Reducing Waste in Issue Tracking for Regulated Software Development

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Touby Austin Drew

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Maria Gini

December, 2013

Acknowledgements

There are many people that have earned my gratitude for their contribution to my thesis.

The numerous conversations and great advice of my adviser Dr. Maria Gini and my committee members Dr. Mike Whalen, Dr. Mats Heimdahl, and Dr. Mani Subramani have been invaluable to me. Their patience, insight, and mentoring has enabled and greatly improved the quality and content of this thesis.

For our common mission, meaningful work and years of productive collaboration, I am grateful to Medtronic and my many friends there. Several of my remarkable colleagues have played central roles in making this thesis possible.

Of course, my family is the foundation on which all of my accomplishments are built. For their constant love and support in this endeavor and more broadly, I am eternally grateful. My remarkable wife Leigh-Ann Drew has been there for me every step of the way. She has never hesitated to make sacrifices or wavered in her efforts to help me realize this dream. I will never be able to give the time back to her or our children, Kyle Drew and Lily Drew, and for that I am in their debt. For their exceptional efforts to motivate and support my education, I am also especially grateful to my life long friend Jonathan Burns, my sister Amber Drew, and my parents Stephen Drew, Shirley Drew, K. C. Williams, and Theresa Williams.

Dedication

To my wife Leigh-Ann Drew, my father Stephen Drew, and my mother Shirley Drew.

Abstract

Issue tracking in software development for the medical device industry is the process of converting issues into documentation and product to be delivered to such stakeholders as regulatory agencies, clinicians, and the medical device business itself. Issues track and document the concern, its resolution, and associated review. Issue tracking is essential to rigorously documenting, guiding, and exposing the story of the work completed as part of each issue. A significant amount of effort is spent in the issue tracking process and the problem of reducing waste in it is the focus of this thesis.

The work described in this thesis makes contributions in the form of improved support for issue tracking and advances in theory for addressing sources of waste within the medical device industry. This thesis examines sources of waste identified by issue analysis and ethnography. It also describes novel capabilities developed into advisor agent software to address these sources of waste. Then it describes and evaluates the practical use and impact of targeted education and the advisor agent software capabilities for issue tracking over a number of months within the world's largest medical device manufacturer. Finally, evidence of significant improvements in the rate and nature of issue rejection are presented along with data showing greater improvements associated with use of the advisor agent.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis involves the development of local theories about the sources of issue rejection, targeted support to address those, and evaluation of the impact of the support provided. Significant improvement is demonstrated and even greater improvement is linked to the use of the advisor agent software deployed as part of this research.

This chapter provides background and summarizes this thesis and its organization. After introducing software development, issue tracking, and finally the teams and constraints of the domain that this thesis examines, the following sections provide a summary of the thesis and describe the organization of the remaining chapters.

1.1 Software Development in the Medical Device Industry

In the medical device industry, software is developed under strictly regulated processes to ensure patient safety. A hallmark example of why this is crucial can be seen in the case of the radiation therapy device called *Therac-25* where poor software development practices were identified as the root cause of three associated patient deaths in the 1980s [1]. Issue tracking systems are essential tools at the heart of development, management, and quality assurance processes in this domain.

Within this industry there are a number of companies that produce a wide variety of software-intensive medical devices [2]. Pacemakers, for example, are currently prolonging the lives of more than 1.5 million people in the US alone [3]. Each of those life-sustaining pacemakers and the instruments used to interact with them has medical

device software within it. In this context, software developed by the medical device company associated with this thesis plays a part in contributing to human welfare. A recent attempt to put this in perspective noted that "every four seconds ... 16 babies are born, 8 people die of diseases, and Medtronic helps improve another life [4, 5]."

This thesis examines the issue tracking activities of engineers working together to develop software for next generation neuromodulation therapies within Medtronic. Clinicians will use the software developed as their primary means of interacting with a number of medical devices including most of the worlds implantable drug delivery systems.

High confidence in the quality of the software and other components of implantable drug delivery systems is critical. Imperfect performance of software components in implantable infusion systems can lead to death from overdose (e.g., in opiod pain treatment) or underdose (e.g., in Baclofen spacticity treatment). Consider for example, the 2004 Class I recall of Model 8840 N'Vision Programmer software related to users incorrect entry of durations that lead to at least two deaths [6]. The "need for more careful infusion pump design and testing" (FDA director cited in [7]) underscores the importance of this thesis in not only saving effort and improving the quality of products and related documentation, but also in improving product safety.

1.2 Issue Tracking

At Medtronic Neuromodulation, the Production Neuromodulation Issue Tracking System (PNITS) is used to track issues (aka PNITs) as depicted in Figure 1.1 and Figure 1.2. Issues describe concerns and work related to products and the process that produces them. Even once closed, issues represent a chain of evidence that is important to the business and to regulatory bodies like the Food and Drug Administration (FDA).

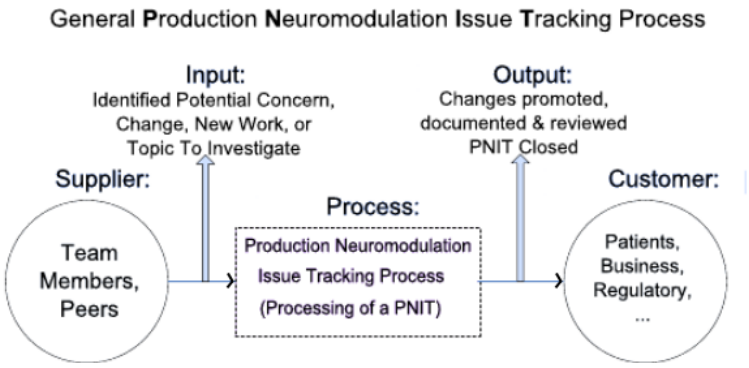

Figure 1.1: General "PNIT" issue tracing process

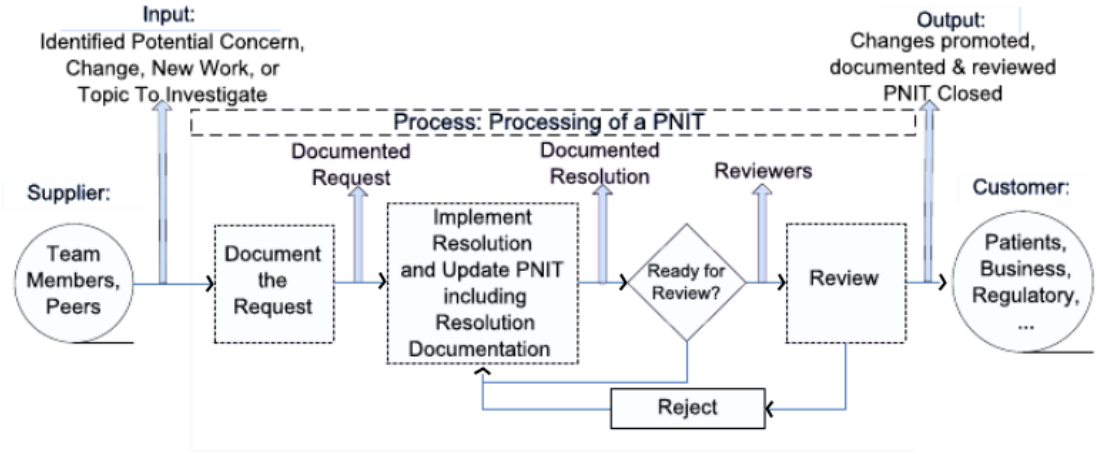

Figure 1.2: Process of addressing a "PNIT" issue.

The process of dealing with a particular issue (see Figure 1.2) involves not just completing the particular product-related work being requested, but also documenting the story of what the concern was, how the concern was resolved, and how the issue was reviewed. First the concern, or request, is documented and 'submitted' as an issue in the issue tracking system. The concern is later resolved and details of that resolution, related activities, and the plan for its review are documented in the issue until the resolver identifies the issue as ready for review. Next, reviewers examine the issue and any related artifacts before either rejecting the issue or sending it on for further review or closure.

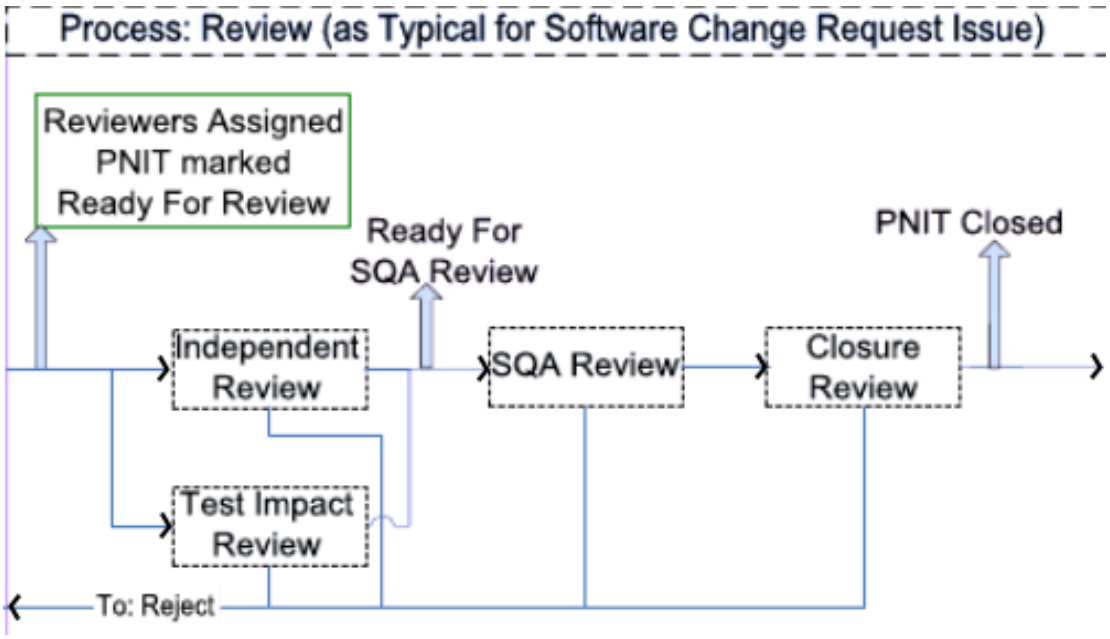

Figure 1.3: Process of Reviewing a Typical "PNIT" Issue.

Steps in the review of an issue vary to some degree with the team and the specific situation. As depicted in Figure 1.3, with an issue where a relatively minor change has been made to product software code, the relevant issue should typically be assigned an independent reviewer, a test impact reviewer, a software quality assurance (SQA) reviewer, and a closure reviewer. Each of these reviewers must reject or approve the issue. All reviewers must approve an issue (or be removed from its review) before it is closed. By general agreement, the SQA review is completed only after approval of the independent and test impact reviews. By strict process requirement, closure review may only be approved after approval from all other reviews.

Though aspects of the issue tracking process are spelled out in company and external requirements (e.g., [8]) as part of the broader quality system for developing medical device software, the issue tracking process is still ambiguous and its application is challenging. The details of the concerns and activities tracked by issues are highly varied and there is significant room for definition and alignment on best practices for important, but detailed aspects of issue tracking. Though issue tracking in Medtronic Neuromodulation is grounded in decades of experience, detailed practices are complex and not well

documented. So with a strong general understanding, but limited specific framework for issue tracking, the details of issue tracking activities are, to some extent, left to the dedication and experience of those involved.

It has long been understood at Medtronic that "we make implants and evidence" and that "bad documentation equals bad product in the eyes of the FDA[9]." To this end it has also been clear that "the story" told by an issue is as valuable as the associated "content" changes to the software. As such an essential part of the process, this narrative is patiently enforced under the review and professional integrity of those involved in issue tracking. Though not clearly documented, it is widely accepted that issues should be rejected if they or their associated artifacts are not consistent, concise, substantiated, complete, appropriately reviewed, and able to be understood by someone else decades in the future[9]. Despite some awareness of these general issue acceptance criteria, their interpretation and the most frequent causes for issue rejection in practice were not widely understood before the work associated with this thesis.

However, issue rejections are a crucial tool to understand and measure waste, or the effort spent on issue tracking that is not needed for satisfactory quality in *the story* of an issue and in the issue's associated artifacts. Each rejection reflects waste because it requires additional review, documentation and other costs that could be avoided if the sources of rejection were removed earlier in the process. If the causes of, and thus need for, rejection could be sufficiently reduced, the effort associated with review could be reduced. Thus, this thesis explores support that will reduce the effort required for review and address sources of rejection in this domain.

Reducing the number of unneeded issues is another approach that could lead to waste reduction in this domain, and has been a focus of some Related Work (Chapter 2). This is not, however, a focus of this dissertation.

1.3 Pilot Users and Use Environment

Prior to initiation of this thesis, Medtronic developed mechanisms for identifying software issues that related to quality and for tracking resolution of those issues during development of the product. The characteristics and performance of these efforts set the baseline for performance against which to measure potential improvements. Medtronic

has commissioned teams of experts to identify issues and oversee them through resolution to final approval. One such team is identified as the *S team*. This section introduces the S team to help provide motivating context before further discussion related to the S team and the similar *P team*.[1]

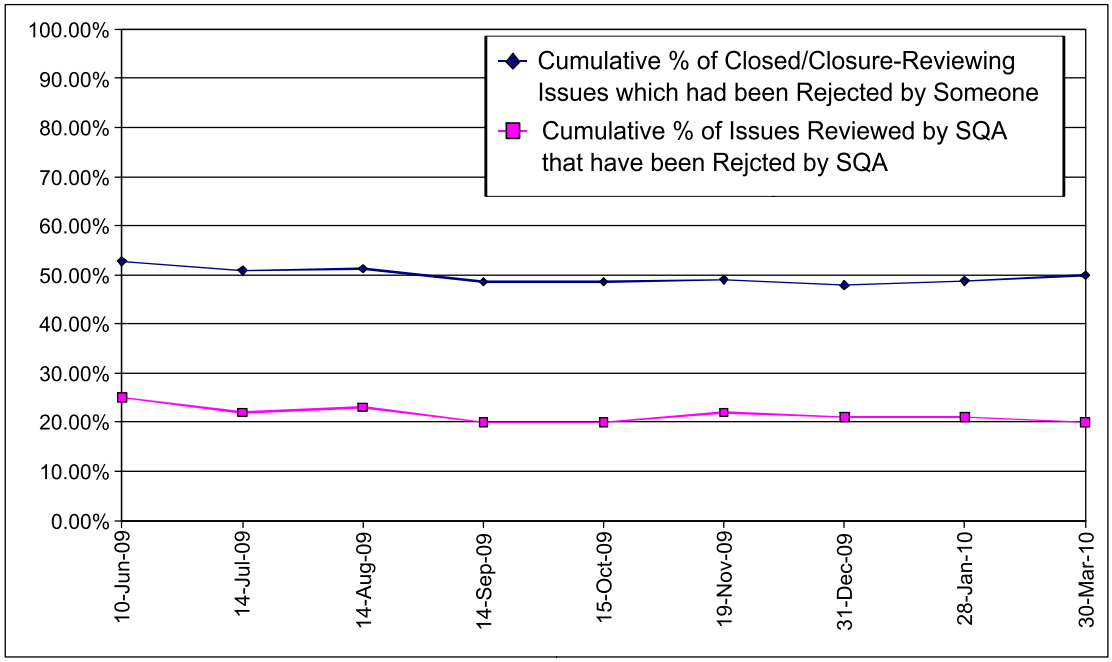

Figure 1.4: High Level Rejection Graph June 2009 To March 2010

Although individual users and issues vary greatly, it is telling to evaluate issue tracking in the aggregate. As of March 2010, the *S team*, made up of approximately 20 individuals, had accumulated a number of issues over the previous 2 years. By March 2010, 537 issues had been closed, 560 issues had been reviewed by software quality assurance (SQA) and more than 1000 had not yet received initial review by SQA (see Issue Tracking (Section 1.2)). Of those reviewed by SQA, more than 50% (268) had been rejected at least once by an individual, and more than 20% had been rejected at least once by SQA (see Figure 1.4).

