR
2b. Selects "add comment". A multi-line input dialog opens that is anchored to the selected text.

Add Comment...

Does this really work for clients that fall under the "lorem ipsum" demographic? What about zaboomafoo customers?

iii. Recent development activity feed



03/29/2009 @ 8:45pm: Bug opened by Babak
03/29/2009 @ 8:46pm: Bug assigned to Ben by Babak
03/30/2009 @ 9:45am: Ben
I can't reproduce this on my system. What db build were you using?
03/30/2009 @ 10:02am: Babak
I'm on build X. Something tells me it might be the fizz-bang code..
...
03/31/2009 @ 11:10am: 3 files checked in by Ben
03/31/2009 @ 11:11am: Bug resolved by Ben (fixed)
03/31/2009 @ 11:11am: Bug assigned to Babak by Ben
03/31/2009 @ 3:15pm: To: Joe Customer, From: Babak

Dear Mr. Customer,
Hey! We fixed the bug you reported the other day and a patch is now available. What seemed to be happening was the fizz-bang balancer wasn't fizz-banging properly. It's all taken care of now though. Just follow this link...blah blah blah...blah blah blah...blah

03/31/2009 @ 3:17pm: Bug closed by Babak

7.3 Future Work

If we look at the original design of the research study and its resultant findings that were presented in this thesis, we see a study that aims primarily for depth rather than breadth in its attempt to gain a better understanding of how issue tracking fits into the picture of small, collocated development teams. Only three primary roles—developers, quality assurance personnel, and project managers—were sampled for our study. One large area for potential future work, then, would be to broaden that sampling to include some of the many other stakeholders identified as playing a role—sometimes pivotal—to the development process within these teams. Such stakeholders could include customer service and support personnel, customers of the systems being developed that are external to the company, higher-ups within the management hierarchy, or even members of the sales staff within the organizations studied.

In an alternate dimension, the study presented here examined only *established* teams that had well-defined work practices and routines that had been custom-tailored over time to fit snugly alongside the toolsets these teams had become familiar with. If we were to explore further back along this axis, the questions asked in this study could also be applied during earlier stages of development team formation. This could shed light onto the process of how development teams appropriate issue tracking systems and morph them into the roles of providing communication and collaboration support to their various audiences over time. Additional insight could also be gained by finding out not only *how* a development team's practices came into being, but also attempting to identify *why* such practices arose. Do development teams that are comprised primarily of recent graduates tend toward a specific set of tools and practices versus teams comprised of more experienced developers? Or does a development teams' toolset develop organically based on the immediate problems and frustrations dealt with by the team on a day-to-day basis?

Of the four teams studied, only three major brands of issue tracking systems were observed (ExtraView, FogBugz, and SharePoint). There are easily dozens of commercial and open-source issue tracking systems on the market today, and their nuances likely lead to a wide variety of work practices. Examining all these tools would likely be prohibitive for any individual study, but recruiting participant teams with the intent of broadening the cross-section of issue trackers sampled would likely reveal a number of interesting comparison points. An alternate direction might instead select for teams using the same tools, but of different sizes or of a different experience make-up in order to tease out why different types of teams might use the same tools in interesting or surprisingly different ways. Regardless of the specific target set, by taking a broader look at a larger set of development teams and issue tracking systems, we could begin to understand some of the many different ways that coordination manifests itself with regards to issue tracking systems—and coordination systems more generally—within small development teams. By broadening the number and variety of teams examined, we would then be better equipped to begin looking for patterns and more readily consider how existing theories within CSCW may be applied to the coordination dynamics observed within these teams.

Once these (and other) axes have been reasonably explored and we have gained a better overall understanding of the coordination dynamics within small, collocated software development teams, we could then turn our attention toward finding best practices and experimenting with various features in issue tracking systems to see which provide better support for the communicative and collaborative needs of their users. These experiments could manipulate various combinations and trade-offs among the design considerations presented in this thesis as well as elsewhere in the literature. The benefits of new, "from scratch" solutions could be contrasted with the potentially more streamlined approach of layering or integrating new functionality overtop of existing systems. Additionally, the barriers of introducing new toolsets to well-established teams could also be explored and potential workarounds could be tested.

In short, issue tracking systems are but one of many tools used by software development teams of all sizes to help coordinate their efforts in creating quality software. These systems do so by facilitating collaboration amongst team members as well as the communication both inside and outside the immediate development team. Our study has revealed that even small, collocated teams with easy, inexpensive access to rich face-to-face communication *still* make use of issue tracking systems as an essential communication and collaboration tool. We—as a research community—need to continue studying how these tools, and others like them, are used within development teams in order to continue our collective pursuit of improving the overall software development process.

References

M.S. Ackerman and C. Halverson (1998). Considering an organization's memory. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, pages 39–48.

J. Anvik, L. Hiew, and G.C. Murphy (2006). Who should fix this bug? In *Proceedings of the International Conference on Software Engineering*, pages 361–370.

J. Aranda and G. Venolia (2009). The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the International Conference on Software Engineering*, pages 298–308.

Atlassian (2009). *Workflows and fields*. Retrieved October 30, 2009 from http://www.atlassian.com/software/jira/tour/workflow.jsp

Balsamiq Studios (2009). *Balsamiq Studios, makers of plugins for Web Office applications.* Retrieved October 30, 2009 from http://www.balsamiq.com

N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann (2008). What makes a good bug report? In *Proceedings of the ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 308–318.

H. Beyer and K. Holtzblatt (1997). *Contextual Design: Defining Customer-Centered Systems (Interactive Technologies)*. San Francisco: Morgan Kaufmann.

R. Black (2002). *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*. New York: Wiley Publishing.

Bugzilla (2009). *Life Cycle of a Bug*. Retrieved October 30, 2009 from http://www.bugzilla.org/docs/tip/en/html/lifecycle.html

B. Buxton (2007). *Sketching User Experiences: Getting the Design Right and the Right Design (Interactive Technologies)*. San Francisco: Morgan Kaufmann.

G. Canfora and L. Cerulo (2006). Supporting change request assignment in open source development. In *Proceedings of the ACM Symposium on Applied Computing*, pages 1767–1772.

P.H. Carstensen, C. Sørensen, and T. Tuikka (1995). Let's talk about bugs! *Scandinavian Journal of Information Systems*, **7**(1):33–54.

E.F. Churchill, J. Trevor, S. Bly, L. Nelson, and D. Cubranic (2000). Anchored conversations: Chatting in the context of a document. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 454–461.

J. Corbin and A. Strauss (2008). *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Los Angeles: Sage Publications, 3rd edition.

M. D'Ambros, M. Lanza, and M. Pinzger (2007). "A bug's life": Visualizing a bug database. In *Proceedings of the IEEE International Workshop on Visualizing Software for Analysis and Understanding*, pages 113–120.

T. Dingsøyr and E. Røyrvik (2003). An empirical study of an informal knowledge repository in a medium-sized software consulting company. In *Proceedings of the International Conference on Software Engineering*, pages 84–92.

J.B. Ellis, S. Wahid, C. Danis, and W.A. Kellogg (2007). Task and social visualization in software development: Evaluation of a prototype. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 577–586.

T. Erickson, D.N. Smith, W.A. Kellogg, M. Laff, J.T. Richards, and E. Bradner (1999). Socially translucent conversations: Social proxies, persistent conversation, and the design of "Babble". In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 72–79.

ExtraView Corporation (2009). *ExtraView website*. Retreived October 30, 2009 from http://www.extraview.com

Facebook (2009). *Welcome to Facebook!* Retreieved October 30, 2009 from http://www.facebook.com

N. Fenton and M. Neil (1999). A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, **25**(5):675–689.

G. Fitzpatrick (2003). *The Locales Framework: Understanding and Designing for Wicked Problems*. Berlin: Springer.

G. Fitzpatrick, P. Marshall, and A. Phillips (2006). CVS integration with notification and chat: Lightweight software team collaboration. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 49–58.

Fog Creek Software (2009a). *FogBugz - Bug & Issue Tracking, Project Management, Help Desk Software*. Retrieved October 30, 2009 from http://www.fogbugz.com

Fog Creek Software (2009b). *The Basics of Bug Tracking*. Retreived October 30, 2009 from http://www.fogcreek.com/FogBugz/docs/70/topics/basics/Thebasicsofbugtracking.html

Greasemonkey (2009). *Greasespot*. Retreived October 30, 2009 from http://www.greasespot.net

R.E. Grinter (1995). Using a configuration management tool to coordinate software development. In *Proceedings of the ACM Conference on Organizational Computer Systems*, pages 168–177.