Software quality assurance engineers had agreed to stop reviewing every issue and

[1] Here the real names of these teams have been replaced with S and P out of respect for the confidentiality of Medtronic and the individuals involved.

begin spot checking if the SQA rejection rate could be maintained around 5%. At about $\frac{1}{4}$ of the rate that had been achieved previously, many involved believed this to be an impossible goal. On the other hand, if achieved, this would allow SQA and other resources more time to focus on additional quality assurance activities.

SQA engineers occasionally published basic metrics including counts of issues rejected in certain states. However, no one had empirically and methodically attempted to understand the sources of rejection and there were conflicting opinions as to what these were.

All members of the S team (and P team) had received mandatory training on the tools and process of the "Production Neuro Issue Tracking System" (PNITS), but significant opportunity for improvement in understanding and executing the issue tracking process remained. In January 2010, the entire team again received training, similar to that they had received previously. In addition to various training presentations and process documents, the team used a formal Software Quality Plan that had included an appendix with a checklist for SQA review of issues.

Much of the development team was co-located[2] in a lab-environment. They held daily meetings, practiced pair programming and issue tracking, sat with the development lead that acted as the local process owner, and employed a TWiki website for distributed capture and review of detailed process description and other purposes. Much of the relevant process described on this TWiki site had been shared with SQA and another software development team that they were working closely with. However, this description was incomplete and detailed process alignment, scenario-specific application, and in context recall of the relevant details remained an area of significant opportunity. Additionally, complexities of other tools, such as the version control system used by the team, occasionally lead to legitimate, but avoidable, rejections.

For this thesis, it was not feasible to change the issue tracking system itself. To do so, even to a minor extent, would require high level approvals and the support of costly and time consuming evidence even for narrowly defined changes. Limitations on such changes are in place to maintain confidence, quality, predictability and consistency in the issue tracking system and the related processes used by numerous groups for critical

[2] In addition to occasional telecommuting, around half of the S team members worked on different floors of the same building in the United States of America or as contract verification test support in Kharkiv, Ukraine.

activities. In addition to not changing or replacing the existing issue tracking system, it was also not feasible to substantially change the governing process or to provide any direct replacement for user review or editing of issues for similar reasons.

Those involved were looking for new ways to reduce rejection rates because the up side was readily apparent. Product development would become faster and cheaper due to better alignment, fewer rejections, fewer reviews, and less rework with $100,000 or more of direct savings for each project. This in turn could have a market share effect with more rapid release and/or greater R&D reinvestment.

Improvement could also lower effort to maintain or reduce the risk of public opinion or regulatory action. As the largest medical device company in the world, the company's reputation for quality, perhaps its greatest asset, is continually at risk and can be called into question by a pattern of only loosely connected concerns. In the month following the voluntary recall of the Fidelis cardiac leads, Medtronic stock fell more than 15%.

Furthermore, although it has a number of facilities and distinct business units, Medtronic is generally considered a single organization by the Food and Drug Administration and many of its customers. Medtronic and other medical device manufacturers are audited by the FDA many times each year. In some cases, the FDA issues "observations" that can escalate to warning letters and other actions. Initial response and subsequent updates provided to the FDA following such observations can cost as much or more in personnel hours than the time to support an audit (well more than $500,000).

Given the constraints of this domain, this thesis aims to only non-intrusively analyze issue tracking activities and supplement, improve, and extend them with targeted advisor agent software and education. Furthermore, because they are not validated, but may be used in combination with other approaches, the use of the software capabilities produced as part of this thesis is strictly at the discretion of those involved with the issue tracking activities. As a result the advisor agent software had to be appropriate for user adoption (and accepting of user rejection) under real conditions. As a result, the advisor agent software needed to meet a variety of usability, performance, and other requirements including the flexibility to be used and maintained by a variety of different users with low effort.

1.4 Thesis Summary

This thesis involves the development of local theories about the sources of issue rejection, targeted support to address those, and evaluation of the impact of the support provided. Significant improvement is demonstrated and even greater improvement is linked to the use of the advisor agent software deployed to two software development teams.

This thesis involves two central postulates:

P_1 Sources of waste in issue tracking activities for software development in the medical device domain can be identified through a combination of ethnography and empirical analysis of issue rejection.

P_2 Some of these sources of waste identified as described in the first postulate P_1 can be significantly reduced through a combination of education and external advisor agent software support.

More specifically, this thesis predicts (see Detailed Research Questions (Section 3.8)) and later demonstrates (see Data, Analysis and Evaluation of Impact (Chapter 6)) significant improvements in issue rejection rates, both overall and to an even greater extent for those who use the software agent more often.

This research is intended to be valuable to those on the teams it involves directly and hopefully can be generalized to allow for improved issue tracking in the broader medical device domain.

This thesis involves ethnography and quantitative empirical analysis at Medtronic Neuromodulation from 2009 to 2011. It proposes major classes of rejections and validates them with users and through long term measurement and the introduction of targeted means to address these proposed sources of waste. For the most part, the support aimed at reducing these specific sources of waste is provided and tracked by novel software capabilities of a software advisor agent called "PnitAgent" that was created for this purpose.

This thesis reveals that more significant sources of waste were related to the contents of the issues, rather than the artifacts associated with those issues. In particular, most rejections were associated with problems in the version, resolution, related issue, and review information in the content of the issue. After successful implementation and

application of support targeted at improving each of these sources of waste, there was substantial adoption and use of the novel software capabilities of "PnitAgent." Finally, there was significant improvement in quantitative aspects of review and rejection and in the nature of rejection with greater reduction in targeted sources of waste.

This thesis explores, applies, and validates techniques that improve understanding of waste and rejection in issue tracking of complex, real world, regulated software development. It demonstrates successful conceptualization and implementation of novel agent-based information technology that promotes standardization of language and improves the quality and effort associated with targeted sources of waste. Finally, it presents discussion of this work, its relation to previous work, and the potential for further related work.

1.5 Thesis Organization

The remainder of this thesis is organized as follows:

- Chapter 2 briefly presents other work related to the topics presented in this thesis.
- Chapter 3 presents the method used to identify sources of waste, provide capabilities to address them, and evaluate the impact of those capabilities. It covers the central hypotheses and strategy of this research.
- Chapter 4 presents the primary sources of waste identified and the novel software capabilities created to address them.
- Chapter 5 presents a summary of key data collected and analysis of that data including basic narrative and statistical evaluation of potential impacts of this research.
- Chapter 6 summarizes the architecture and some key design concerns of the advisor agent software created as part of this thesis.
- Chapter 7 examines this research and its apparent impacts as a knowledge management system providing improved standardization of language.

- Chapter 8 presents a final discussion of various aspects of the research presented in this thesis and a detailed list of its contributions.

Chapter 2

Related Work

The following sections frame this research in the context of related work. This helps to ground it and bring it into focus at the nexus of Computer Science and Business, and the different communities within them. This research overlaps with Software Engineering, as it addresses software development and issue tracking, which have been a focus that community. Key aspects of this work also include ethnography and the development of novel, agent-based software to interface with and support users that relate primarily to the Artificial Intelligence (AI) and Human Computer Interaction (HCI) communities. Finally, this work aims to understand and improve business processes that involve complex knowledge management, information, and decisions within a business setting that are a primary concern of the Information and Decision Sciences community.

2.1 Software Engineering

Issue tracking and other software development practices are of central interest to the broader Software Engineering community within Computer Science. This section presents some key related work in software engineering starting with Software Configuration Management, which is concerned with issue tracking and other activities that support the controlled evolution of software.

2.1.1 Software Configuration Management

In the broader software engineering community, the term Software Configuration Management (SCM) has been used to describe "the control of the evolution of complex systems," or practically what allows "us to keep evolving software products under control, and thus contribute to satisfying quality and delay constraints," but there is "no clear boundary to SCM topic coverage" [10]. This thesis considers software change management to be "the processes for managing changes to software items and configurations in any phase of the software life cycle [11]."

Though the focus of this thesis is primarily on software, it is worth noting that the relevant application domain and more general engineering problems span across disciplinary boundaries. For example, while SCM is traditionally considered distinct from Product Design Management (PDM), at some "level of abstraction, SCM and PDM look identical," [12] and although they differ in details [13, 14, 12] it is "important to provide (where possible) a common set of support mechanisms and principles within, as well as between, the disciplines [14]." Within the medical device industry, issue tracking systems are used across various disciplines. Although this thesis primarily focuses on software, non-software components of medical devices have helped to shape and constrain the tools and processes associated with issue tracking. Therefore, non-software concerns are related to this thesis and possibly an area for its future extension.

Version control and build support concerns have been a major focus of SCM research and application, but "Software problem tracking is one portion of . . . SCM [15]." In a paper from the 2010 ICSE workshop on emerging trends in software metrics [16], Meneely shows the potential to "enrich the data gained from version control change associated with "the online discussions of 602 issues from the OpenMRS healthcare web application." This example reflects the nascent research understanding of the importance of considering data from issue tracking systems in the context of broader analysis in the healthcare domain where its not surprising that "developers were meticulous about linking change sets to [issue tracking] tickets," though it should be surprising that this practice is "optional". More broadly, the importance of process support within SCM has been recognized since the 1980s. This can be seen in the experience that Leblang and Esublier [10, 17, 18] have recounted from SCM9 where "Attendees were asked to list what they considered the most appreciated and the most deficient features in SCM.

Surprisingly, process support was number one in both categories! This indicates the progress and the frustration that process support has introduced in SCM [17]."

As Estublier notes: "Modern, high-end SCM sstems [like those used in the medical device industry] push process support even further. They do not just support change control, but allow organizations to design and enforce general development processes throughout the enterprise... Little research is expected in the future with regards to change control. Based on its wide acceptance and implementation, it is considered a mature technology. The area of growth concerns a better integration with other tools, such as Project Management [17]." The focus of this thesis is in these less mature areas at the nexus of process support, documentation, project management, quality assurance, and other areas surrounding the evolution of software and related artifacts within the medical device domain. While this thesis touches on mature topics such as version control, workspaces, change-sets, the design of issue tracking systems themselves, and even built-in support for process enforcement, it is fundamentally focused on supporting their use externally as a set of tools, activities, and perspectives that can be brought to bare as part of a more comprehensive regulated software development process.

Static (code) analysis tools and practices provide an interesting example. Issue tracking systems are, at least, closely related to SCM in the literature, in practice and in regulated software development domains. Likewise, static analysis tools including products like Coverity Static Analysis can lead to the discovery of defects and there is an expectation that "source code management (SCM)" tools "including tracking tools such as ClearQuest" be used as part of a "plan to manage those defects after finding them [19]." Similarly, static analysis has been shown to predict defect density [20] and is often integrated into continuous integration environments. On the other hand, it does not appear that the concept of static analysis as related to issue tracking has been fully explored. In particular, research associated with this thesis is the first to propose two related topics: 1) the potential for focusing static analysis given the issue or set of issues from the issue tracking system for context and 2) the potential for performing something like static-analysis on the issues themselves (retrospective, non-run-time, external analysis of issues aimed at identifying potential errors in those issues and their related artifacts). Consider for example the review of an issue before its closure where a reviewer considers both the issue and the changes made other artifacts

related to the issue. In this situation, despite the best efforts of all involved to keep mistakes out (of the issues as well as the related artifacts), it makes sense to provide some automated support for identifying them, but only within the scope of the issue(s) being reviewed. Within this area, there is not only a timely opportunity to look at the contents of the issue and artifacts with a more focused and appropriate lens, but also unique opportunity to leverage the combined data about the issue, process and related artifacts. Consider for example opportunities related to the practice of Code Annotation common in software engineering.

2.1.2 Code Annotation

Storey et al [21] examine "a number of issues related to managing annotations and ... insights into how these findings could be used to improve tool support and software process." They note the relationship between task management "embedded in comments such as TODO, FIXME, and XXX" and "issue tracking systems ... [that] contain links or references to the code" as well as automation support provided by related development environments and frameworks. However, they stop short of describing the potential for automating checking for TODO comments at the time of issue review/approval/closure. This thesis proposes and implements guidelines to help relate TODO code annotations with the related issue(s) and automated capability to present related TODO comments. This includes comments from code as well as documents that reference the issue under review or are within the scope of the issue under review (and either reference another issue that could be important for understanding the context or do not reference an issue despite contraindicating guidelines). Put into practice with the software agent associated with this thesis, this checking alone has lead to the creation of new issues and the rejections of others at the time of issue review. This may have further improved understanding of the context of the issue and related artifacts under review in other ways as well.

2.1.3 Effort Management

Prior to the last decade, the use of almost exclusively plan-driven effort management was primarily used within Medtronic (and presumably the broader medical device industry).

Recently, more agile methods of estimation and effort tracking have played an increasing role with many coming to the conclusion that more "empirical" effort management was a key part of "a better way" to pursue software development in this domain [22].

Some teams within Medtronic have attempted to exclusively use empirical approaches to effort management (e.g., through the agile backlog tool VersionOne [23]). Ultimately, however, most teams involved with significant cross-functional projects used some hybrid form of empirical and plan-driven effort management processes. More general SCM literature also suggests a hybrid approach: "Traditional project management systems are not ideal for software projects ... Specifying every tasks assignment and dependencies is required for project planning system to load balance and predict a schedule but minor delays and newly arriving work items make the schedule unstable and generally not too useful. In contrast, workflow systems use statistical methods ... based on ... the request backlog, and the average time to process a request. ... Some aspects of software projects are best modeled by workflow systems, for example, incoming defect reports; other aspects, like planned enhancements and major milestones, are better modeled by project management systems [18]."

How to most effectively find the balance between empirical and defined (or workflow and project management) systems in practice is often not clear. As the domain in question requires that work done on major artifacts (including code, review, and documentation) be tracked by the issue tracking system, the issue tracking system provides an existing mechanism and data repository for most of the major items associated with effort for software development done by teams like the S and P teams. Furthermore, while the issue tracking system contains a mix of issues associated with scheduled and unplanned tasks and may not be designed a priori to most effectively support effort management activities in a particular circumstance, it can be used for effort management in combination with the advisor agent associated with this thesis. In particular, the agent can effectively support individual and larger team levels of effort analysis, tacking and management with relatively low overhead for users. The relevant agent capabilities can support a range of activities from

1. tracking resolution and review velocities within areas of interest to

2. identifying changes and imbalances in team and individual effort backlogs to

3. reporting planned and retrospective accomplishments at a high level of detail to
4. identifying who and for what a body of work is waiting to
5. supporting collaborative teamwork through communicating requests, priorities and so on.

2.1.4 Process Support in Issue Tracking Systems

Another key area of focus relates to detailed process understanding, alignment and application for issue tracking. While significant process definition exists in SCM related tools (such as was found in ClearGuide [18] or more recently IBMs Unified Change Management, UCM [24]) and significant best practice guidelines and process definition exist for software development within the medical device industry [8, 25, 26], there is a level of detail where it is not understood how best to apply process related to issue tracking and associated activities. Indeed, more generally while detailed process support provides value, it also comes with complexity that the SCM market must struggle with how to understand and apply and so fine control general process support tools (like ClearGuide) swing out of fashion and are replaced by more simplified and removed process support (like UCM) [27]. A potentially important contribution of this research is detailed articulation of some guidelines and best practices for issue tracking and related activities in the medical device industry as well as ideas for supporting them in practice based on the experience and alignment derived from this research.

There has also been some notice of deficiencies in process support for issue tracking systems that capabilities created as part of this thesis may begin to better address. For example, some of the research associated with this thesis will tackle role support as roles in this domain are critical to the appropriate handling of issues. Delguach noted that better explicit role support for issue tracking systems is needed "if we want to reason automatically about roles and their appropriateness or legitimacy [15]."

Improved role support is of particular importance as an enabler for automation of the process of ensuring the appropriateness of reviewers. The Food and Drug Administration, for example, requires "formal documented reviews" and that related "procedures shall ensure that participants at each ... review include representatives of all functions concerned with the design stage being reviewed and an individual(s) who does not

have direct responsibility for the design stage being reviewed, as well as any specialists needed" [28]. These functional, independent, and specialist reviewer roles are important in the review of issues, but not well supported by issue tracking tools. The advisor agent associated with this thesis provides support for this externally, attempting to meet the domain need to ensure appropriateness of reviewers (e.g., in the roles of independent, SQA, and so on) for the situation of the issue at hand (e.g., duplicate, bug in test, requirement change, ...).

2.1.5 Examining Issue Contents

A great deal of work has been done in areas that contribute generally to reducing the number of unneeded issues and thereby reducing unneeded issue tracking effort. As described in the Introduction, reducing the number of unneeded issues is not the focus of this thesis. Instead, this thesis focuses primarily on reducing rejection, moving rejection earlier, and supporting the issue tracking process more directly. The advisor agent support associated with this thesis involves examining the contents of issues and has its own more limited set of prior art.

Previous research that is more closely related to analyzing the content of issues includes:

- Detection of duplicate issues [29, 30]
- Visualization [31, 32]
- Developer/contributor networks [16]
- Linguistic aspects of issue descriptions [33]

2.2 AI and HCI: Agents and Intelligent User Interfaces

In the areas of artificial intelligence and human computer interaction, a significant amount of relevant work has been done that relates to, inspires, and is carried further by this thesis. The agent and intelligent user interface communities are two areas of focus particularly relevant to the advisor agent associated with this thesis.

The software advisor agent associated with this thesis might be considered to meet many of the criteria and completely satisfy some of the notions set for agency in AI tradition and literature such those put forth by Woodridge and Jennings [34], Smith et al [35], and others. By reviewing specific agent concepts, the advisor agent software associated with this thesis can be placed in the spectrum and taxonomy of agents. However, this thesis aims to relate "real world concepts" rather than "arguments about how a word should be used" [36] and focus on "the notion of an agent" as "a tool for analyzing systems [37]."

This thesis involves the creation of distributed software for users of issue tracking systems. A user can start zero or more copies of the software to operate on her behalf. These advisor agents are meant to have a degree of persistence. The agents are also designed to (and in some cases are allowed to) operate continuously for days or weeks, but may be stopped and started by the user at any point. More importantly they act to store and retrieve data in such a way that they become personalized for their user and retain information that shapes their activities regardless of how long their current incarnation has been running. When started, the agent autonomously interacts with its data and environment to observe what capabilities are available to it and the extent to which it will need to request additional credentials from the user[1] before providing the user with support in some cases proactively, but in many cases only in response to a users request. That support involves at times acting on the file system, retrieving data from various sources, starting and interacting with other programs and windows, displaying information to and soliciting information from the users, and performing what might be called domain oriented reasoning, but never altering the issue tracking system or other product-related artifacts. With these aspects of the software in mind, it is apparent that the software associated with this thesis is close to what Lieberman calls an "advisory agent" in the book section *Agents for the User Interface* [38].

Another interesting aspect of the software is that it interfaces with other existing software through a variety of means. In this sense, it faces some of the challenges

[1] The agents persist passwords that users have provided to them for accessing issue tracking, version control and other systems with in a protected manner and automatically connect to these systems without additional user login after the first time so long as their data has not been lost or they suspect the password to be invalid (which they might detect after attempting to use an old password when the user has changed their password recently).

that Lieberman [39], Castelli [40], and others have described as associated with this sort of integration and perhaps falls into what Castelli et al. call "a class of systems that provide advanced UI functionality on top of existing applications [40]." The relevant work includes various attempts to support task modeling [41, 40, 42], "groupware procedures,"[43] automation after pattern recognition through programming by demonstration [40, 44, 39], and the like. This thesis examines related topics such as the architecture and implementation; modeling and observation concepts like caching into a "world model" [40], "dynamic negotiation of protocol between agent and application" [39] and how to "adaptively monitor" [40]; usability concerns such as graceful failure, partial automation, perceived value and other "critical barriers to widespread adoption" [45]; and various other important concerns.

Though these areas are relevant in many regards and will be examined further in this thesis, they appear to neglect some interesting and related topics that this thesis expands on, contributes to the existing literature, and applys in the medical device industry. These include:

1. the potential for leveraging user demonstration in non-procedural work such as annotation,

2. the potential for ubiquitous advisory support[2] with reduced operating requirements,

3. an applied area of context associated advisory support that does not automate tasks or involve task recognition,

4. consideration of instrumentation for certain types of support with widget and window focus as the key features in context recognition for a ubiquitous agent,

5. complex and multi-modal action control spanning the spectrum from direct and immediate user control to context appropriate option presentation and user selection to more autonomous support within the same agent, and

6. evaluation and conclusions drawn from a unique, practical, real world implementation as well as its use and impact within the medical device industry.

[2] Ubiquitous advisory support includes such things as annotation or next step suggestion across programs, technologies, and so on.

To some extent these can draw from and can be related to slightly different related areas, such as the potential for using window and widget focus in improving file system search [46] related to number 4 above or more broadly physical context awareness [47] In a previous publication [48] related to this thesis describes early prototype software and associated contributions related to items 1, 2, 3, 4, and to some extent 5 listed above. This thesis also expands on these contributions and adds significant contributions related to item 6 as it examines the deployed agent that incorporates previously described functionality as well as other work in a larger and more practical application with the S and P teams. Some of the areas this thesis examines related to item 6 include perceived and measured use and impact of deployed software, the capabilities implemented and applied, the role of this research in supporting collaborative work, practical concerns and their mitigation in this domain, opportunities and guidelines for related future work, and so on.

2.3 Information and Decision Sciences

Knowledge management has been defined as the process of acquiring and using knowledge for "other employees . . . to be more effective and productive in their work" [49, 50]. Here the challenge is access and integration [49, 51], and a key aspect of addressing this is the process of turning difficult to retrieve tacit knowledge of individuals into explicit organizational knowledge that can codified, communicated and applied across an organization [52, 53, 54]. This is similar in concept, if not fundamentally the same, as the important process referred to by Osterweil in software engineering work as "materializing [55]."

Issues tracking and related systems as well as the software advisor agents designed to support them are knowledge management systems (KMS) and knowledge management is an area in which this thesis aims to make contributions. This thesis takes key steps to gather, align, and transform tacit knowledge about how to do issue tracking and software development within the medical device industry into explicit knowledge and KMS that can be applied by Medtronic and others more effectively and productively. To the extent that some of this transformation of knowledge is achieved and driven by software (artificial intelligence/agents) this thesis adopts a slightly broader, less (human)

employee focused definition of knowledge management: knowledge management is the process of acquiring and applying knowledge to allow for more effective work.

Hahn and Subramani [50] identify a framework for knowledge management systems with dimensions of the locus and degree of a priori structure. The nature of a priori structure and locus of knowledge in the PNITS issue tracking system and related systems are diverse and range from issue specific details and individual judgments encoded in free-form text fields and (at least initially) undocumented and inconsistently applied rules for reviewing and rejecting issues to required selection of predefined classifications and automatically encoded in formation (such as review information including pass or reject, time, reviewer and so on). Similarly, the existing system of support for PNITS and related systems is a similarly complex mixture including, for example, individual knowledge, documentation, and Twiki and other websites.

Despite investments of this type to support software engineering more broadly, research on the information needs of developers concludes that their "most frequently sought information included awareness about artifacts and coworkers . . . [and] developers often had to defer tasks because the only source of knowledge was unavailable coworkers [56]." These needs are substantial and interrupt the work of developers. Research [57] suggests that such "interactive activities" (1) occupy more than half of a developers day, (2) are often unplanned, and (3) often involve those who may have left the team recently but who are still available. Furthermore, there is the challenge of desktop-based communication being "too slow" with the need to "type out a coherent description of the problem." These suggest that a context-aware, agent-based solution could reduce context capture and communications time in a new modality for minimally interruptive collaboration that is still relatively robust to synchronous human resource availability. Baysal et al. [58] identified this problem noting that their "analysis of developers who use [the issue tracking system] Bugzilla heavily revealed . . . challenges maintaining a global understanding of the issues they are involved with, and that they desire improved support for situational awareness that is difficult to achieve with current issue management systems. . . [highlighting] the need for personalized issue tracking . . . together with improved logistical support for the tasks they perform."

Before personalizing and bringing support into context for developers though, more detailed alignment is needed. In particular, the S and P teams faced a general lack of

consistency and definition at a practicable level of detail, the challenges described by Hahn and Subramani [50] as separation of groups, complex artifact trajectories, and what and what Ren et al. [59] call "technological and political barriers to integrate ... systems." Together these appear to have lead to somewhat fragmented communities of practice with limited consistency in their understanding and language tracked by a high degree of deviation evident in the documentation and review of issues that has in turn lead to extensive rework, waste and frustration.

To the extent that a key goal of this thesis is to reduce waste (lean) and deviation (6-sigma), lean sigma practices are also related. Discussing six sigma and the DMAIC (Define, Measure, Analyze, Implement, Control) process, others [13, 60] have examined aspects of the nexus of lean six sigma and knowledge management noting their complementary and overlapping nature. This overlap is partially related to the extent that the processes in question (e.g., issue tracking) can be defined, measured, aligned on, streamlined and so on. A key aspect of the software and work associated with this thesis enables measurement and support of the process in a way that fundamentally enables lean sigma to be applied to it.

Chapter 3

Research Approach

Previously this thesis has described the high level postulates that additional sources of waste could be revealed by issue analysis and ethnography (see P_1) and then reduced by new external support (see P_2). This chapter presents the strategy for examining these postulates. It then provides a more detailed synopsis of key aspects of this strategy and how it was implemented from early baseline and development, to formation of a "PNIT Tiger Team", to completion of formative rejection analysis, to concerns related to validation and ethnography. It also introduces some more detailed research questions and briefly discusses some variables and some potential threats to validity.

3.1 Central Strategy

The basic strategy employed for examining our postulates (P_1 and P_2) consists of:

1. **baseline and development** - collection and review of baseline data in the issue tracking system and development of context and tools expected to be useful later,
2. **rejection analysis** - collection and classification of a sample of rejections,
3. **validation and ethnography** - validation of rejection classifications and formative discussions with a group of users,
4. **identification of sources of waste** - analysis to identify themes that define broader potential sources of waste,

5. **design support to reduce waste** - design and prototyping of new software capabilities to reduce sources of waste that users can adopt,

6. **deployment of novel support** - broadly announcing new software and providing training in the hopes they are adopted and measurably address the identified sources of waste,

7. **collection of data** - collection of data from the issue tracking system and the new software associated with this thesis (e.g., their use of new software capabilities),

8. **evaluation** - analysis and evaluation of adoption and impact of capabilities as well as change in issue tracking (e.g., in the timing or nature of rejection or sources of waste),

9. **conclusion** - discussion of the outcomes, themes, and broader implications revealed by this thesis.

3.2 Two Related Postulates

This section presents a brief overview of the relationship between the two postulates (P_1 and P_2) and a synopsis of the approach intended to explore both of them.

To examine P_1 and measure whether sources of waste can be identified through a combination of ethnography and empirical issue rejection analysis this thesis describes the application of those techniques to the issue tracking of two teams in the relevant domain to identify potential themes in sources of waste and later examine apparent change in those themes. More specifically, a sample of rejections is taken, classified, reviewed with an experienced software quality assurance engineer and, along with related topics, discussed extensively with a number of individuals involved with the work. The classes into which the rejections are partitioned represent potential themes or sources of waste. To examine P_1 these potential themes are reviewed to determine if they might be valid abstractions that represent sources of waste. Some evidence supporting their validity came in the form of affirmation and acceptance during discussions with software quality assurance engineers and others involved.

Education and software advisor agent capabilities aimed to address the potential sources of waste are introduced and, *after* this, a second sample of rejections are examined along with other issue tracking and use data to determine if the potential sources of waste were in fact reduced. By limiting other changes and showing that these potential sources of waste were mitigated, we provide evidence that the potential sources of waste are real (further supporting P_1) and that the education and software advisor agent capabilities contributed to the improvement (supporting P_2). More detailed measures involved with this study, including those (e.g., H_2) that examine the role of greater use of the agent capabilities in reducing waste in the targeted areas, are described in Detailed Research Questions (Section 3.8).

The two postulates (P_1 and P_2) are supported by the work related to this thesis, and, perhaps, may generalize to other situations.

3.3 Timeline

This section presents a brief overview of some key time periods and events that are relevant to this work illustrated in the Overview Timeline (Figure 3.1).

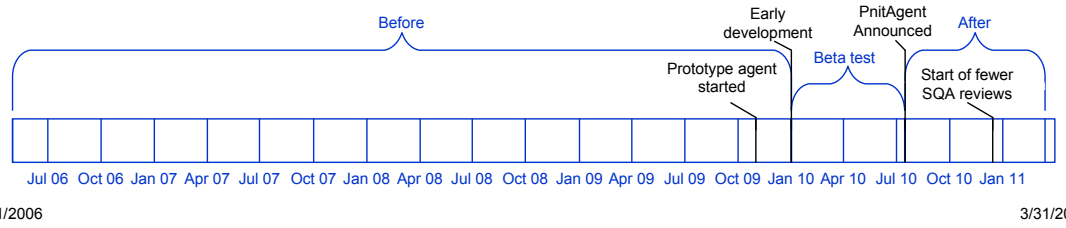

Figure 3.1: Overview Timeline

As the timeline illustrates, there are three key ranges of time relevant to this thesis: *before*, beta test, and *after*. These periods represent the time ranges *before* any influence of this work on the S and P teams, a period during which the agent and the results of its analysis were visible to a small group on those teams (some of whom helped to direct its evolution), and finally the period *after* the agent was announced and made available to everyone on the S and P teams. This thesis will compare the *before* and *after* periods as a central means of evaluating this research and the agent developed as

part of it.