J. Grudin (1994). Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, **37**(1):92-105.

M. Gunderloy (2007). *Painless Project Management with FogBugz*. New York: Apress, 2nd edition.

C. Halverson, J. Ellis, C. Danis, and W. Kellogg (2006). Designing task visualizations to support the coordination of work in software development. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 39–48.

C. Henderson (2006). *Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications*. Sebastool, CA: O'Reilly Media.

J. Hollan, E. Hutchins, and D. Kirsh (2000). Distributed cognition: Toward a new foundation for human-computer interaction research. *ACM Transactions on Computer–Human Interaction*, **7**(2):174–196.

D.B. Knudsen, A. Barofsky, and L.R. Satz (1976). A modification request control system. In *Proceedings of the International Conference on Software Engineering*, page 187–192.

R.E. Kraut and L.A. Streeter (1995). Coordination in software development. *Communications of the ACM*, **38**(3):69–81.

T.D. LaToza, G. Venolia, G., and R. DeLine (2006). Maintaining mental models: A study of developer work habits. In *Proceedings of the International Conference on Software Engineering*, pages 492–501.

J. Liebowitz and T. Beckman (1998). *Knowledge organizations: what every manager should know*. Boca Raton, FL: St. Lucie.

Microsoft Corporation (2009a). *Excel Home Page - Microsoft Office Online*. Retrieved October 30, 2009 from http://office.microsoft.com/excel

Microsoft Corporation (2009b). *Microsoft Office SharePoint Server - Connecting People, Process, and Information*. Retrieved October 30, 2009 from http://sharepoint.microsoft.com

B.A. Nardi, S. Whittaker, and E. Bradner (2000). Interaction and outeraction: Instant messaging in action. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 79–88.

NCH Software (2009). *Digital Transcriber - Free Transcription Software Player*. Retrieved October 30, 2009 from http://www.nch.com.au/scribe/

J. Oikarinen and D. Reed (1993). *RFC1459 - Internet Relay Chat Protocol*. Retrieved October 30, 2009 from http://www.faqs.org/rfcs/rfc1459.html

D.E. Perry, N. Staudenmayer, and L.G. Votta (1994). People, organizations, and process improvement. *IEEE Software*, **11**(4):36–45.

R.S. Pressman (2004). *Software Engineering: A Practitioner's Approach***.** New York: McGraw-Hill, 6th edition.

C.R. Reis and R.P. de Mattos Fortes (2002). An overview of the software engineering process and tools in the Mozilla project. In *Proceedings of the Workshop on Open Source Software Development*, University of Newcastle upon Tyne, pages 155–175.

R. Sandusky and L.Gasser (2005). Negotiation and the coordination of information and activity in distributed software problem management. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work*, pages 187–196.

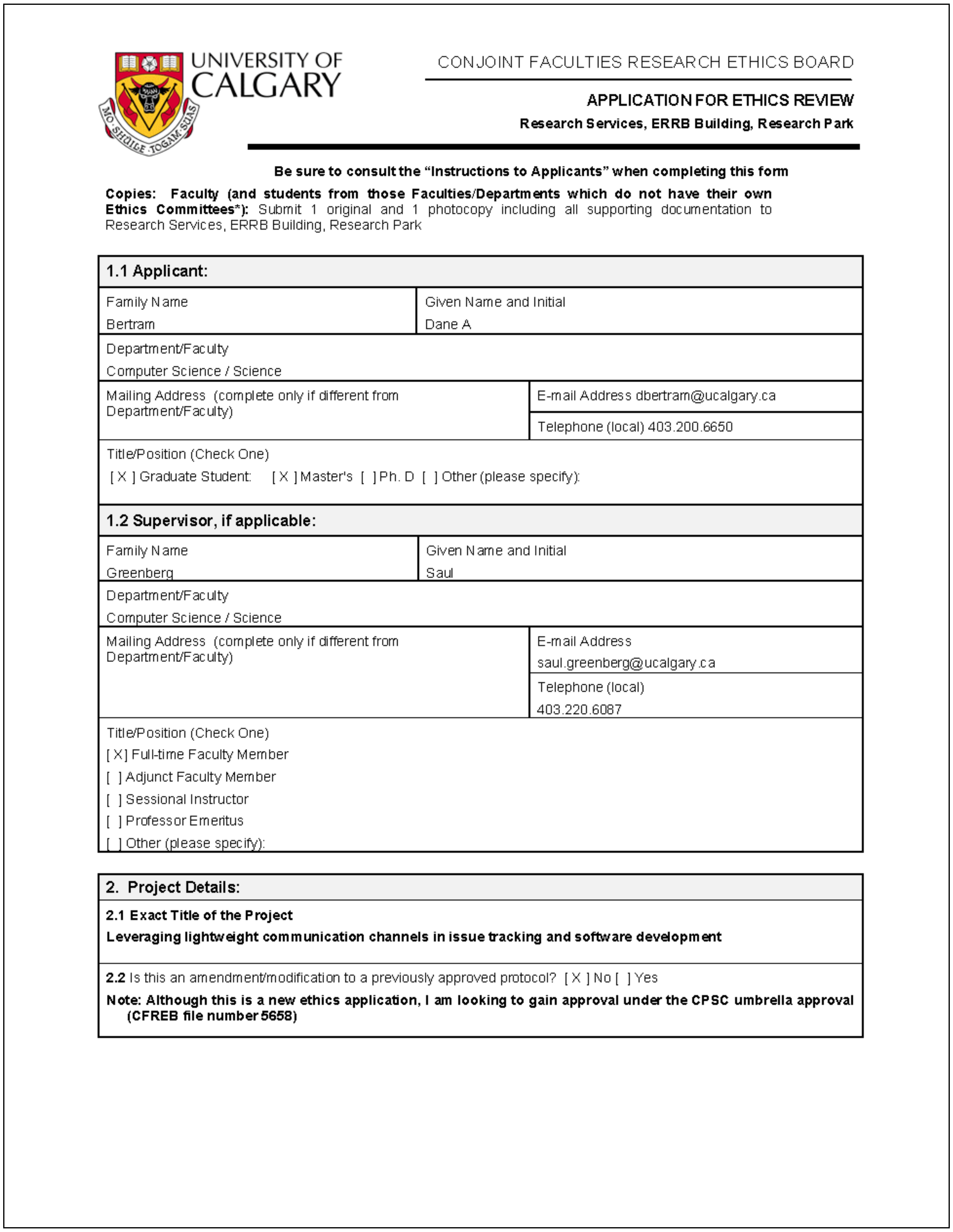
S.L. Star and J.R. Griesemer (1989). Institutional ecology, 'translations' and boundary objects: Amateurs and professionals in Berkeley's Museum of Vertebrate Zoology, 1907–39. *Social Studies of Science*, **19**(3):387–420.

M. Storey, J. Ryall, R.I. Bull, D. Myers, and J. Singer (2008). TODO or to bug: exploring how task annotations play a role in the work practices of software developers. In *Proceedings of the International Conference on Software Engineering*, pages 251–260.

Y. Ye (2006). Supporting software development as knowledge-intensive and collaborative activity. In *Proceedings of the International Workshop on Interdisciplinary Software Engineering Research*, pages 15–22.

Appendix A. Ethics Application

Below are the ethics application, consent form, and approval certificate associated with the study described in Chapter 3.