In addition to the 'early development' and 'PnitAgent announced' events delineating the *before*, beta test, and *after* periods, two other key events are shown:

1. the start of early prototype work discussed in Subsection 3.4.1 that eventually was the foundation for the agent that was developed during the beta test period that is refered to as PnitAgent and is described in Subsection 3.4.2

2. the date when software quality assurance engineers working with the S and P teams chose to spot check rather than exhaustively review issues (reflecting the remarkable improvement in the rate of rejection).

3.4 Establishing Context and Early Development

A key first step in identifying sources of waste, and what kind of support users will accept to address those sources of waste, is to start the process of understanding the work, users, and environment in which they exist. A challenge in understanding the users and their work is the lack of a clear, common and sufficiently detailed definition of the relevant processes, best practices, and expected outcomes. After numerous informal interviews, group discussions, review of existing support means (documents, procedures, Twiki pages, and so on) and consideration of the outcome of thousands of reviews of issues and related artifacts it is apparent that one key reason for review rejections is lack of alignment on the details of the best practice, process, and acceptance criteria for the artifacts produced. This thesis aims to facilitate this alignment and ultimately produce support that incorporates the accepted best practices that are identified. It was clear there was an opportunity to begin work towards understanding, defining, supporting and communicating expectations, guidelines, and practices that could be applied across projects, issue types, disciplines, and so on.

Hoping that sources of waste could be identified and new, context appropriate support to address these could be developed, prototyping and exploration of tools began. The development of support focused on building external, agent-based software capabilities. Early work produced a framework for additional development, established early feasibility of practical use of such an agent in practice, and provided the tools to facilitate empirical analysis of issues and their rejections.

3.4.1 Prototype Meta-Software Agent for Issue Tracking Help

In November 2009, we began to explore the potential for issue tracking support observing only two key features, the window and widget in focus, while providing generalizable support for users ubiquitously across a variety of software technologies. Early prototyping involved development leveraging the UIAutomation framework from Microsoft as well as the Mozilla Firefox browser and the MozRepl plugin. The prototype agent software was able to provide ubiquitous and easily generalizable annotation support as well as basic sequential support that would predict/recommend the next window/widget context based on the current one. This prototype work was applied to the native Microsoft Windows, Eclipse, and web-based interfaces of the ClearQuest PNITS system with encouraging results. It was also found to be generalizable beyond the issue tracking area of application. The early software functioned seamlessly in numerous programs including the Explorer windows as well as the start menu and showed promise for nearly universal support for the annotation and context recommendation in the Microsoft Windows environment. It was also found to be rather simple to maintain and install once setup. More details on this research were published in AAMAS 2010 [61].

On the other hand, the intentional generality of the prototype and limitations in what and how it could interact with other software revealed challenges to its application in a practical setting. It was dependant on the users it observed to get information through their activities, requests and interactions; it was intentionally kept from interacting directly with other software. It could provide basic support well (annotation recall and focus recommendation), but no specialized forms of support such as information extraction, examination and presentation of related information, analysis and warning about potential defects, and so on. It also raised potential privacy concerns (always watching the user) and was not designed for easy deployment or setup. Looking for an additional approach with these shortcomings in mind, work began on the initial PnitAgent software that was eventually deployed.

3.4.2 Early PnitAgent Software

Extending the Prototype Meta-Software Agent for Issue Tracking Help (Subsection 3.4.1), we built more complex and practical agent software named PnitAgent. Initial work on

PnitAgent was aimed at macro-level analysis of collections of issues to facilitate understanding and large numbers of sequential reviews. The agent provided a basic framework for tightly coupled support for rapid analysis of issue contents and some combinations of this with other capabilities. PnitAgent was not deliberately deployed to any users, but shared only with a small group of users that helped its early development. It was first shared casually with an individual developer in response to some related work and discussion. PnitAgent benefited greatly from related discussion, work, and eventually use and feedback from a few other users.

PnitAgent was extended to help analyze and generate data that could be pasted into MS Excel and tracked across periods of time with users specific context awareness, settings files, and so on. The agent was eventually extended to support Rejection Analysis (Section 3.6) in which all rejections occurring over a period of time could be identified, extracted, and partially classified. The new capabilities helped to automate some issue tracking related metrics and analysis tasks that were being performed by SQA. This software support drove a greater degree of definition and alignment across related activities for the S and P teams described in Pilot Users and Use Environment (Section 1.3). Referring to some of the new software agent capabilities in a leadership meeting on April 13, 2010, a software quality analysis engineer said "This will save us at least 3 person-weeks over the next year, ... I've helped the other SQAs install it."

3.5 The PNIT Tiger Team

A new PNIT Tiger Team was formed in early April 2010 to better understand and address 'PNIT debt' and practices as applied on the P and S projects. Some attempts had been made at various points in 2008 and 2009 (and for projects prior to and overlapping with those of the P and S teams) to define issue tracking practices/expectations in formal documents and other means such as in an appendix of the Software Quality Plan where a checklist for SQA review of issues was captured. Unfortunately, these were not based on any significant empirical evidence, were not comprehensive, and were difficult to apply consistently.

It was widely believed within the PNIT Tiger Team that most of the reasons for rejection were at a higher level of detail than that addressed by the available training

and formal documentation. On the other hand, many on the team believed improved support could contribute to the teams alignment and level of performance.

Early on, opinions were mixed in the PNIT Tiger Team as to what the sources of rejection were. This question and related discussions, rejection analysis, and finally what to change, were the focus of a series of 15 meetings with the new PNIT Tiger Team from 4/8/10 to 7/12/10. A few days later, the agent was announced and related training provided as described in PnitAgent Announcement, PNIT Tiger Team Findings, and Related Training (Section 4.5).

One early topic was the degree to which issues went through "content rejects" where the rejections were caused by problems in the source code, document or other associated artifacts rather than the fields of the PNIT itself. In contrast to what one might expect in less regulated domains, several on the team stated that they believed content rejects were by far the minority. Evidence, that is presented in Rejection Analysis (Section 3.6), supports their assertions. To this end, a useful tool in identifying potential sources of waste has been the methodical and exhaustive classification of rejections over a period of time.

3.6 Rejection Analysis

3.6.1 Rejection Classification

First, using the agent software developed for this thesis, details of rejections were extracted from the issue tracking system. The set of rejections was bounded to include only the issues associated with the S and P teams. Furthermore, the set was constrained to a particular window in time. As a formative means for understanding the rejections more broadly, SQA rejections over the last week were examined each week generally without any additional classification or discussion.

The more significant basis of formative rejection analysis was performed over a longer window of time in March 2010 looking at all types of rejection where each rejection was more formally discussed and classified through discussion amongst a team of expert users. The findings of this effort are described in Identifying Sources of Waste (Section 4.1). This group of expert users was part of a "PNIT Tiger Team" organized to understand and improve use of the Production Neuromodulation Issue Tracking System

at least within the S and P teams. Its members included two principal software quality assurance engineers, a project manager, a functional manager, the development lead of both teams, a developer, a human factors engineer, the verification test lead of one of the teams, and another verification test engineer. The team's consensus in classification of the rejections was achieved through brief discussion of each of the numerous cases.

The basis for summative comparison of change in the nature rejection was performed over a longer window of time from September 2010 to March 2011 looking at all types of rejection where each rejection was discussed and reviewed with a principal software quality assurance engineer who had participated in the original "PNIT Tiger Team." The findings of this effort are described in Change In Rejection Classification Data (Subsection 6.1.8).

3.6.2 Rejection Rate and Timing Analysis

In addition to Rejection Classification (Subsection 3.6.1), a record of rejection and other issue tracking activities are provided in abundance by the issue tracking system itself and the agent software created to support. Data from these records is useful for exposing rejection in the broader context of the issue tracking work. This thesis aims to not only expose themes in classification of rejections, but examine the potential for new capabilities to reduce these sources of waste. Additional data was collected and used to test the hypothesis that a positive impact was achieved, as evidenced by change in rates of rejection, usage of the new support, and timing of rejections. This is discussed in more detail in Data, Analysis and Evaluation of Impact (Chapter 6).

3.6.3 Software Quality Assurance Rejection Analysis

Often in Rejection Classification (Subsection 3.6.1) and other rejection analysis, it is useful to consider just those rejections that were "SQA Rejections" or rejections performed by a software quality assurance engineer. As discussed in Issue Tracking (Section 1.2), SQA review is performed after other independent and test reviews. Software quality assurance engineers who perform these reviews are responsible for reviewing a large number of issues and focus primarily on process rather than product content. SQA reviewers are expert process reviewers and their reviews can be interesting to look at

first or separately due to the smaller number of reviews, their process focus, and their late position within the overall review process.

Through discussion with the "PNIT Tiger Team" and various others, it was generally understood that defects in the product under development itself were often found in other activities such as design, implementation, review meetings, or test development (as opposed to review of related issues in the issue tracking system). Most issue rejections in development and to a greater extent in SQA reviews were understood to be due to problems in the story told by the documentation or gaps in understanding of processes relevant to the situation at hand for a particular issue.

3.6.4 Peer/Self Rejection Analysis

After segregating out SQA rejections as discussed in Software Quality Assurance Rejection Analysis (Subsection 3.6.3), what is left are so called peer and self rejections. A rejection is considered a self rejection if the rejecter and the resolver are the same person. This encompasses rejections as a result of a resolver realizing, often very quickly, after marking an issue as ready for review, that they want to make some changes. It also includes rejections where the resolver has been informed outside of the issue tracking system about deficiencies in their work found during the review process by another reviewer and decides to, or is asked to, reject the issue they had previously resolved. There are also more complicated scenarios, such as those that involve multiple owners of the same issue over time. Due to this complexity, self rejections are not separated from peer rejections for the purposes of this work. Peer rejections include development, test, closure and other reviews, except for SQA reviews, where the rejecter and the reviewer are not the same person.

3.6.5 Finding Themes in Rejection Classification

Although Rejection Classification (Subsection 3.6.1) provides interesting data on the sources of waste in issue tracking, the relatively unconstrained number and granularity of potential classifications of rejections does not necessarily lend itself to exposing larger patterns in frequent high-level classes of rejections. This work identifies such major sources of waste, extracting more macro level themes from the detailed classifications of

SQA and other rejections. Themes were proposed after discussion and experience with the issue tracking process and those involved with it on the S and P teams. Subsequently, rejection themes were discussed and aligned on with the "PNIT Tiger Team." Although more detailed classifications and other data are presented in Identifying Sources of Waste (Section 4.1) and elsewhere in this thesis, their abstraction into a hand full of more general themes is intended to support more general analysis and conclusions.

3.7 Validation and Ethnography

Certain aspects of this work require extensive input from those with the greatest collective expertise in this area; those who are actively working with the issue tracking tools and process in the medical device domain. Key areas where such user centered involvement is essential are 1) the identification of sources of waste and 2) the design of capabilities to reduce waste in the issue tracking activities that real users would actually adopt and find useful. Uncoerced formative inputs, long term actual use, and summative feedback from real users is an essential part of this work's strategy. These are intended to reduce individual and external bias, validate and inform decisions made in implementation and deployment, and support evaluation of the impact of the attempts made to reduce the major sources of waste. The degree to which the proposed sources of waste are successfully moderated in turn validates the sources of waste first identified.

While the involvement of users is central to the success of this work, it should not, on its own, be considered the sole source of any improvement measured. One key reason for discounting the 'Hawthorne effect' [62] where the performance of the issue tracking participants might be improved simply in response to their knowledge that their work is being monitored is quite simply that they and their work would continue to be monitored regardless of this work. For example, not only does the team regularly have their work submitted to review as part of the issue tracking process, but their leadership, software quality assurance engineers, and others regularly monitor and discuss their work.

Similarly, a reason for discounting the 'Pygmalion effect' [63] in which the performance of the participants is improved by the higher expectations of their observers is that their observers expect improvement and exceedingly high performance from all of those involved regardless. Additionally, this thesis presents evidence suggesting that

the improvement of those who use the capabilities provided improved at a similar or greater rate than those who did not despite the fact that they may already have higher performance and thus lower expectations for improvement.

3.8 Detailed Research Questions

This section focuses the two high level postulates of this thesis into more detailed research questions (H_1, H_{1a}, ...) that were formulated before analysis of the data aimed at testing those hypotheses. In other words, to help examine if sources of waste could be revealed by issue analysis and ethnography (see P_1) and then reduced by new external support (see P_2) aimed at addressing them, additional, more directly testable hypotheses are articulated that, if true, provide evidence the high level postulates (P_1 and P_2) are true as well.

An initial question relates to whether the rate of rejections is reduced *after* support to reduce the sources of waste was provided. If the major sources of waste can truly be identified and some of them addressed without creating new sources of waste, then the rate of rejection should be reduced.

H_1 The overall rate of rejection will be reduced in the period after the introduction of the education and external advisor agent software.

This establishes a central causality research question to examine if the introduced support causes improvement in rejection when compared with the frequency and distribution of rejection in the *before* period.

This thesis also examines how reduction in rejection breaks down across different types of review (e.g., SQA rejections or closure rejections). It seems apparent that if the major sources of waste that were identified and targeted by education and agent support were causing rejection in a certain type of review, then rejections associated with that type of review should be reduced. Furthermore, since this work aims to identify and mitigate sources of rejection across the different types of reviews, and not introduce additional sources of rejection, overall rejection should be reduced.

H_{1a} The rate of rejection from software quality assurance reviews will be reduced in the period after the introduction of the education and external advisor agent software.

H_{1b} The rate of rejection from closure reviews will be reduced in the period after the introduction of the education and external advisor agent software.

Potential users of the agent software support have the freedom to choose if, and to what extent, they will use the software agent to support their issue tracking work. As random selection/control of users would not be consistent with business practice, would affect the perceptions of those involved, and would be unethical assuming the support improves rejection as desired. An interesting exploratory/existence question, therefore, is if and to what extent the agent is used in the period after the agent's availability. It seems apparent that if the agent sounds, and is later still considered, useful by users, they will not only try the agent, but use it multiple times and may influence their peers to use it. It seems unlikely that all users will try the agent or that those who do will use it to the same extent. Instead, it seems reasonable that the potential users of the agent will be able to be divided into groups by how extensively they use the software advisor agent support.

Furthermore, if the agent is not used or not used regularly by some users, it will have less opportunity to directly support their issue tracking and help them reduce their personal rejection rates. Though users that rarely or never use the advisor agent may indirectly benefit from it (e.g., by learning about it or through feedback from others who use it) this benefit should be reduced and therefore the rate of their rejection should be the same or higher and the degree to which their rate of rejection improved from before should be the same or lower.

H_2 Those who use the software advisor agent to a greater extent will have a lower overall rate of rejection.

H_{2a} Those who use the software advisor agent to a greater extent will have a lower rate of rejection from software quality assurance reviews.

H_{2b} Those who use the software advisor agent to a greater extent will have a lower rate of rejection from closure reviews.

H_{2c} Those who use the software advisor agent to a greater extent will have a greater improvement in their rate of rejection from before the introduction of the external advisor agent software to after.

By examining this more specific set of causality questions focused on the role of agent use in improvement, this thesis begins to approach the design question as to what an effective way to support issue tracking is.

With fewer rejections per issue due to the positive impact of the work, the total number of reviews (including rejects) should decrease (as described in H_1). More interestingly though, as the agent and education provided by this research improves the understanding of what reviewers/roles should be included in review, a source of variation in the number of reviews should be removed (though different circumstance may still require different reviewers). Thus, the degree of deviation in the types of reviewers included with reviews may be more directly and appropriately dependent on the distribution of different issue types. Members of The PNIT Tiger Team and others had shared that they believed test impact reviews were not always being requested when they should be. If the agent helps identify missing test reviewers (as it is designed to), and the relative proportion of reviews requiring test reviewers does not decrease, the average number of test reviews (both total and those not associated with rejections) should increase.

Additionally, assuming the agent and education support provided are highly effective, each issue should, on average, be rejected less frequently (lower rejection rate discussed earlier) and be rejected earlier in the review cycle. In order to isolate a measure of issues rejected earlier in the review cycle, one could look at the average number of other reviews before each rejection. The average number of reviews preceding a rejection should be reduced in the agent.

Finally, the nature of the major sources of waste is important to consider. The major sources of waste identified and targeted by this work should be reduced, not just in total rate, but in their relative proportions of the different classes of rejections from before the introduction of the education and external advisor agent software to after.

3.9 Variables and Validity

It is useful to briefly examine the variables of this study and potential threats to its Internal Validity (Subsection 3.9.1), External Validity and Reliability (Subsection 3.9.2) as well as Other Measures of Validity (Subsection 3.9.3). The independent variable

is the training and agent capabilities introduced, the dependent variables relate to issue tracking performance, and the observed differences relate to improvements in issue tracking performance.

3.9.1 Internal Validity

Considering the threats to the internal validity of this study, or rival explanations and concerns about the correctness of its conclusions, is of fundamental importance.

First, it is useful to examine the potential for ambiguous temporal precedence in this work. In doing so, this thesis examines the extent to which it is clear that the cause (training and agent capabilities) precedes the effect (observed improvements in issue tracking performance). One obvious concern is that of partial exposure of tools or training materials while they were being developed. To address this concern the beta-period, during which this material was being developed but had not yet been presented, has been removed from consideration leaving only the surrounding *before* and *after* periods in the evaluation. By comparing just these periods, it is possible to see if the improvement occurs in the period after the change in independent variables. Furthermore, a variety of training had been available to the team over the course of the before period, but that did not change in the beta-test or after period other than through the efforts of The PNIT Tiger Team and agent as part of this work. Some of those involved with the PNIT Tiger Team and this work also worked on issue tracking activities throughout the time in question (e.g., both *before* and *after*).

It is also important to consider confounding, or the possibility that the effect might be caused by a change in a variable other than the independent variable.

Fundamentally, the details of each issue in issue tracking are slightly different and it is worth considering how this might explain improvement. To help address internal variation that might occur between the before and after populations of issues, we consider common attributes (e.g., rejections) of a population of issues and consider issues from the same very long running projects. Similarly, the after period was stopped before either project entered a different phase or the general nature of the work changed. Though the issue tracking interfaces themselves were kept basically constant, the agent was introduced and tracked as a key independent variable in the experiment.

Another potential concern might be education or metrics feedback, other than that

provided as part of this work, may have caused improvement. As one control, there was no other education after the early before period. As a second, reporting of high level rejection and other issue tracking metrics continued in the after period the same as in the before period. Basic monthly rejection rate statistics were shared throughout the time in question (e.g., both *before* and *after*).

Similarly, changes in the process of issue tracking could result in improvement, but there were no relevant process changes after the early *before* period. Alternately, changes in the issue tracking or other development tools (other than the agent that was part of the independent variable) could result in improvement, but only ongoing infrastructure maintenance and security patches that were evaluated and allowed because they do not significantly impact the issue tracking work.

Another factor to consider is the potential for selection bias, or that differences between groups might be responsible for improvements. Those involved with issue tracking were allowed to use the agent as, and to the extent, they chose and continued to receive rejection feedback through their issue tracking activities. Another selection bias concern relates to the idea that those who chose to use the agent may not have seen greater improvement than those who did not, but simply have performed better all along. To examine this, we normalized improvement of users and others against themselves for comparison of improvement.

It is also important to consider history as a threat, examining events outside the study or between the before and after measures that may have caused the effect. The concern here is that changes in corporate structure, posture or status could have caused improvement. In this case, there appears to have been no significant changes in these areas. For example, there were no significant changes in leadership, organizational structure, regulatory inputs, process, or in related teams.

With regards to maturation, it is useful to address the notion that changes in the subjects (those involved with issue tracking) between the measurements (i.e. the *before* and *after* periods) occurs and may be a result of the passage of time rather than change in the independent variable. More specifically one concern is that temporary changes in those involved with issue tracking may have caused improvement. To help control this, the before and after samples were taken over relatively long periods. Another concern is that permanent changes such as the accumulation of experiences (not as a result of

the change in the independent variable) in those involved with issue tracking may have caused improvement. However, there does not appear to have been a substantial trend of improvement in rejection rates for a number of months in the before period, but there was between the before and after period. Additionally, the improvements observed were greater in those who used the agent to a greater extent.

As relates to repeated testing (testing effects), the repetition of similar testing may have lead to bias. More specifically, it is important to address the concern that those involved with issue tracking know that they are being tested and improve as a result of that knowledge. In this case, however, there was awareness that those participating were being monitored/tested throughout the study. All issues were reviewed and rejections tracked during all periods of the study. See also related discussion in Validation and Ethnography (Section 3.7).

A related concern is that learning or accumulation of experience with repetition of issue tracking work may have caused improvement. In addition to what is described with respect to maturation concerns, this was controlled by the facts that the *before* period had already exposed users to extensive repetition of issue tracking work and that the details of issue tracking work varies from issue to issue. It is also important to recognize that these sorts of testing effects may have magnified the impact of the changes to independent variable (e.g., with learning and reinforcement in those being rejected by others who used the agent) and that this is not inconsistent with the hypothesis.

When considering instrument change (instrumentality), the concern is that changes in instruments or observers involved may cause the effect rather than the change to the independent variable. There were no changes to the instruments in this case (see related early description of confounding concerns about changes to the issue tracking or other development tools earlier in this section). The other aspect of this threat is the concern that changes in the observers may have caused improvement. In this case, human observers were largely out of the loop with most of the observation automated by the issue tracking system and agent software systems (which were not changed as described earlier). It is not clear if changes in those involved with classifying the before and after samples may have affected classification, but this would not affect other measures such as rejection rates.

Another threat is regression toward the mean or the idea that outliers in testing

or selection may cause sufficient bias that the measured effect is due to regression to the mean rather than change in the independent variable. One concern is that outliers in the issues sampled in either the *before* or *after* period may have caused apparent improvement in issue tracking performance. However, there were not any apparent extreme outliers (e.g., issues that were rejected many multiple times) in either sample and relatively large samples were used from both periods. Another related concern is that selection of unusual participants in one of the periods may have caused apparent improvement, but in this study participants remained largely constant in both samples.

Another type of threat is that of differential attrition or mortality, where the drop out of participants may have caused improvement, but the participants remained largely constant in and across both samples.

Another sort of threat relates to selection-maturation interaction where improvement might be explained by differences in age or age-categories. In this case the concern is that the relative age of participants in the after period to those in the before period may have caused improvement. There was a wide range of age (mid 20's to late 50's without known specifics) maintained across both samples though the average age in the after period presumably would have been slightly higher due to the passage of time between and during samples.

Another concern is diffusion or that a spread of effects of the independent variable may cause a lack of apparent change despite the effect of the independent variable. Or in this case, as participants work together, including through rejection driven norming in the issue tracking system, effects of agent use spread to those who use it to a lesser extent and thereby the effect of the agent may be diluted. This is believed to have occurred, but a measurable difference in effect with greater use of the agent was still observed.

The potential that the control groups may alter as a result of study (potentially masking, adding to, or offsetting the apparent effects of the independent variable) is also a type of threat to internal validity. On the other hand, it is not clear what specific concerns there are in this quasi-experiment as the participants are allowed to readily choose the extent of their use of the agent, all are exposed to training, and those involved with the before period were involved with the after period.

Finally, another potential threat is experimenter bias where those conducting the

study inadvertently affect the outcome by behaving differently with members of different groups. Along these lines it is useful to note that the degree to which different users used the agent was collected by the agent and not visible or considered until after completion of the study. Also, some differences in behavior were noted in the after period (most notably the stop of SQA review of most issues), but these were a result of reaching pre-arranged rejection rate thresholds that had been set and discussed early in the *before* period. Finally, those involved were not aware that the effect of the independent variables was being measured, but were aware throughout the before and after period that issue tracking performance was being measured more generally.

In the context of these considerations, there appears to be reasonable support for internal validity especially as the make up of the teams working with these issues, general subject matter to which the issues relate, the process under which they were developed, and the issue tracking system in which they are held remained largely the same. Furthermore, the after period was stopped prior to significant ramp down in the team size. Additionally, as discussed previously, this work aimed to allow some normalization of agent users and others against themselves in providing comparison of their improvement (in terms of rejections). Finally, in data analysis, the users and issues are treated as samples of the larger population and considered in terms of averages, standard deviations and statistically significant changes.

3.9.2 External Validity and Reliability

External validity relates to the extent to which the conclusions of this work may generalize. This work may not generalize to other groups as it had unique features (involved a small number of people and projects within a particular division of a particular company) and agent use had idiosyncratic features (use was voluntary) as is common with experiments involving human participants.

Similarly, reliability relates to the extent to which the same results could be replicated by other researchers, and is limited in part by the unique features of the work. Though everyone in the company has some natural stake in the agent and training developed as well as the success of the team, we have aimed to minimize these sources of bias, even when in some degree of conflict, for example in not further improving the agent during the *after* period.

Rosenthal or Pygmalion effects (limitations on generalizability to other investigators or researchers) and reactivity (effects as a result of studying the situation) are described in Validation and Ethnography (Section 3.7).

Despite the threats to the extent to which the conclusions of this work may generalize, it may inspire future work that will show this generalization post hoc and perhaps more importantly in some way leverage and extend its benefits.

3.9.3 Other Measures of Validity

As referenced by [64], we address several other measures of validity including: triangulation with multiple data sources (e.g., PnitTiger team, rejection classification, and quantifiable rejection data), member checking (e.g., PnitTiger team), clarification of bias, reporting of discrepant information, prolonged contact with participants, limited peer debriefing (i.e., review by internal software quality assurance engineer and academic committee members), problem and participant authenticity, appropriate organization access and intended change, and clear outcomes for the participants.

3.10 Alignment with Research Stances and Approaches

As described in [64], it is useful to clearly articulate how a piece of empirical software engineering research aligns with major philosophical stances on and approaches to empirical research.

3.10.1 Philosophical Stance

This thesis is most consistent with pragmatism, aiming to solve a real problem, acknowledging that any knowledge gained from this research is approximate and incomplete, and applying mixed methods research to achieve practical improvements within its specific area of application. This thesis also aligns to some extent with other major stances. Though this thesis is not isolated from its context and does not align as well with post-positivism, it is structured and examined with a focus on control and increased confidence by failure to refute (e.g., see Variables and Validity (Section 3.9)). Consistent with constructivism, the research associated with this thesis was performed in its human context and in approaching foundational exploratory questions and classifying

sources of waste, it employs ethnography and emphasizes validation of interpretations (e.g., see Validation and Ethnography (Section 3.7)). This thesis also examines and provides support intentionally designed to affect the meaning and standardization of language in the S and P teams. Finally, in line with critical theory, this research is directly motivated to help meet real needs of a group that it engaged directly. It did in fact help that group, and demonstrated the use of novel advisor agent capabilities as a means to address restrictive systems of thought related to the perceived need to change the highly controlled tools, essential underlying work, or specific people involved with issue tracking.

Fundamentally, in this context, the truth exposed by this thesis is acknowledged to be incomplete and approximate but sufficient and valuable in that it worked to improve the real and specific problem at hand as both measured and acknowledged by those involved. It is grounded as a local theory in its human context, but relatively controlled and intended, in part, to highlight a potential role for novel advisor agent capabilities in helping to free people from a challenging situation.

3.10.2 Research Approach

The empirical research in this thesis can be characterized as mixed methods research. It uses a sequential exploratory strategy to first develop theories about sources of waste postulated in P_1 and then measure the effect of support intended to improve them consistent with P_2. After the exploratory phase, it also uses a concurrent triangulation strategy to provide corroborating evidence for the causal positive effect of the targeted training and agent capabilities on the problem (with issue rejection) at hand.

In the exploratory phase of this research, consistent with constructivism, a less central set of local theories on the sources of waste were encouraged to emerge from ethnography and classification of rejections. This classification, with rejections as the unit of analysis, was performed first in the *before* period as exploratory case studies under the study proposition that a large proportion of rejections could be attributed to a small number of different types of non-content sources of waste. Based on the theory in P_2, a similar approach was used in a limited confirmatory case study in the *after* period with the study proposition that the small number of different types of non-content sources of waste identified in the *before* period were still present but associated

with a smaller proportion (of a smaller amount) of rejections while content associated rejections filled a relatively larger proportion of rejection.

Survey research is not a major component of this work. A limited application of it was mentioned, as relates to questions asked after the training provided at the time of PnitAgent announcement, in PnitAgent Announcement, PNIT Tiger Team Findings, and Related Training (Section 4.5)). Responses were only received from about half of those that attended the training and it is not apparent whether it would generalize or even be meaningful to generalize to larger populations of the P and S teams or more broadly.

More centrally, in the design, application, and evaluation of targeted support, this thesis exposes verifiable hypotheses (see Detailed Research Questions (Section 3.8)) that are tested as part of a quasi-experiment. This research is also authentic in that it is directly motivated by, and achieves, not only statistical significance, but acknowledgement by many involved of the role that support played in practical improvement that benefited them. This work can also be considered action research in that it attempts to solve a real world problem with rejection on the P and S teams while simultaneously studying it. There are several problem owners in this case including the researcher, SQA, management, and those performing the issue tracking activities and part of the reason for not selecting control participants is the desire to not withhold a potentially beneficial tool from potential users.

At the same time it examines and employs controls to help move from correlation to causality and it exposes a case in which the targeted, advisor agent support is featured as a novel, successful, perhaps even liberating, approach.

Chapter 4

Sources of Waste and Means to Address Them

This chapter presents the results of formative rejection classification analysis and the identification of sources of waste, the types of capabilities introduced to mitigate them, a mapping between sources of waste and the means employed to address them, and finally a brief description of how this was shared with the S and P teams.

4.1 Identifying Sources of Waste

The formative classification of rejections was completed following the process described in Rejection Classification (Subsection 3.6.1) to provide an empirical and quantifiable basis for the sources of waste in this domain. Some excerpts of examples of such rejections with their classification can be seen in the table Example Rejection Classifications (Table 4.3). The overall results of this classification are summarized in the tables SQA Reject Classes (Table 4.1) and Peer/Self Reject Classes (Table 4.2) with the separation between the two as described in sections Software Quality Assurance Rejection Analysis (Subsection 3.6.3) and Peer/Self Rejection Analysis (Subsection 3.6.4).

The sample was taken during March 2010, resulting in 30 SQA rejections and 90 other (peer/self) rejections, all of which were classified for a total of 120 rejection classifications. There were 12 different classifications arrived at for the reasons of the SQA rejections and 17 for the reasons of the peer/self rejections, with similarities between

many of them (and about 6 pairs of classifications being nearly identical between the two SQA and peer/self groups of rejections).