UNIVERSITY OF CALGARY

CONJOINT FACULTIES RESEARCH ETHICS BOARD

APPLICATION FOR ETHICS REVIEW

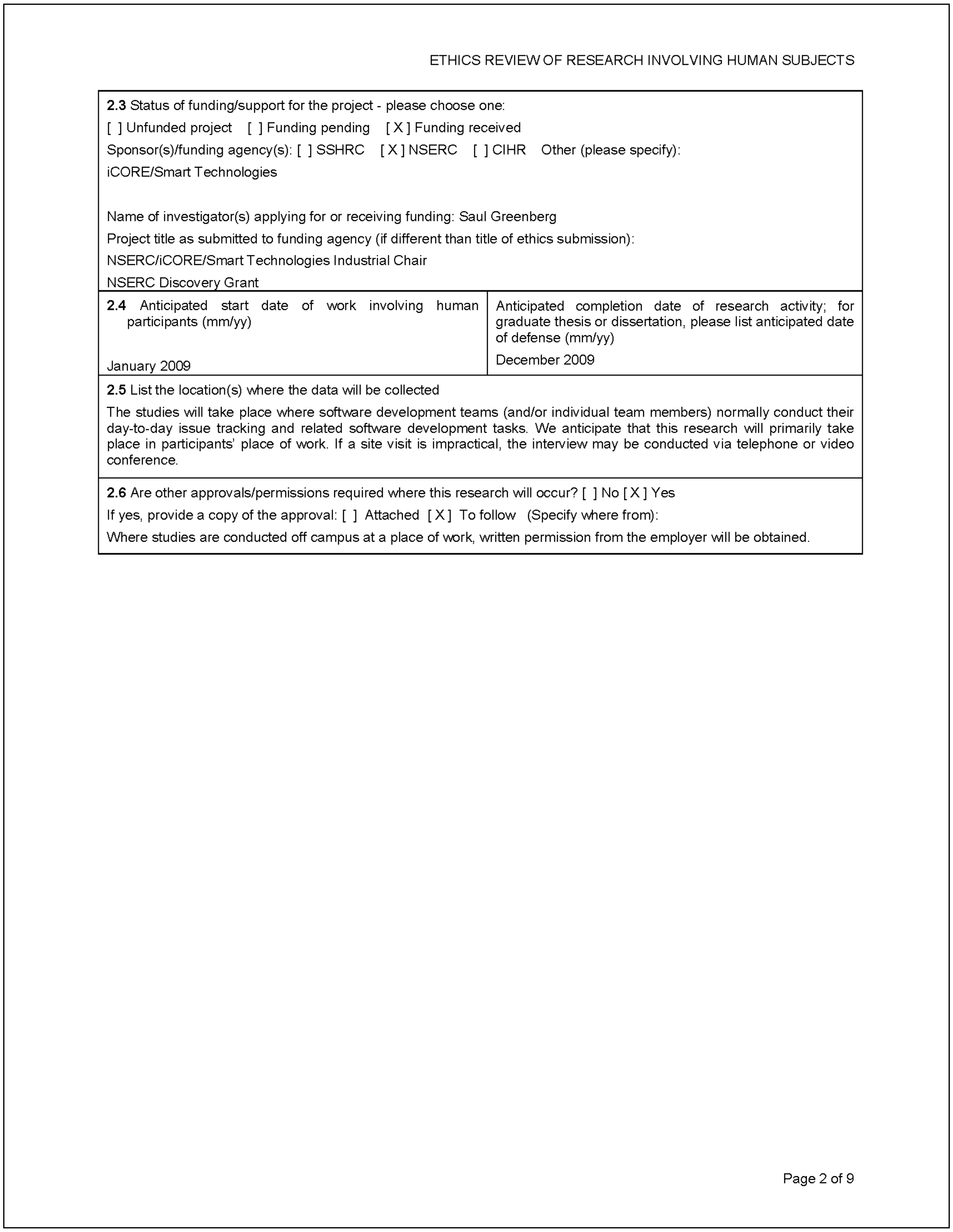
Research Services, ERRB Building, Research Park

Be sure to consult the "Instructions to Applicants" when completing this form

Copies: Faculty (and students from those Faculties/Departments which do not have their own Ethics Committees*): Submit 1 original and 1 photocopy including all supporting documentation to Research Services, ERRB Building, Research Park

1.1 Applicant:	
Family Name Bertram	Given Name and Initial Dane A
Department/Faculty Computer Science / Science	
Mailing Address (complete only if different from Department/Faculty)	E-mail Address dbertram@ucalgary.ca Telephone (local) 403.200.6650
Title/Position (Check One) [X] Graduate Student: [X] Master's [ ] Ph. D [ ] Other (please specify):	
1.2 Supervisor, if applicable:	
Family Name Greenberg	Given Name and Initial Saul
Department/Faculty Computer Science / Science	
Mailing Address (complete only if different from Department/Faculty)	E-mail Address saul.greenberg@ucalgary.ca Telephone (local) 403.220.6087
Title/Position (Check One) [X] Full-time Faculty Member [ ] Adjunct Faculty Member [ ] Sessional Instructor [ ] Professor Emeritus [ ] Other (please specify):	

2. Project Details:
2.1 Exact Title of the Project **Leveraging lightweight communication channels in issue tracking and software development**
2.2 Is this an amendment/modification to a previously approved protocol? [X] No [ ] Yes **Note: Although this is a new ethics application, I am looking to gain approval under the CPSC umbrella approval (CFREB file number 5658)**



2.3 Status of funding/support for the project - please choose one:

[ ] Unfunded project [ ] Funding pending [X] Funding received

Sponsor(s)/funding agency(s): [ ] SSHRC [X] NSERC [ ] CIHR Other (please specify):

iCORE/Smart Technologies

Name of investigator(s) applying for or receiving funding: Saul Greenberg

Project title as submitted to funding agency (if different than title of ethics submission):

NSERC/iCORE/Smart Technologies Industrial Chair

NSERC Discovery Grant

2.4 Anticipated start date of work involving human participants (mm/yy) January 2009	Anticipated completion date of research activity; for graduate thesis or dissertation, please list anticipated date of defense (mm/yy) December 2009

2.5 List the location(s) where the data will be collected

The studies will take place where software development teams (and/or individual team members) normally conduct their day-to-day issue tracking and related software development tasks. We anticipate that this research will primarily take place in participants' place of work. If a site visit is impractical, the interview may be conducted via telephone or video conference.

2.6 Are other approvals/permissions required where this research will occur? [ ] No [X] Yes

If yes, provide a copy of the approval: [ ] Attached [X] To follow (Specify where from):

Where studies are conducted off campus at a place of work, written permission from the employer will be obtained.

2.7 Provide a succinct summary of the purpose, objectives, and aims of the research. Describe your methodology, and what will be required of the human participants. Please use language that can be understood by a non-specialist. Up to 1 additional page may be added, if required. (Note: Project descriptions exceeding the two-page limit will not be considered.) **REMINDER:** Be sure to include a copy of any questionnaire(s) or test instrument(s).

The purpose of the research is to better understand the role lightweight communication (email, chat, instant messaging, wikis, twitter, casual verbal conversation, etc.) plays in the day-to-day completion of software development tasks and more specifically, how such communication might be applied to issue tracking (software bugs, feature requests, etc.) and how this communication might be better leveraged in software development tools. We want to determine 1) if these lightweight communication channels are used during issue tracking tasks, 2) which channels are used, how they are used, and how often, and 3) how the conversations facilitated by such channels might be useful in the future for other developers as well as themselves.

Participants will be asked to engage in five different research activities:

1. A contextual interview during issue tracking tasks and related software development tasks (whichever issue tracking tasks the participant may be working on or preparing to work on as part of their regular work routine) for a period of time not to exceed two hours. During these tasks a researcher will observe and make notes. The participant may be asked to think aloud and clarifying questions may also be asked by the researcher.
2. A questionnaire about the participant and his/her experiences and practices with lightweight communication in respect to issue tracking tasks (see attached).
3. A semi-structured interview about their software development habits related to issue tracking and their usage of lightweight communication channels (may be recorded using audio and/or video with the permission of the participant). A series of open-ended questions will be used to guide the general direction of the interview, but emergent issues and ideas will be explored as they come up (both that arise during the interview and that arise during the observations). The following are samples of the kinds of open-ended questions we anticipate using:
    a. What are the tools you use most frequently when preparing for or working on an issue tracking task? Which do you use the most? Why do you use these particular tools?
    b. What types of information do you frequently require from others in order to complete your day-to-day issue tracking and related software development tasks? How do you usually obtain this information?
    c. What shortcomings have you experienced with the tools you currently use? What workarounds have you come up with? Are there any shortcomings that stop you from completing your issue tracking or related software development tasks?
    d. What communication channels do you use to communicate with your colleagues and other project stakeholders? How frequently do you use each communication channel? Do you have a preference as to which communication channel you use? If so, under what conditions and why?
4. Sketching of his/her ideas for potential user interfaces and tools that would make use of such communication (including design suggestions and potential requirements for addressing their needs and wants).
5. Feedback on prototype systems and software development tools (sketches, computer software, textual and verbal descriptions, etc). This task may be a recurring one in order to facilitate an iterative design process.

As well, information about the physical configuration of the participant's work area as well as the general layout of their place of work will be collected via various means as appropriate: sketching, photos, and/or video.