After examining these classifications and their similarities, broader themes were identified. As an example of this abstraction, a depiction of the Versions (Subsection 4.1.2) theme can be seen in the figure Conceptual process of organizing data into themes or sources of waste March 2010. (Figure 4.1). The relative size in terms of number of associated rejections is depicted in the figure Pie chart accounting for major sources of waste March 2010 (Figure 4.2). The subsections under this section describe in turn each of these themes, or major sources of waste: Content and Change (Subsection 4.1.1), Versions (Subsection 4.1.2), Resolutions (Subsection 4.1.3), Related Records (Subsection 4.1.4), Reviewers (Subsection 4.1.5), and Miscellaneous (Subsection 4.1.6).

Table 4.1: Classification of Rejections by Software Quality Assurance for P and S teams sampled March 2010

Reject Type	**Count**	**Occurrence Rate**
Description implies something not addressed	4	13%
Version information incomplete	2	7%
Resolution incomplete	1	3%
Resolution vague/missing details	6	20%
Incorrect version #	5	17%
ReqPro tab usage	1	3%
Wrong closure reviewer	2	7%
Related records usage	2	7%
Process not understood	1	3%
Inconsistent filename	3	10%
Missing reviewers	2	7%
Missing form content	1	3%
Total	**30**	**100%**

Table 4.2: Classification of Self or Peer Rejections for P and S teams sampled March 2010

Reject Type	**Count**	**Occurrence Rate**
Documents affected != Version Implemented	1	1%
Change during Review - definition	1	1%
Change during Review - workload management	3	3%
Description not addressed	1	1%
Content Reject	14	16%
Inconsistency related record	1	1%
Inconsistency - report references wrong PNIT	1	1%
Inconsistency resolution	4	4%
Inconsistency version	10	11%
Incorrect closure reviewer	1	1%
Resolution vague/missing details	12	13%
Process not understood	4	4%
Missing related records	9	10%
Removed extra reviewer	1	1%
Missing reviewers	7	8%
Tool issue	2	2%
Version missing or incorrect format	16	18%
Total	**90**	**100%**

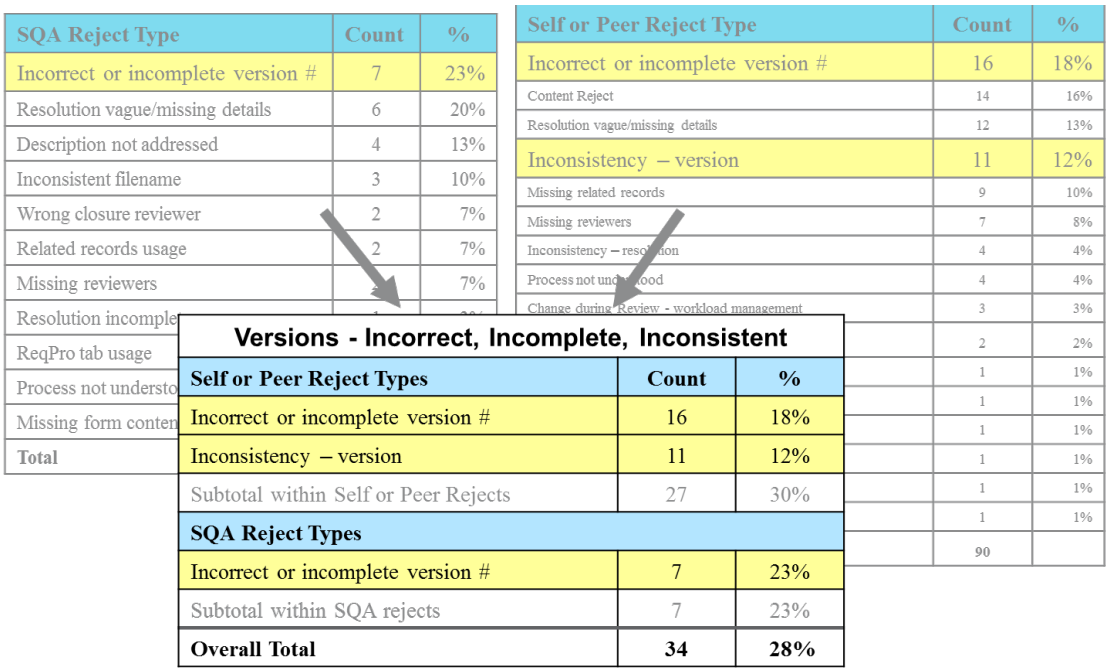

Figure 4.1: Conceptual process of organizing data into themes or sources of waste March 2010.

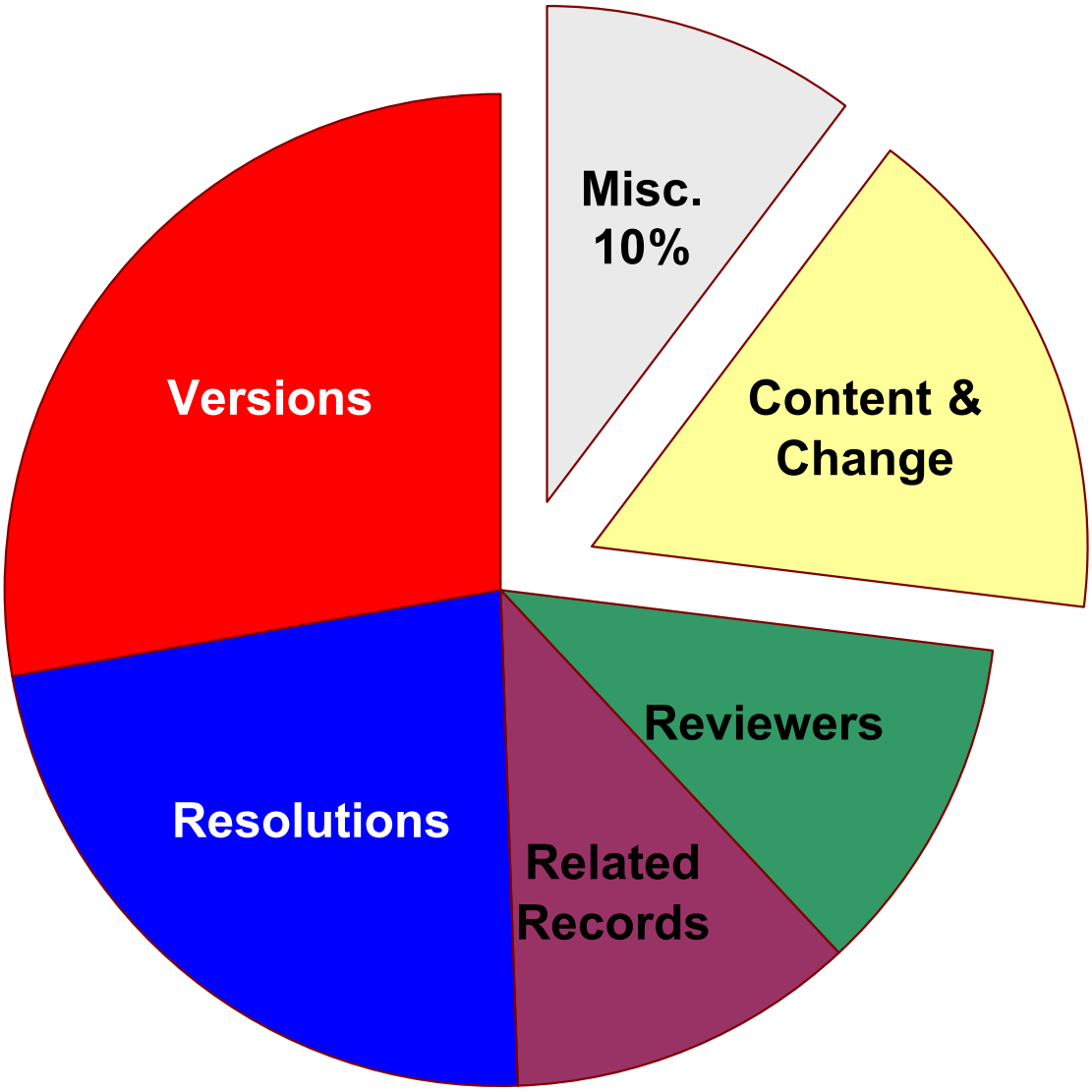

Figure 4.2: Pie chart accounting for major sources of waste
March 2010

Table 4.3: Example Classifications Software Quality Assurance Rejections for S team sampled March 2010

Issue	Discipline	SQA Rejection Note	Cause Class
...7540	Software	Please indicate in the Resolution Description whether or not a design document update was considered and updated.	Description implies something not addressed
...2393	Human Factors	Please list ...4506 on the Related Records tab.	Missing related records
...2584	Systems	Please clarify the location of the Sys. Spec by adding ...and file path.	Missing Req Info
...3562	Software	UIDD Component ....doc ...\3 is listed in the Resolution tab, however, it is not displayed in the AccuRev tab. Please clarify what PNIT it was checked in under.	Missing details - resolution description
...3686	Software	Resolution Tab, Version Implemented In field states ".../119". Review Report Form, section 11.3.5 Change Listing Rev Number states "...\120" Please indicate the correct version ...	inconsistency - version - form & Pnit
...3849	Human Factors	Action Item ...3855 only captures decision and work to be acted on. The Resolution and files updated should be captured in this PNIT.	Process not understood - SDT/Action-Item
...3904	Software	Resolution Tab: ...5661 is not covered by this PNIT. Please clarify or remove the reference.	Incorrect Related Records
...4072	Human Factors	...refers to ...UIDD.doc. It should reference ...UIDD_Main.doc instead.	inconsistency - file name
...4903	Human Factors	Please add a development reviewer and vt test reviewer that are working on ...	missing reviewers

4.1.1 Content and Change

More than 75% of rejections relate to *the story* rather than defects in the product under development. Flaws in the product under development are often found in other activities (development, code/design meetings, or test development), but are certainly within the scope of issue reviews. Product flaws were commonly referred to as "content issues" within the PNIT Tiger Team. Content issues that are caused not by implementation or documentation mistakes, but that were correct at one point but became concerns due to changes related to project redirection were referred to as "change issues." Change issues manifested as rejections when the changes in direction underlying them came about after a related issue was sent for review, but before it was closed. Such changes came, at least in some cases, from the addition or removal of features to/from the project scope and changes in response to new formative input gathered by human factors studies (the team S , for example, benefited from more than half a dozen formative user interface studies).

While change and content issues reflect waste, because they would ideally be caught before issue review, they are not a primary focus of this thesis. This is in part due to the fact that they are generally not as closely related to the issue tracking work itself and have benefited from significant attention in previous work. There are, however, some interesting exceptions to this. Certain types of content rejections could benefit from or are already closely coupled to aspects of the issue tracking work itself. For example, it was noticed that in some cases, even after issues were closed, references to them or the work they were supposed to complete could still be found in comments made within the product code and document artifacts. Unlike many other potential flaws in the product, such references are uniquely suited to be checked immediately before and during issue review.

4.1.2 Versions

Based on the rejections classified, the most significant source of waste is improper documentation of version information. This accounts for aproximately 23% of SQA and 28% of overall rejections during the formative sample period.

This version information is an essential part of *the story* told in tracking issues in this

domain. Problems with version information leave open questions that can make it harder to reproduce, understand, explain, review, and maintain the evolution of the artifacts and story. For example, under-specifying version information can lead to wasted time, degraded review, and even mistakes. A missing filename, path or snapshot prefix could lead to review of the wrong items which at best leads to some early confusion, rejection and waste, and at worst leads to bad assumptions that are the start of a series of failures that allow other problems to slip though until very late detection in an audit or even a bug in the field.

Version information is commonly found in several forms and places. The Version Found In, Version Resolved In, Documents Affected, Resolution, and the text of other fields within the issues themselves often contain version information. Additionally, artifacts (especially code, documentation of design and document reviews) that are associated with issues often include version information. Such information is commonly encoded as a unique snapshot/label/baseline name from the version control system that is used for referencing a specific set of files and their versions. For example, AccuRev is a version control system used by the S and P teams and each time source code is promoted for integration, a build server automatically detects the change, attempts a build and, if that build is successful, creates a new unique snapshot that can be used to reproduce that build. Additionally, specific items are commonly referenced by a unique identifier (e.g., filename including path, requirement tag, or document number) and an item-specific version number. For example, the S and P teams use a document control system referred to as "MRCS" with documents that go into their design history files.

In practical, larger scale application, with a number of opportunities, people involved and a variety of locations for, and forms of, encoding, it is not surprising that some version information might be entered in a way that is inconsistent, incomplete, or otherwise incorrect. For example an issue ...3832 was rejected because the Version Found In field merely stated "1.4.21.4" when it should have specified a full snapshot title "XXXXXXXXX_1.4.21.4"[1] . In this case, the Version Found In had been filled in by the issue submitter, was not changed (or even viewed) by the resolver who was also familiar with the problem in question, and lead to some confusion for the review (who eventually figured it out) and would have been even more problematic for an auditor or

[1] Here XXXXXXXXX is the snapshot prefix specific to the P team's version control stream.

released systems engineer reviewing the issue months or years later. In another example, issue ...3686 (summarized in Table 4.3) the Version Implemented In of the issue referred to a different version number than was listed in an associated review form. It was understood from discussion that developers sometimes struggled with the process of determining the appropriate, not-yet-created snapshot that would be created by the build server in the event that the build including their promote was successful[2] . During discussions with them, resolvers have claimed that sometimes they are so enthusiastic about moving forward with an issue that has been waiting on some information, that they leave a "TBD" type comment or nothing at all where version information should be provided. Transpositions and typographical errors are also particularly problematic in references to version information as a single mistyped number can lead to the entirely wrong version being examined.

4.1.3 Resolutions

Problems with the description of the resolution of an issue comprised roughly 23% of the overall rejections and more than a third of the SQA rejections sampled. They were a very significant source of waste. These primarily related to unclear, incomplete and inconsistent content resolution descriptions within the resolution description or summary resolution classification fields of the PNIT itself.

The resolution description is an essential part of *the story*. Problems with it leave open questions that can make it harder to understand, explain and review what changes were made which is essential for successful evolution of the artifacts. For example, underspecifying resolution information can cause rejection. A missing piece of resolution information could be flagged by an auditor, or more immediately, could cause confusion that hinders review and allows a defect to escape.

For example in issues like ...7540 (as summarized in Table 4.3) the story is incomplete. In this situation (and many similar ones), it is important for related considerations and "ripple effects" of the change to be considered, documented and reviewed. In other cases, the summary resolution classification is incorrect for the situation at hand.

[2] to be successful, the build server not only builds the software, but attempts a number of unit, integration, static analysis, coverage and other tests only after which does it create a snapshot trigger other dependent builds and so on

For example indicating that the issue is "Fixed" or "Non-Issue" when in fact it is a "Duplicate" of another issue where the resolver hopes it will be fixed.

In still other situations including ...3562 (as summarized in Table 4.3) the textual resolution description or resolution classification can be, or appear to be, inconsistent with other related information such as version control information (as is the case in ...3562), description of the problem, other issues, and so on. Even in situations where the user did their work in a way that is normally appropriate (such as association of version controlled changes with an issue) some inconsistencies can present themselves due to tool issues, changes in document strategy and so on and these are expected to be recognized and explained in the resolution description.

Additionally, issue documentation is expected to be sufficiently clear to be accessible to an external reviewer some years in the future. This clarity is also crucial during the project, as there is evidence the reviewers (particularly SQA and closure reviewers) often review in issues months or weeks after they were marked as ready for review. Years later, or even decades later, these may be relevant to others such as released systems engineers, engineers working on derivative products, external auditors, and so on.