3. Recruitment of Participants

3.1 Describe the "types" of participants (e.g. city planners, environmental specialists, minor age children, University students) to be involved in the research. Be very specific about your method(s) for recruiting them, and comment on who will do the recruiting. Describe how and where you will advertise your project. **Include a copy of your recruitment notice, advertisement, information sheet, as well as that used by a sponsor or supportive organization, if applicable.** If actively seeking participation by speaking to specific groups, include the text used for verbal presentations. If remuneration/compensation is offered, provide details, including amount and confirm the budget provisions to meet these obligations. Describe any provisions that have been made to accommodate the participants' language.

We anticipate recruiting industrial software developers and their colleagues. We will rely on an indirect snowball sampling method to identify potential participants via word of mouth referrals as well as by our existing relationships (previous employers, graduates from our lab working in industry, colleagues within the university, researchers conducting similar research, etc.). This recruitment method may be augmented as needed by email recruitment notices (forwarded to potential participants by our friends and colleagues or current participants). Our email recruitment notice will use language such as the following:

> My name is Dane Bertram. I am writing to you on behalf of myself and my colleague Saul Greenberg. We are researchers from the University of Calgary and are conducting a study to learn about the role lightweight communication (email, chat, instant messaging, wikis, twitter, casual verbal conversation, etc.) plays in the day-to-day completion of issue tracking and related software development tasks. Our goal is to use this information to help develop better software development tools. The study will involve several hours of your time at a location of your choosing (ideally wherever you typically complete your issue tracking tasks). A researcher will observe you as you work , and talk with you about your work activities, habits, and practices. You and any of your colleagues who participate in the study will be compensated $20 for their participation. Please contact me at dbertram@ucalgary.ca for more information. I look forward to hearing from you! In addition, if you have friends or other colleagues who you believe might also be interested in participating in this research, we would be grateful if you were to talk to them about this research opportunity and/or forward them this email with information about our study. Thanks so much for your help!

Depending on the type and culture of the organization, we will recruit these industrial programmers directly, or via their employer. We hold no power to coerce participation from an employee. Furthermore, although employers may have power over their employees, participation in the study is largely equivalent to the employee's normal duties and remains completely voluntary.

4. Informed Consent

4.1 Described the informed consent process. **Provide a copy of your consent form.** If there is no written consent form, please provide an explanation for this and details about your alternative procedures. If obtaining verbal consent, a script containing the same points normally covered by written consent is required. Are participants minors or, for other reasons, not able to provide fully informed consent? Explain and justify, and describe alternative procedures (e.g. parental consent).

Please see attached consent form. The consent form will be administered after instructions have been made and the basic goals and requirements of the study have been explained verbally.

4.2 When and how will people be informed of the right to withdraw from the study? What procedures will be followed for people who wish to withdraw at any point during the study? What happens to the information contributed to this point? Please note that the CFREB does not require that researchers withdraw/destroy partial data in cases of participant withdrawal, provided that it is made clear on the informed consent form that data collected to the point of withdrawal will be retained/used.

People will be informed of the right to withdraw during the informed consent process. They may cease participation in research activities at any time. Information contributed to the point of withdrawal will be retained and used in the research, as is noted in the consent form.

4.3 Do you plan follow-up procedures with participants? [ ] No [X] Yes, if yes, what are they? Does your research design require formal debriefing? [ ] No [ ] Yes, if yes, please provide details about the procedures you will use.

Follow-up procedures may include (with permission from the participant) soliciting their feedback on refined prototypes systems and software development tools (sketches, computer software, textual and verbal descriptions, etc) at a later point in time from the initial interview and observation period.

5. Privacy: Confidentiality and Anonymity:

5.1 Check all that apply: **Participant contributions will be: [ ] public and cited; [X] anonymous; [ ] confidential.** Explain the steps you propose to respect an individual's privacy. Describe these precautions in terms of access to raw data, as well as in terms of the write-up of the results.

Only the researchers will have acces to the raw data. When participants' contributions are discussed in written or oral presentations of this research, they will be referred to with either participant numbers or pseudonymns. Video that participants have agreed to allow us to share in presentations of this research will not include a person's identifying information (e.g., no names).

5.2 Provide specific details about the security procedures for the data as well as plans for the ultimate disposal of records/data. Who will have access to confidential data now or in the future? Specify the length of time the data will be retained and the plans for disposal of records/data.

Raw data will be saved either on password protected computers or in locked university filing cabinets or rooms. Only the researchers will have access to the confidential data. The data will be retained for five years, after which it will be permanently deleted.

6. **Estimation of Risks**: Will this study involve the following? Please check Y **When responding, see also Section 3– Information to Help Applicants**	None	Minimal Risk	More than Minimal risk
6.1 Psychological or emotional manipulations – might a participant feel demeaned, embarrassed, worried or upset? Could subjects feel fatigued or stressed?	X		
6.2 Are there questions that may be upsetting to the respondent?	X		
6.3 Does your study have the potential for identifying distressed individuals?	X		
6.4 Is there any physical risk or physiological manipulation?	X		
6.5 Is any deception involved? Withholding of information from, or misinforming, participants?	X		
6.6 Is there any social risk - possible loss of status, privacy and/or reputation?	X		
6.7 Do you see any chance that subjects might be harmed in any way?	X		
6.8 Is there any potential for the perception of coercion? That is, might prospective participants feel pressured to participate in the research (due to, for instance, actual or perceived power relationships between those involved in recruiting and those being recruited, e.g. manager/employee or teacher/student)?		X See text in 3.1 regarding recruiting	
6.9 Are the risks similar to those encountered by the subjects in everyday life?	[X] Yes [ ] No if "no" , elaborate		

Page 5 of 9

- If you answered, "more than minimal risk" to any of the above, describe the manipulations and/or potential risks as well as the safeguards or procedures you have in place. Please provide justification for any risks involved and explain why alternative approaches involving less risk cannot be used. Use additional pages, as required.
- If your study has the potential to upset or distress individuals, arrangements must be made to mitigate such effects. Describe the arrangements you have made. Have participants been informed of any costs to be incurred by them for services? **See "Provision for Rescue – Guidelines for Applicants"**
- If your study has the potential to identify upset or distressed individuals, you must describe the arrangements you have made (if any) to assist these individuals. If you do not make any arrangements, please explain why. Have participants been informed of any costs to be incurred by them for services?
- If, prior to the start of the research session, participants will not be fully informed of everything that will be required of them or deliberately misinformed about some aspect of the study, explain why. Please describe the procedures in detail and justify why deception is necessary to conduct the research.
- If the potential for any perception of coercion exists, please explain what measures have been put in place to minimize the possibility that individuals will feel pressured to participate.

7. Benefits

What are the likely benefits of the research to the researcher, the participants, the research community and society, at large, that would justify asking people to participate?

The benefits of the research should include a more refined understanding of the role that lightweight communication plays in software development teams with respect to issue tracking tasks. The communication of these findings to the computer science community may lead to more compelling software development tool and light communication system design that may, in turn, foster more interesting and satisfying issue tracking experiences among developers such as our participants.

8. Signatures

I/We, the undersigned, certify that (a) the information contained in this application is accurate; (b) conduct of the proposed research will not commence until ethical certification has been granted; (c) the Board will be advised of any revisions to the protocol arising before or after ethical certification is granted; (d) an annual renewal report will be filed 12 months from the date that ethics approval is issued, and a final report will be filed immediately upon completion of research activity. Failure to submit renewal or final reports in a timely manner will be considered a breach of University and Tri-Council policy, and may result in the suspension of research funding and/or the research being rendered academically invalid; students who fail to submit reports may be barred from graduating. Conduct of research using human subjects that has not received ethics certification is a breach of University policy on integrity in scholarly activity.

Applicant's signature: ______________________________ Date: ______________________

Supervisor's Signature: I have been involved in the preparation of this application, and agree with the information it contains.