4.1.4 Related Records

Problems with the citations (and lack there of) from an issue to related issues comprise about 11% of the overall and 7% of the SQA rejections. These primarily related to improper or inconsistent citation of related issues from the textual and relationship selection fields of the PNIT itself of in related documentation artifacts.

While resolving known issues with related records can be relatively painless, proper referencing of related issues is an essential part of *the story* of issues and their context. Problems with citations can lead to questions that can make it harder to understand, explain and review issues which is essential for successful evolution of the artifacts. For example, not citing or incorrectly referencing other issues can lead to wasted time, degraded review, and even mistakes. A missing issue reference could lead to assumptions about where and in what context work was completed. This can lead to early confusion, questionable documentation, rejection and waste. It could even lead to assumptions about that could be the start of a series of failures that allow other problems to slip though until very late detection in an audit, a bug in the field, or an even more subtle

effect from incorrect action taken on a related issue or activity.

4.1.5 Reviewers

Problems with missing, extra, or inappropriate reviewers of issues comprise 11% of the overall and 13% of the SQA rejections. These primarily related to improper fit between reviewers identified to review an issue and the review roles expected to be filled for that issue.

Resolving issues with reviewers is generally straight forward, often requiring the addition, removal, change or clarification of one or more reviewers and their roles. However, it can be a serious problem if reviewers are not correct as appropriate review is essential to ensuring compliance, quality, communication, completeness, objectivity, cross-pollination, and so on. For example, depending on the nature of an issue, it can be important to ensure any changes to product have been independently reviewed or that the problem described by the submitter is the same as that addressed by the resolver. As another example, it can be important to ensure that changes or new additions are visible to those in other roles that may have to create tests for them, consider their broader system impacts, validate their use, and so on. Review itself might be considered a form of waste if consistency, quality, and common understanding were achieved by other means. Currently, this is not the case and so appropriate review remains an integral part of the issue tracking process.

4.1.6 Miscellaneous

Finally, those sources of rejection that do not relate to Content and Change (Subsection 4.1.1), Versions (Subsection 4.1.2), Resolutions (Subsection 4.1.3), Related Records (Subsection 4.1.4), or Reviewers (Subsection 4.1.5) were grouped together under the rubric of Miscellaneous by default. This group compromised about 10% of the overall rejections sampled in March 2010. Though this group did contain some interesting rejections, these were considered rare and different enough to not qualify initially as a major source of waste, and, thus, are not a focus of this thesis.

4.2 Example Agent Use

This section presents an example use case to introduce the domain and agent. In this hypothetical scenario, a developer was assigned an issue; has made changes to product software and documentation; and has described her initial work in the issue.

Intending to check over the issue, the user opens it in the issue tracking system. PnitAgent has been running in the background, recognizes that an issue has been brought into focus, and identifies the issue. It gathers information from the issue tracking system and presents a small bubble window in the lower right hand corner of the user's screen. The bubble window has links that allow the user to check the issue for problems or take other actions. The window would fade away over a few seconds, but first the user clicks on a link in it to request *Automated Issue Analysis*.

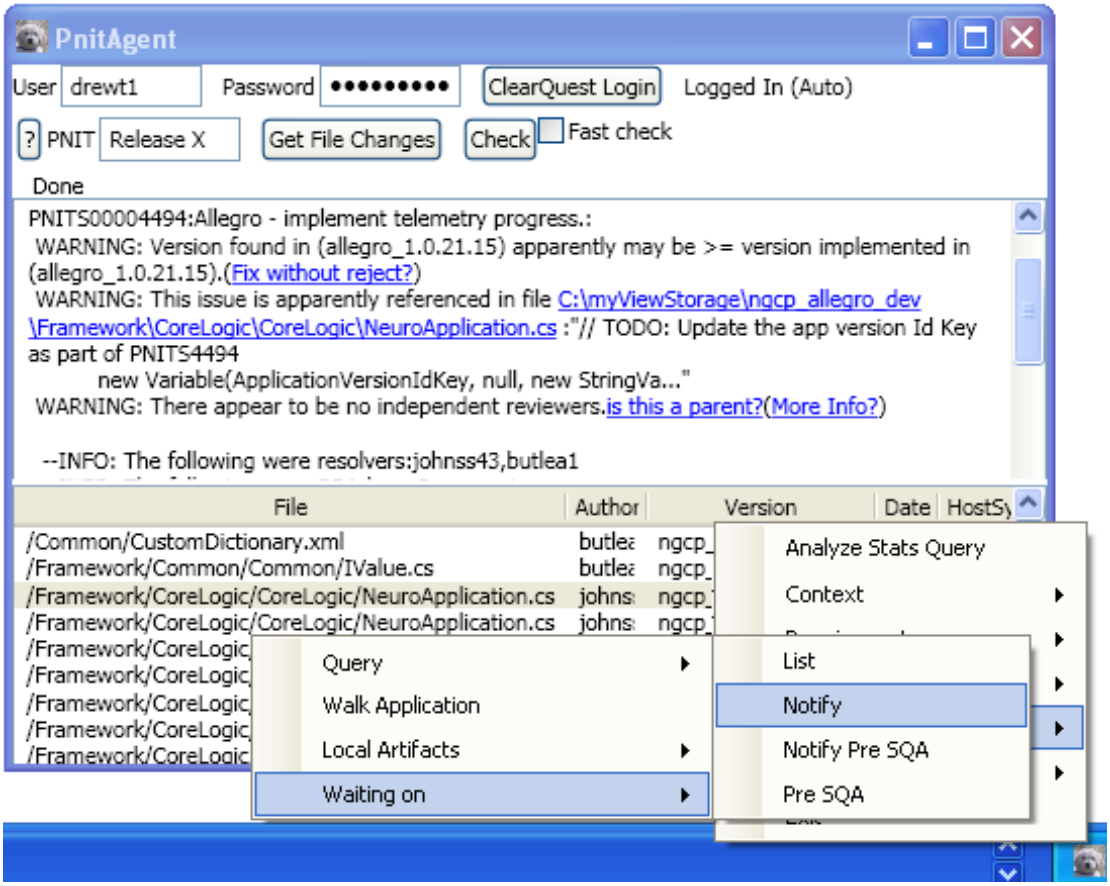

Figure 4.3: Example Agent Window and Tray Menu

The agent collects data about the issue from the issue tracking system, the version control system, its local memory files, and product artifacts. It uses text processing and information retrieval techniques to analyze the data (as described later in *Agent Design* and in [65]). When it identifies some concerns, it presents them to the user in the form of warnings and related information. In this case, the agent has determined from the free-form English text entered in the issue description and resolution description that the user has made changes to fix a problem in the product software and that there is no indication that review of these changes should be deferred to another issue.

The agent analyzes the assigned reviewers and who has contributed to the resolution of the issue and its associated artifact changes. Based on this, it produces a warning that there does not appear to be an independent reviewer even though one is expected

in this situation. It explains which reviewers appear to have been contributors based on issue tracking and version control data, and which users do not seem suited to be independent reviewers of this issue because they are verification or SQA engineers.

The agent also warns of areas of vagueness in the resolution description, contradictions between a version mentioned in the resolution description and other fields in the issue, and that the issue has been referenced from comments about work that is not yet complete in the code and design documents. In addition to textual explanation, the agent provides a link to a relevant website that documents the review roles expected and required in common scenarios.

After resolving or deciding to ignore the warnings, the user asks the agent to get all the artifact changes associated with the issue by clicking the "Get File Changes" button. This triggers a related desire/event for the requested goal to be raised in the agent's primary control thread. The agent starts a short term background thread to act on the goal. The background processing is tied to the "Get File Changes" button, which the user can use to stop it prematurely, and is *safe* in the sense that abort or failure of, or in, the thread will not be treated as an error. The background thread checks that the issue context is valid and gathers information about it and the files referenced from the issue tracking system, dispatching status updates back to the primary control thread that presents them on the main UI window if it is open.

The agent extracts information on change sets from the issue tracking and version control systems before examining the text of the issue to identify any changes made to artifacts not kept in the version control system. This involves retrieving free form text fields within the issue, replacing noise characters, removing noise words, using a regular expression to match references to documents in the document control system, then doing some additional information retrieval to extract version information about any document references that were found. The agent eventually integrates, organizes and dispatches a list of all the associated file changes to the primary thread.

Finally the agent presents a list of changes to the user, who re-sorts them by clicking on the column headers. The user then double clicks on the change-items and the agent takes its default support action for each. This generally takes the form of retrieving the changed and previous versions (e.g., from the version control system or the Medtronic Revision Control System) and displaying them in a graphical differencing tool to the

user, converting them if needed.

Occasionally, the user uses a context menu to do other types of review activities, such as start static analysis, view associated commit comments, or review approval status. Finally satisfied, the user marks the issue as ready for review.

With her last issue resolved, the user wants to urge her colleagues to complete any remaining work for the upcoming release. Having already created a query named "Release X" in the issue tracking system to return the relevant issues, she types "Release X" into the agent's entry box. Then she uses a menu item to notify the outstanding users (see Figure 5.5). The agent sends an email to each relevant user letting them know that issues are waiting on them, for what, and who asked that they be notified.

4.3 Novel Capabilities in PnitAgent

This thesis involved the development and incorporation of novel software capabilities into the PnitAgent software to support issue tracking activities as further described in Prototype Meta-Software Agent for Issue Tracking Help (Subsection 3.4.1), [65] and [66]. These capabilities are briefly summarized in the following sections.

4.3.1 Annotation and Sequential Support

One form of support provided was to allow for ubiquitous in-context annotation and sequence recommendation as described in Prototype Meta-Software Agent for Issue Tracking Help (Subsection 3.4.1). This capability requires the agent to monitor the focus of the user within the graphical user interface presented by the operating system and the applications running on top of it. In consideration of the potential performance and privacy aspects of monitoring, this capability required user initiation (although alternate, always on support was developed, it was not deployed). User initiation allows for generic monitoring support and subsequent selection of dependent support capabilities, such as annotation support or sequence support, but also allows for user to select "Usage Help" which would automatically enable annotation display and load a set of basic annotations relevant to how the S and P team use the issue tracking system. This can be seen in Figure 4.4.

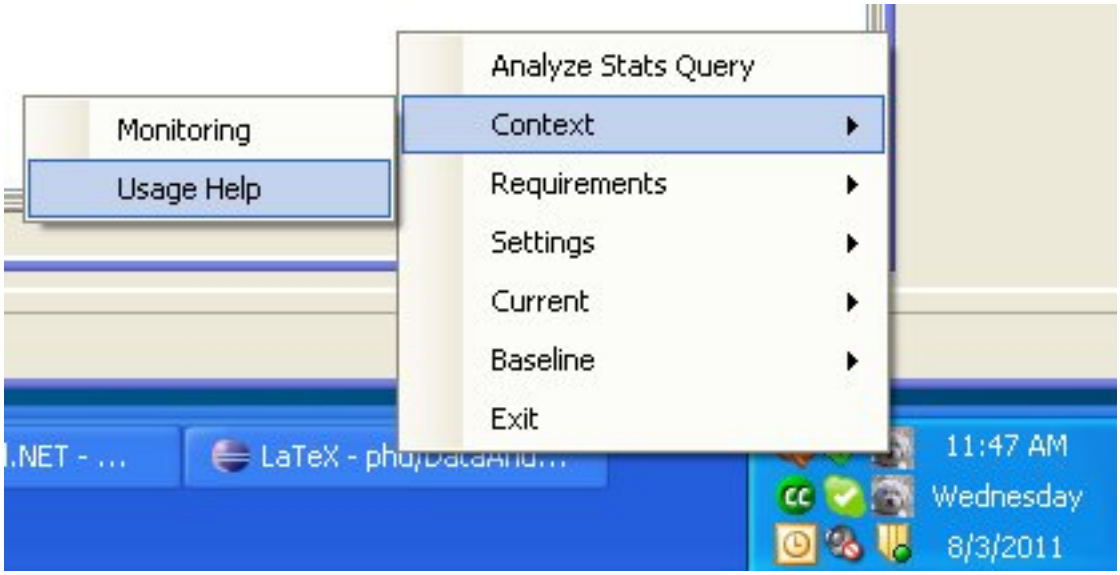

Figure 4.4: Screenshot Example of Menu To Initiate Monitoring

4.3.2 Automated Issue Analysis

Capability was implemented to provide automated analysis of issues and their associated artifacts with the aim of providing warnings to help users identify common problems. In addition to the description provided in this section, some additional details can be found in [66, 65]. In relation to Annotation and Sequential Support (Subsection 4.3.1), this is easily initiated but is not as easily customized and requires significantly more conscious effort from the user to manage, interpret and so on.

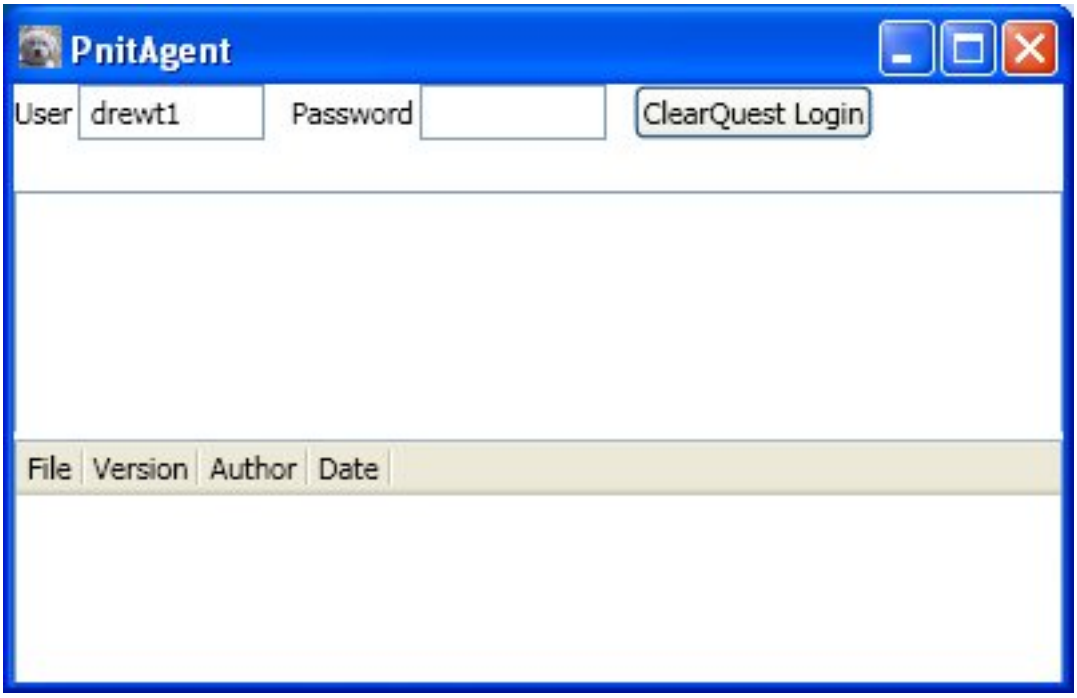

Figure 4.5: Screenshot Example of Initial PnitAgent Window

Automated issue analysis support requires access to issue tracking (and possibly other) data that is password protected. When the agent is first started it checks for the currently logged in user's username. It then looks for a local, encrypted file based on that username. If it does not find this file, its UI will show as seen in Screenshot Example of Initial PnitAgent Window (Figure 4.5). The user must then provide their password and press the ClearQuest Login button that will cause the agent to attempt to login to ClearQuest and if successful to save the password in a local, encrypted file (or indicate error information and prompt the user to try again). If the user has previously logged in and the loaded password information is still valid, the agent logs in automatically. If the agent has not been closed, it will continue to use the login information with the issue tracking, version control and other systems as needed. Once logged in, the agent will indicate that it has logged in and make additional functionality available as seen in Figure 4.6. Similarly, a number of other detailed activities are performed when starting up, including checking for support for its preferred method of accessing version control, and textual output and UI adjustments are made as appropriate.