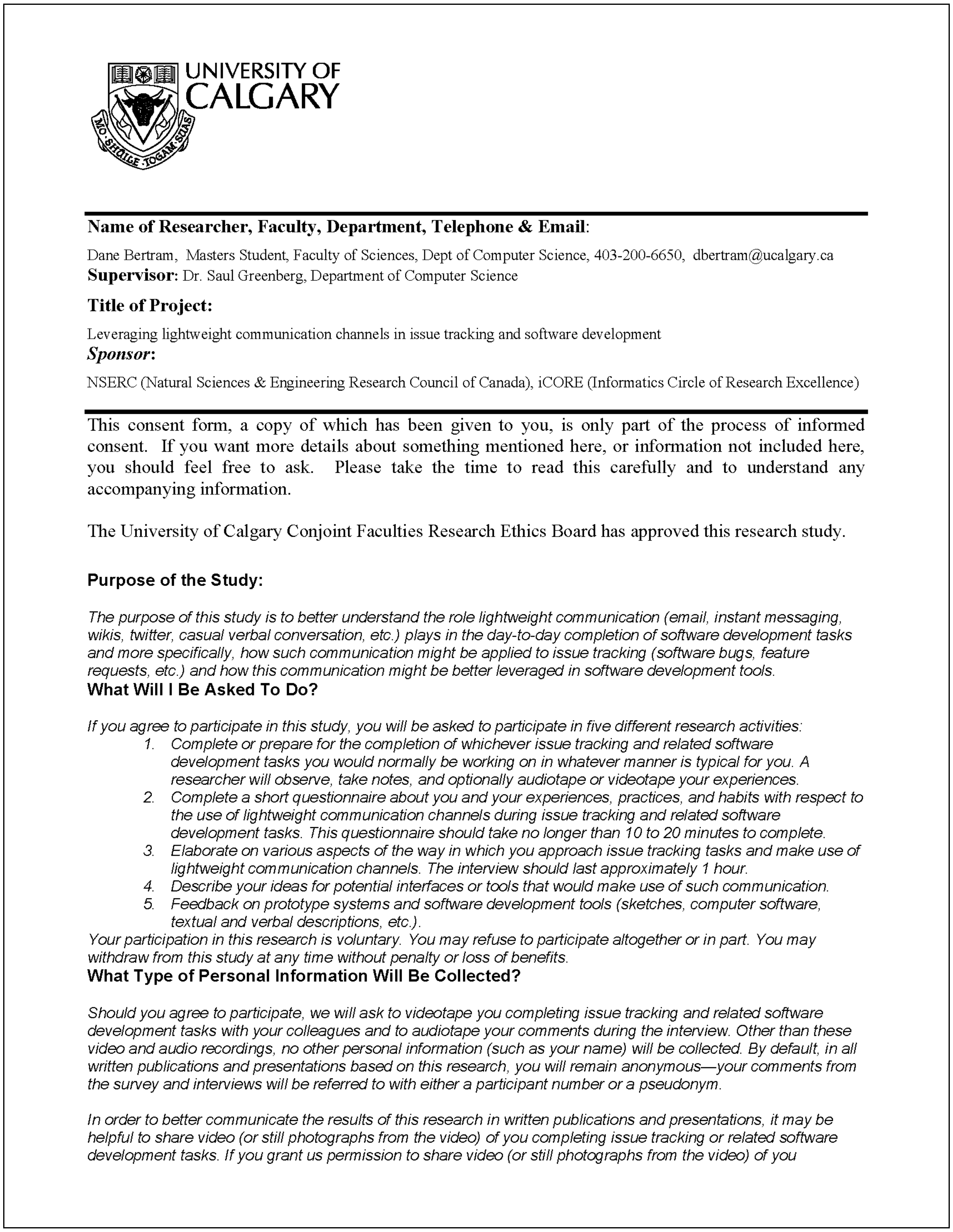
Supervisor's Signature: ______________________________ Date: ______________________

PROTOCOL CHECKLIST – required	N/A	Attached
Copy of the verbal or written explanation that will be provided to participants before they are asked for consent to participate		See text in 3.1
Copy of the informed consent(s) that will be distributed to each participant.		X
If written consent is not used, a detailed explanation of alternative procedures is required in Section 4 of this application, along with one or more of the following:	X	
• If verbal consent is to be obtained, (e.g. telephone surveys), a script containing the equivalent points covered by written consent is required.	X	
• Totally anonymous online or mail out questionnaires: Signed consent is not necessary. A covering letter, containing the equivalent points covered by written consent, is required.	X	
Copies of questionnaire(s), sample questions or thematic overview, interview guide		Sample question-naire attached; See also 2.7 for sample interview questions
Recruitment: Your recruitment notice, advertisement, and/or information sheet as well as that used by a sponsor or supportive organization, as may be applicable		See text in 3.1
Documents or information specific to or requested by the potential sponsor.	X	
Completed and signed application for review with the required number of copies.		X

Revised: 03/07

Note: The information contained in this application is collected under the authority of the Freedom of Information and Protection of Privacy (FOIP) Act. It will be used to evaluate your application for ethics certification. Anonymized data will also be used to fulfill reporting obligations.

If you have any questions about the collection or use of this information, please contact the Ethics Resource Officer (Research Services, ERRB Building, Research Park) at (403)220-3782.



UNIVERSITY OF CALGARY

Name of Researcher, Faculty, Department, Telephone & Email:

Dane Bertram, Masters Student, Faculty of Sciences, Dept of Computer Science, 403-200-6650, dbertram@ucalgary.ca
Supervisor: Dr. Saul Greenberg, Department of Computer Science

Title of Project:

Leveraging lightweight communication channels in issue tracking and software development
***Sponsor*:**

NSERC (Natural Sciences & Engineering Research Council of Canada), iCORE (Informatics Circle of Research Excellence)

This consent form, a copy of which has been given to you, is only part of the process of informed consent. If you want more details about something mentioned here, or information not included here, you should feel free to ask. Please take the time to read this carefully and to understand any accompanying information.

The University of Calgary Conjoint Faculties Research Ethics Board has approved this research study.

Purpose of the Study:

The purpose of this study is to better understand the role lightweight communication (email, instant messaging, wikis, twitter, casual verbal conversation, etc.) plays in the day-to-day completion of software development tasks and more specifically, how such communication might be applied to issue tracking (software bugs, feature requests, etc.) and how this communication might be better leveraged in software development tools.

What Will I Be Asked To Do?

If you agree to participate in this study, you will be asked to participate in five different research activities:

1. *Complete or prepare for the completion of whichever issue tracking and related software development tasks you would normally be working on in whatever manner is typical for you. A researcher will observe, take notes, and optionally audiotape or videotape your experiences.*
2. *Complete a short questionnaire about you and your experiences, practices, and habits with respect to the use of lightweight communication channels during issue tracking and related software development tasks. This questionnaire should take no longer than 10 to 20 minutes to complete.*
3. *Elaborate on various aspects of the way in which you approach issue tracking tasks and make use of lightweight communication channels. The interview should last approximately 1 hour.*
4. *Describe your ideas for potential interfaces or tools that would make use of such communication.*
5. *Feedback on prototype systems and software development tools (sketches, computer software, textual and verbal descriptions, etc.).*

Your participation in this research is voluntary. You may refuse to participate altogether or in part. You may withdraw from this study at any time without penalty or loss of benefits.

What Type of Personal Information Will Be Collected?

Should you agree to participate, we will ask to videotape you completing issue tracking and related software development tasks with your colleagues and to audiotape your comments during the interview. Other than these video and audio recordings, no other personal information (such as your name) will be collected. By default, in all written publications and presentations based on this research, you will remain anonymous—your comments from the survey and interviews will be referred to with either a participant number or a pseudonym.

In order to better communicate the results of this research in written publications and presentations, it may be helpful to share video (or still photographs from the video) of you completing issue tracking or related software development tasks. If you grant us permission to share video (or still photographs from the video) of you *completing issue tracking or related software development tasks in written publications or presentations of this research, there is a chance that you may be recognized and so we cannot guarantee your anonymity. We will never, however, reveal your name in association with your image. Additionally, digital copies of publications or presentations may be posted online which may include video (or still photographs from the video) of you.*

I grant permission for video (or still photos from the video) of me completing issue tracking and/or software development tasks to be shared in publications or presentations of this research: *Yes: ___ No: ___*

I grant permission to be audio taped: *Yes: ___ No: ___*

Are there Risks or Benefits if I Participate?

There are no reasonably foreseeable risks, harms, or inconveniences that should result from your participation in this study. For participating in this study you will receive $20.

What Happens to the Information I Provide?

Participation in this research is completely voluntary and confidential. You are free to discontinue participation at any time during the study. Any information you contribute up to the point at which you choose to discontinue your participation will be retained and used in the study. No one except the researchers will be allowed to see or hear any personally-identifiable information unless you have given permission for us to share video or photographs of you completing software development and issue tracking tasks in publications or presentations of this research. The audio/video tapes, sketches, and questionnaires will be kept on password-protected university computers or in a locked cabinet only accessible by the researchers. The data will be stored for five years, after which it will be permanently erased.

Signatures (written consent)

Your signature on this form indicates that you 1) understand to your satisfaction the information provided to you about your participation in this research project, and 2) agree to participate as a research subject. In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. You are free to withdraw from this research project at any time. You should feel free to ask for clarification or new information throughout your participation.

Participant's Name: (please print) ____

Participant's Signature ______________________________ Date: ________________

Researcher's Name: (please print) ____

Researcher's Signature: ______________________________ Date: ________________

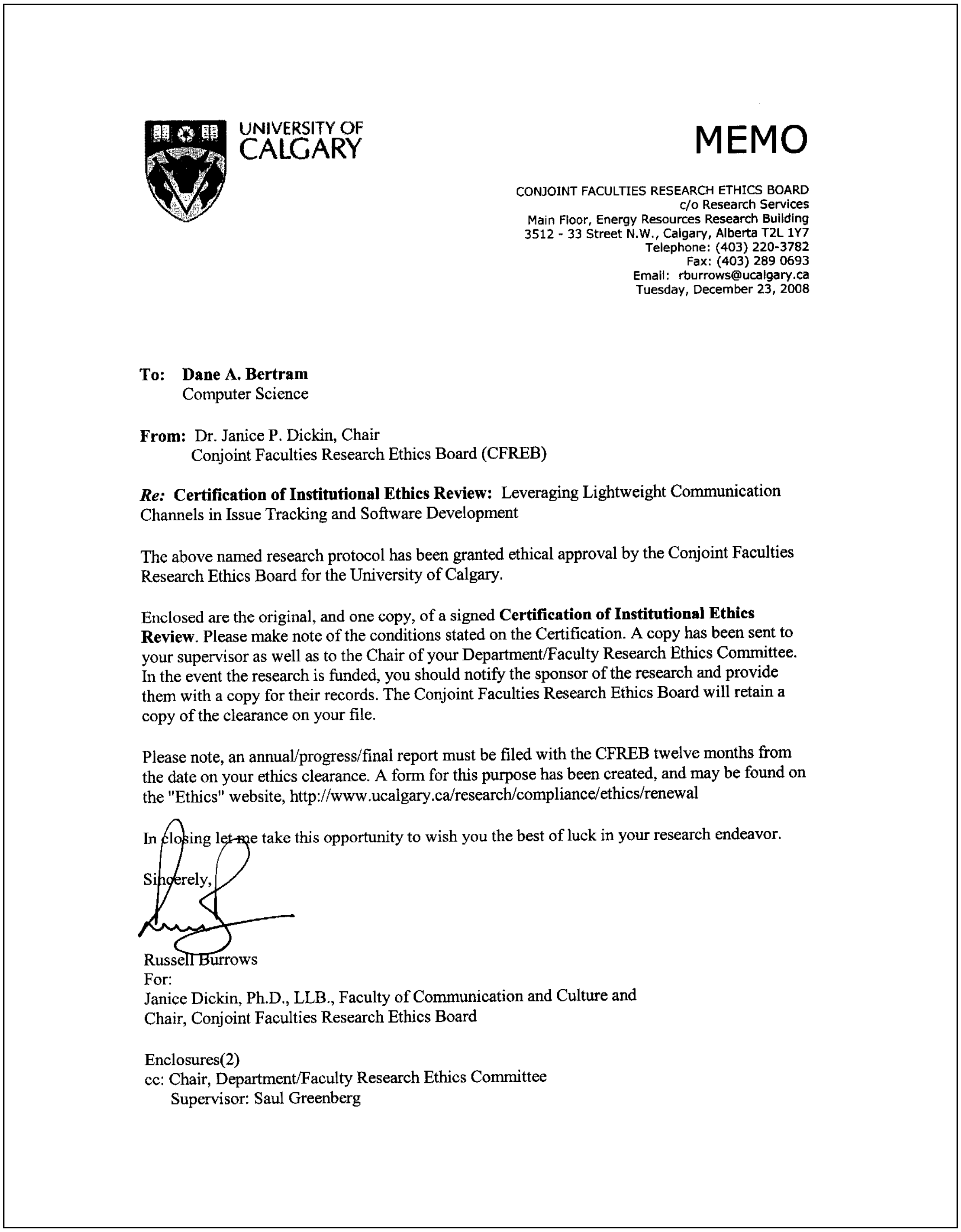
Questions/Concerns

If you have any further questions or want clarification regarding this research and/or your participation, please contact:

Mr. Dane Bertram, Dept of Computer Science/Faculty of Sciences, 403-200-6650, dbertram@ucalgary.ca
OR
Dr. Saul Greenberg, Dept of Computer Science/Faculty of Sciences, 402-220-6087, saul.greenberg@ucalgary.ca

If you have any concerns about the way you've been treated as a participant, please contact the Senior Ethics Resource Officer, Research Services Office, University of Calgary at (403) 220-3782; email rburrows@ucalgary.ca.

A copy of this consent form has been given to you to keep for your records and reference. The investigator has kept a copy of the consent form.



UNIVERSITY OF CALGARY

MEMO

CONJOINT FACULTIES RESEARCH ETHICS BOARD
c/o Research Services
Main Floor, Energy Resources Research Building
3512 - 33 Street N.W., Calgary, Alberta T2L 1Y7
Telephone: (403) 220-3782
Fax: (403) 289 0693
Email: rburrows@ucalgary.ca
Tuesday, December 23, 2008

To: Dane A. Bertram
Computer Science

From: Dr. Janice P. Dickin, Chair
Conjoint Faculties Research Ethics Board (CFREB)

***Re:* Certification of Institutional Ethics Review:** Leveraging Lightweight Communication Channels in Issue Tracking and Software Development

The above named research protocol has been granted ethical approval by the Conjoint Faculties Research Ethics Board for the University of Calgary.

Enclosed are the original, and one copy, of a signed **Certification of Institutional Ethics Review**. Please make note of the conditions stated on the Certification. A copy has been sent to your supervisor as well as to the Chair of your Department/Faculty Research Ethics Committee. In the event the research is funded, you should notify the sponsor of the research and provide them with a copy for their records. The Conjoint Faculties Research Ethics Board will retain a copy of the clearance on your file.

Please note, an annual/progress/final report must be filed with the CFREB twelve months from the date on your ethics clearance. A form for this purpose has been created, and may be found on the "Ethics" website, http://www.ucalgary.ca/research/compliance/ethics/renewal

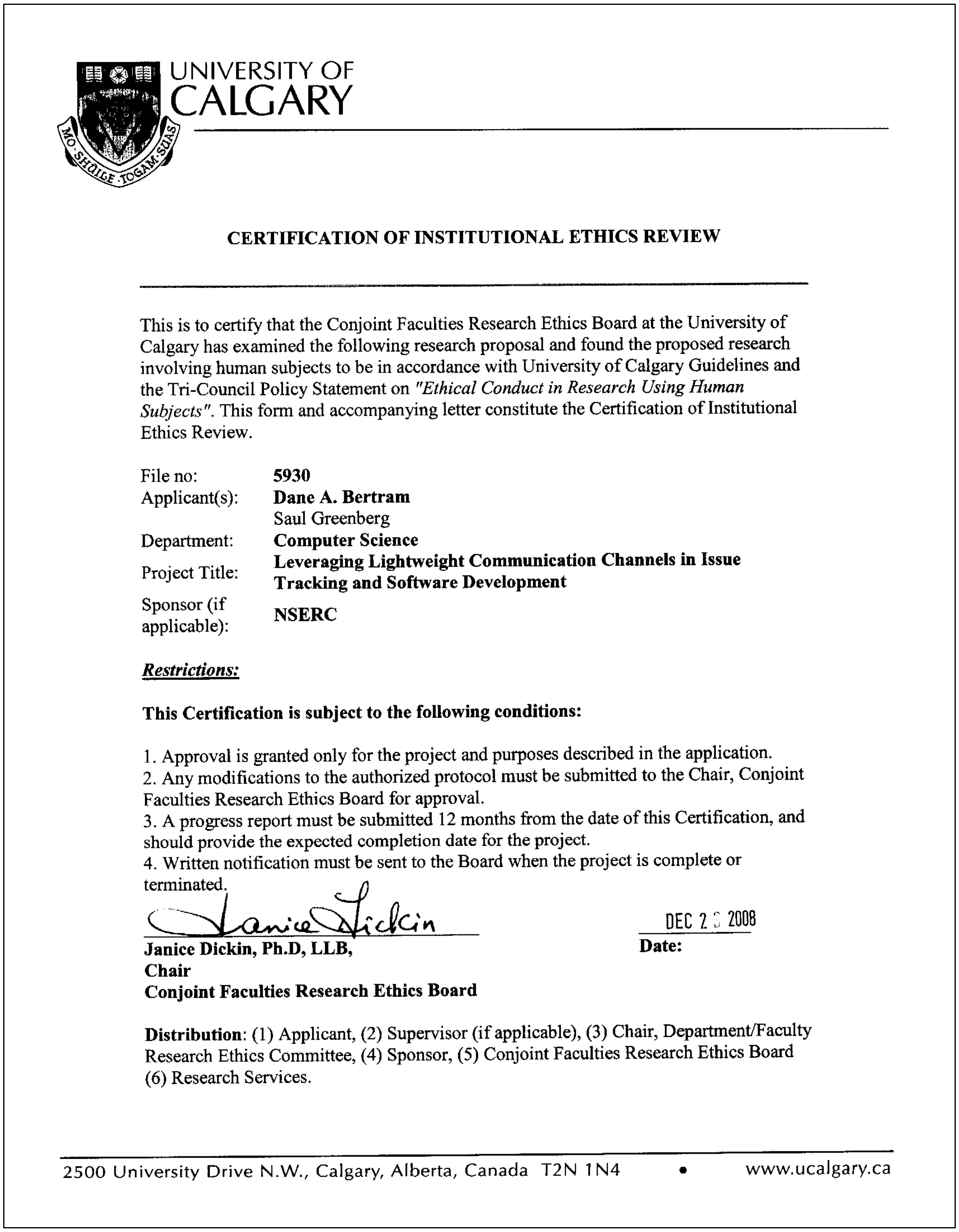
In closing let me take this opportunity to wish you the best of luck in your research endeavor.

Sincerely,

[illegible]

Russell Burrows
For:
Janice Dickin, Ph.D., LLB., Faculty of Communication and Culture and
Chair, Conjoint Faculties Research Ethics Board

Enclosures(2)
cc: Chair, Department/Faculty Research Ethics Committee
Supervisor: Saul Greenberg



UNIVERSITY OF CALGARY

MO SHÙILE TOGAM SUAS

CERTIFICATION OF INSTITUTIONAL ETHICS REVIEW

This is to certify that the Conjoint Faculties Research Ethics Board at the University of Calgary has examined the following research proposal and found the proposed research involving human subjects to be in accordance with University of Calgary Guidelines and the Tri-Council Policy Statement on *"Ethical Conduct in Research Using Human Subjects"*. This form and accompanying letter constitute the Certification of Institutional Ethics Review.

File no:	**5930**
Applicant(s):	**Dane A. Bertram**
	Saul Greenberg
Department:	**Computer Science**
Project Title:	**Leveraging Lightweight Communication Channels in Issue Tracking and Software Development**
Sponsor (if applicable):	**NSERC**

Restrictions:

This Certification is subject to the following conditions:

1. Approval is granted only for the project and purposes described in the application.
2. Any modifications to the authorized protocol must be submitted to the Chair, Conjoint Faculties Research Ethics Board for approval.
3. A progress report must be submitted 12 months from the date of this Certification, and should provide the expected completion date for the project.
4. Written notification must be sent to the Board when the project is complete or terminated.

Janice Dickin

Janice Dickin, Ph.D, LLB,
Chair
Conjoint Faculties Research Ethics Board

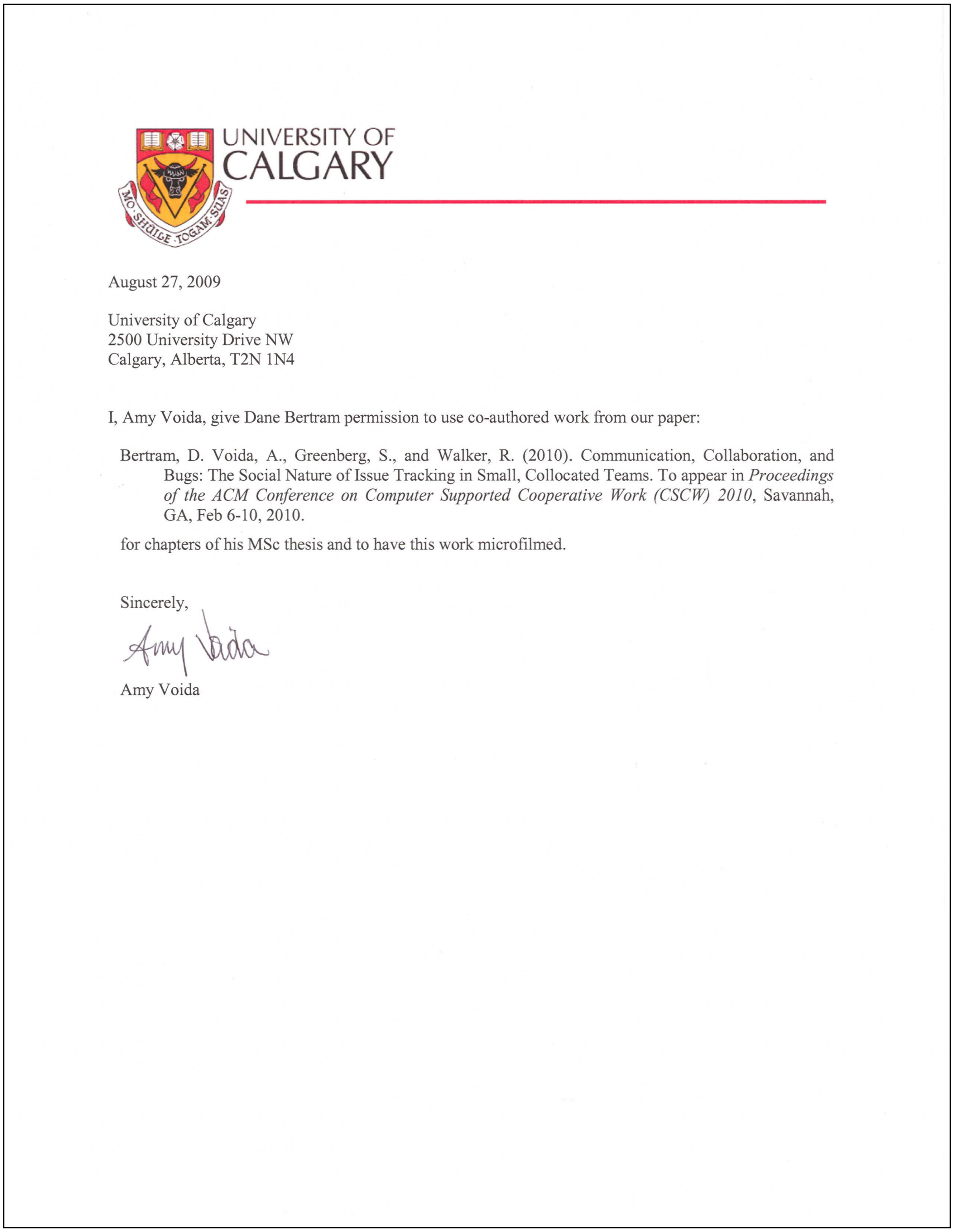
DEC 23 2008
Date:

Distribution: (1) Applicant, (2) Supervisor (if applicable), (3) Chair, Department/Faculty Research Ethics Committee, (4) Sponsor, (5) Conjoint Faculties Research Ethics Board (6) Research Services.

2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4 • www.ucalgary.ca

Appendix B. Co-author Permission

Below are the signed co-author permission forms associated with our paper listed in the Publications section located at the beginning of this document.



UNIVERSITY OF CALGARY

August 27, 2009

University of Calgary
2500 University Drive NW
Calgary, Alberta, T2N 1N4

I, Amy Voida, give Dane Bertram permission to use co-authored work from our paper:

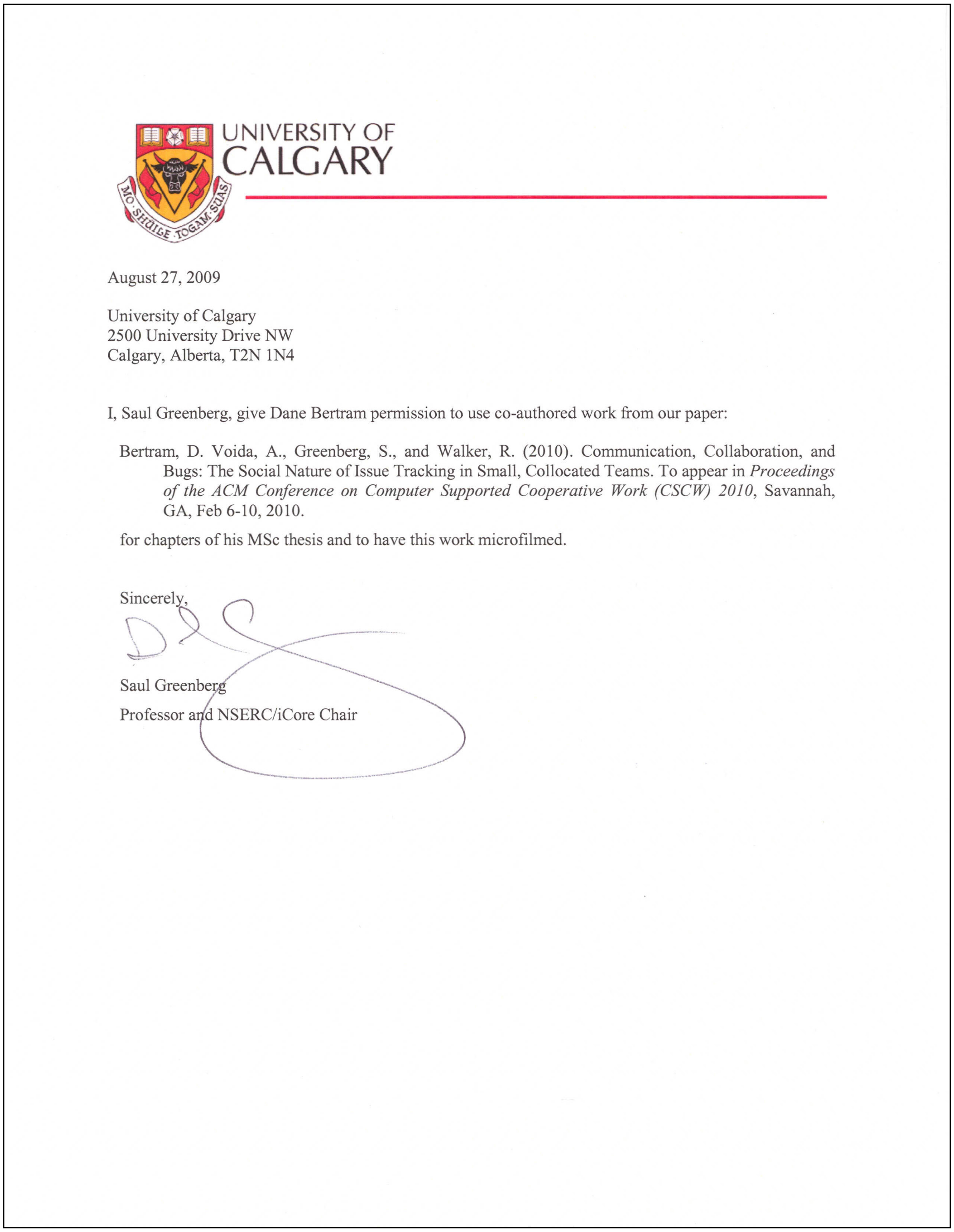
Bertram, D. Voida, A., Greenberg, S., and Walker, R. (2010). Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams. To appear in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW) 2010*, Savannah, GA, Feb 6-10, 2010.

for chapters of his MSc thesis and to have this work microfilmed.

Sincerely,

Amy Voida

Amy Voida



UNIVERSITY OF CALGARY

August 27, 2009

University of Calgary
2500 University Drive NW
Calgary, Alberta, T2N 1N4

I, Saul Greenberg, give Dane Bertram permission to use co-authored work from our paper:

Bertram, D. Voida, A., Greenberg, S., and Walker, R. (2010). Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams. To appear in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW) 2010*, Savannah, GA, Feb 6-10, 2010.

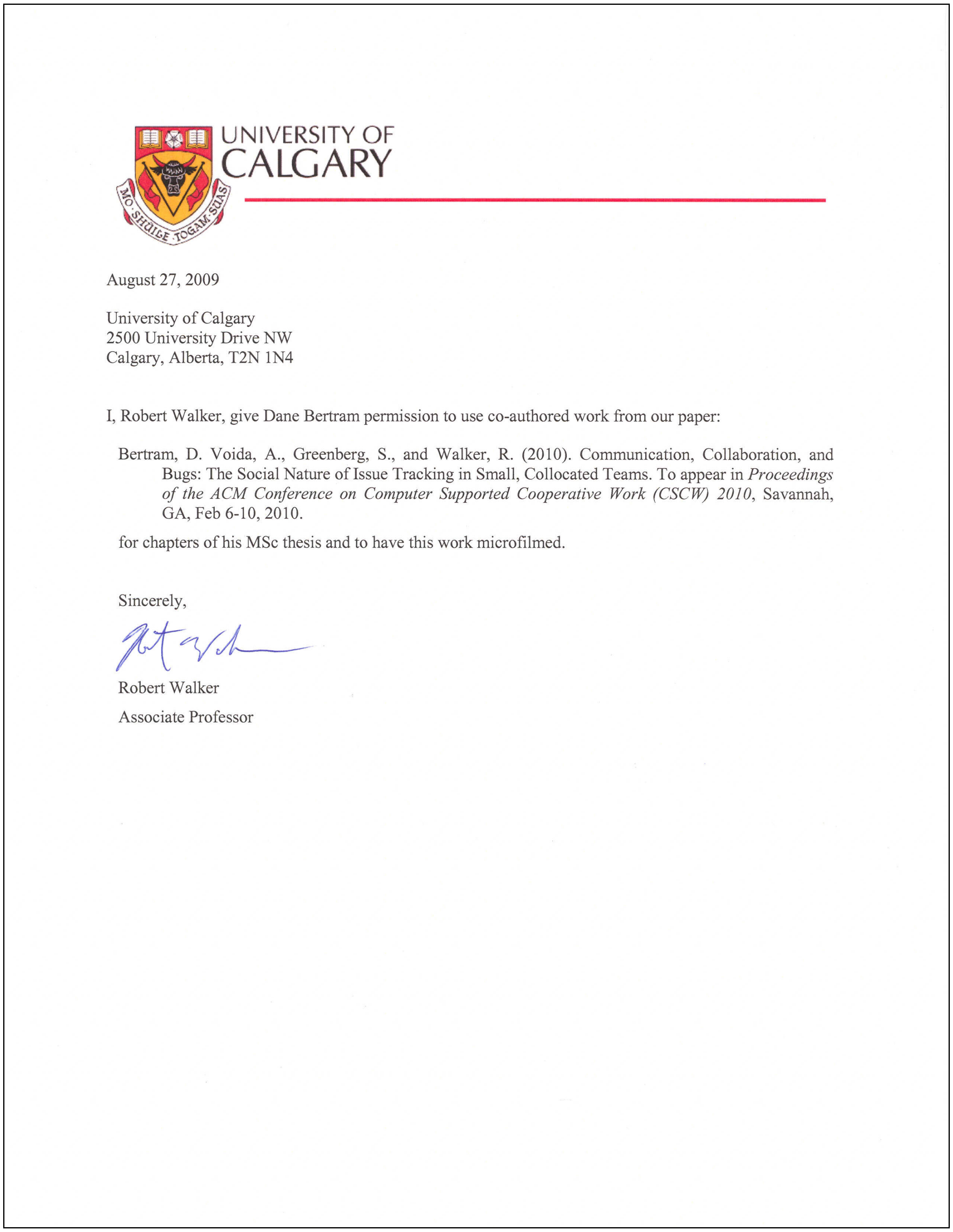
for chapters of his MSc thesis and to have this work microfilmed.

Sincerely,

[illegible]

Saul Greenberg

Professor and NSERC/iCore Chair



UNIVERSITY OF CALGARY

August 27, 2009

University of Calgary
2500 University Drive NW
Calgary, Alberta, T2N 1N4

I, Robert Walker, give Dane Bertram permission to use co-authored work from our paper:

Bertram, D. Voida, A., Greenberg, S., and Walker, R. (2010). Communication, Collaboration, and Bugs: The Social Nature of Issue Tracking in Small, Collocated Teams. To appear in *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW) 2010*, Savannah, GA, Feb 6-10, 2010.

for chapters of his MSc thesis and to have this work microfilmed.

Sincerely,

[illegible]

Robert Walker

Associate Professor