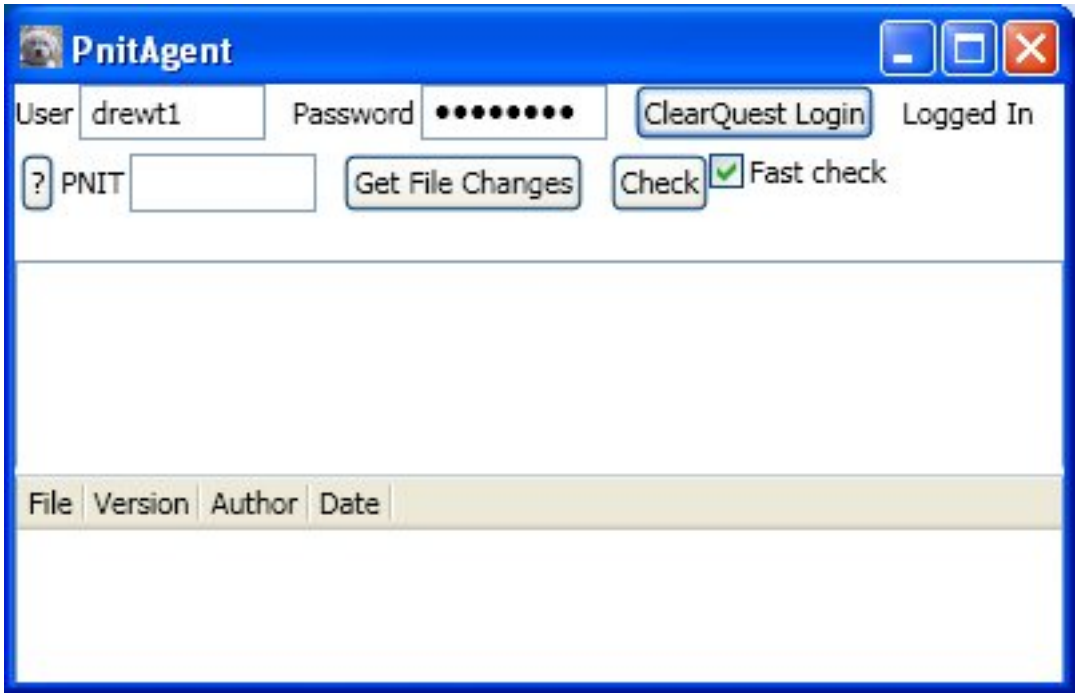

Figure 4.6: Screenshot Example of PnitAgent Window After Login

Once the agent has started up, the user may identify the issue of interest and ask the agent to analyze that issue. To accomplish this, users type a PNIT id or the name of a query stored in the issue tracking system that is somewhat flexible in format (e.g., "4494," "PNITS00004494," "Issues Waiting on Me To Review" or so on), but requires the user to know and correctly enter this information. After that the user activates the "Check" button to begin analysis (which can be aborted by pressing the same button again). The user may first want to change the state of the "Fast Check" checkbox to configure whether slower but more thorough analysis or faster less thorough checking is performed (taking only a few seconds per issue or taking up to a minute or more depending on caching, number of associated files and so on). While checking, the agent describes its status and what it is doing in a one line, transient status text block and prints out the results of its analysis and associated explanations, links, and etcetera into a larger rich text area below that (as shown in Figure 5.5). Many of these checks require information that is retrieved automatically from multiple sources including settings files,

the issue tracking system, the version control system, build server websites, and source code and other artifacts on the local hard drive.

4.3.3 Hybrid Interactive Support

Support was also provided in the form of a hybrid between more traditionally interactive capabilities such as Automated Issue Analysis (Subsection 4.3.2) and more loosely coupled monitoring capabilities including Annotation and Sequential Support (Subsection 4.3.1). This involved monitoring the user's focus and when they enter certain recognized contexts, providing context appropriate dynamic content or capabilities. For example, the agent may recognize the Version Implemented In field of an issue that is being modified and allow the user to request that it help identify, and copy to the paste buffer, information about the appropriate build by selecting a link presented in a transient window as seen in Figure 4.7. In another example, the agent may recognize when the user selects a new issue in the issue tracking UI, perform Automated Issue Analysis (Subsection 4.3.2) in the background, and alert the user if any significant warnings are found (as described in [66]).

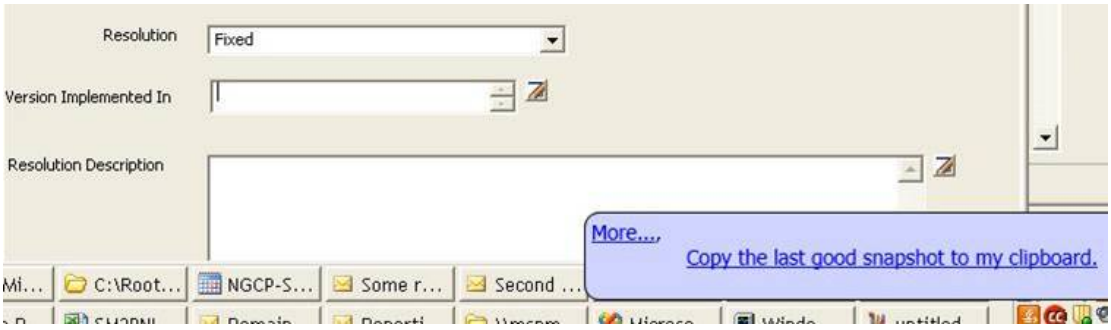

Figure 4.7: Example of Hybrid Support for Version Recommendation

4.3.4 Miscellaneous

In addition to the relatively novel forms of support described above, some additional, miscellaneous support was implemented to help address the major sources of waste identified. This included minor changes to practice related to use of review report

forms, detailed documentation and discussion of the findings of the PnitTiger team, and agent support for rapidly retrieving and reviewing a list of artifact changes/versions associated with an issue. The agent was also extended to support load balancing or issue ownership for resolution and review, monitoring and reporting support including metrics that have been adopted by several other teams, and other less relevant capabilities such as automated fixing of copyright information that was ultimately not applied to the artifacts of the S and P teams due to the burden of validating this capability or reviewing the changes made. Some additional details can be found in [66, 65].

4.4 Mapping From Major Sources Of Waste To Support

This section describes the mapping from the sources of waste identified to the efforts aimed at addressing them. This is summarized and indexed by Sources of Waste and Means to Address (Table 4.4) and detailed in the subsequent subsections for each source of waste.

Table 4.4: A summary of the major sources of waste identified and the capabilities and steps taken to address each.

Source of Waste	**Support Capability Created to Address . . .**
Versions	Warnings (missing found or implemented in, malformed, . . .)
(Subsection 4.1.2)	Usage Help
	Context Appropriate Recommendation
	see Versions Support (Subsection 4.4.1)
Resolutions	Warnings (inconsistent classification, TBD in resolution, . . .)
(Subsection 4.1.3)	Documentation of meaning of resolution classifications
	Usage Help
	see Resolutions Support (Subsection 4.4.2)
Related Records	Warnings (text vs related records, duplicate missing related, . . .)
(Subsection 4.1.4)	see Related Records Support (Subsection 4.4.3)
Reviewers	Warnings (incorrect closure; missing independent, test, sqa, . . .)
(Subsection 4.1.5)	Documentation of expected review roles in common scenarios
	see Reviewers Support (Subsection 4.4.4)
Content	Warnings (TODOs in artifact, missing security attribute, . . .)
and Change	'Waiting On' support
(Subsection 4.1.3)	see Content and Change Support (Subsection 4.4.5)
Miscellaneous	Usage Help
(Subsection 4.1.6)	Similar Issue Identification
	Warnings (undesirable language, missing requirement, . . .)
	see Miscellaneous Support (Subsection 4.4.6)

4.4.1 Versions Support

To address the Versions (Subsection 4.1.2) source of waste, several approaches were employed. These centered around defining and providing training about appropriate use of version information in issues, providing automated issue analysis to warn users about possible mistakes, and providing simplified access to recommended version information.

At a high level, what is appropriate for including version information in issues was

not controversial. However, in more detailed application, it was apparent from rejections and early Validation and Ethnography (Section 3.7) work that best practices were not always apparent or followed. Description and examples of appropriate use of all the version related fields in some common scenarios were proposed and reviewed by the PNIT Tiger Team. These were eventually encoded in annotations and more detailed text linked by them that could be shown (by the agent) in the context of the relevant fields of an issue as a user is editing it. Document focused help, for example, would be presented by the agent in a transient message as the user gave focus to the Documents Affected field.

One challenge with this "context sensitive" usage help for version information was that, while it was related closely to the field in question, the content was sometimes complex and would describe conditional concerns that depend on the scenario of the particular issue. For example, appropriate use of the Version Found In field varied with the functional classification of the issue's subject matter (e.g., it could be left blank in some cases for "Human Factors" concerns, but must be filled for software bugs). Similarly, if the Version Found In was provided and the Version Implemented In or Documents Affected referenced the same or a lower number version (of the same or apparently the same artifact), this was a problem that should be fixed or explained.

To help users address the complexity of applying the various rules to the issue at hand, support for Automated Issue Analysis (Subsection 4.3.2) was created to examine many of these rules in the context of the issue at hand and warn the user about any apparent problems. This includes accounting for the example concerns identified above as well as others in providing warnings, such as those when version information is apparently missing, incomplete, malformed, inconsistent with other version information or something else in the issue and so on.

While these kinds of warnings and their associated information can be helpful, in identifying problems and suggesting how to fix them or avoid them in the future, they were not always helpful in catching or preventing errors during entry of version information (e.g., digit transposition or other typographical errors). To help address this, warnings and training reaffirm a simple "best practice" encouraging users to reference snapshots instead of multiple individual files so as to reduce the chance for errors. The use of snapshots, however, has also been identified as challenging and still prone to some

errors because members of the S and P teams would have to look it up from the version control system or the appropriate build server's web or tray interface after a build was complete. This information was not easily and rapidly accessible where it could be transcribed, let alone electronically copied from one place, and it was common for an off-by-one build or incorrectly prefixed snapshot to be recorded in an issue. To improve this, users could ask PnitAgent to recommend the Version Implemented In build and it would respond with the likely snapshot labels that the user could then copy and paste. Users could alternately enable monitoring and when they navigated to the appropriate field a link would be presented in a transient message and, if they clicked that link, it would put the recommended snapshot information into their clipboard so they could simply paste it into the currently focused field (e.g. Version Implemented In as shown in Example of Hybrid Support for Version Recommendation (Figure 4.7)).

4.4.2 Resolutions Support

Support for the Resolutions source of waste focused on defining, documenting, and providing help for the use of concrete resolution classifications; and providing automated issue analysis to warn users about possible mistakes.

Due to their discrete and consistent nature, the use of concrete resolution classifications was a primary area of focus in the effort to define and document detailed expectations. In addition to some basic rules (such as 'Avoid use of Other if a more specific Resolution [classification] applies.'), the meaning of each resolution classification option was described and some simple 'red flags' were described and made available through training and usage help annotations that would be shown when the Resolution drop down was in focus or (overriding the previous) one of its options were highlighted but not yet selected. For example, if the "Fixed" option were highlighted a transient annotation message would display summarizing its meaning (that the concern was fixed in *this* issue) and providing a link to more information from the Twiki website material (that had been validated by the PNIT Tiger Team).

Checking of the 'red flags' associated with the selected resolution classification was one of the automated issue analysis capabilities provided. For example, if "Fixed" were selected as the classification and the issue indicated its type had to do with "Code" but there were no changes from the version control system associated, then the agent would

warn the user.

In addition to the consistency related "red flags" type checking and warnings, there was also automated issue analysis support for other types of resolution related concerns. Checking for incompleteness, for example entailed using natural language processing techniques to look for identify possible indicators in resolution description such as the use of "TBD," "TODO," "???" and so on before extracting some of the surrounding text to present to the user so they could rapidly determine if it was a false positive without reading through the text in the relevant field. Similarly, capability to identify common types of issues, such as those associated with a design review, and detect whether an appropriate aspect of the resolution was present, such as a reference to a reasonable design review report document, was employed for some further resolution checking. Additionally, if the issue appeared to be highly complex, the agent would warn the user as this was suspected to be associated with a more difficult review and with higher rates of rejection due partially to inappropriately detailed or complex resolution description and inconsistency between the resolution and other aspects of the issue.

4.4.3 Related Records Support

Support for the Related Records (Subsection 4.1.4) source of waste focused primarily on providing automated issue analysis to warn users about possible mistakes. This capability was created in two basic forms:

1. natural language processing analysis of the issue at hand to identify textual references to other issues that were then considered against the list of related records to identify possible mismatch errors and

2. analysis of issue details to identify scenarios that require a cited or related records in cases where none have been.

In form 1, if a record is referenced in the text of an issue but not selected as a related record, the identifier(s) of the referenced record(s) is provided and the user is warned that they may be missing a related record or have an inconsistency with their related records. In form 2, a user might be warned that the issue in question has no related/referenced records, but has been marked as a duplicate issue (where the issue that it is a duplicate of must be identified/related). There are several other examples related to the

second case, such as no related records in the case of an issue that has been classified as out of scope where the issue or "action item"[3] describing the decision to change scope must be related.

4.4.4 Reviewers Support

To address the Reviewers source of waste, the capability to check for and warn about concerns relating to missing or incorrect reviewers has been introduced as discussed briefly in [65]. This includes improved roles support and a number of warnings related to:

1. **Independent Reviewers** - These warnings are raised when there is no independent reviewer listed in the outstanding or completed reviewers for an issue where one is expected.[4] The involvement of an independent reviewer can be required by higher level process and to ensure reasonable objectivity where needed.

2. **SQA Reviewers** - These warnings are raised when there is no software quality assurance engineer listed in the outstanding, completed or closure reviewers for an issue associated with an issue where one is expected.[5] It is important to ensure SQA participation in reviews where expected as part of ensuring high quality and consistency in issues as well as agreed upon coordination between quality assurance and development. Each user can configure what users are considered SQA or can accept the default configuration.

3. **Test Reviewers** - These warnings are raised when there is no functional verification test engineer listed in the outstanding, completed or closure reviewers for an issue associated with an issue where one is expected.[6] It is important to ensure

[3] Here the term "action item" refers to a special type of record that is kept in the issue tracking system.

[4] Here independence is measured by exclusion of reviewers who are testers, SQA, a resolver of the issue, or (if configured for more thorough "slow" checking) a contributor to the source repository versions associated with the issue. Expectations for inclusion of this reviewer depends primarily on the resolution type classification. For example, an independent reviewer is not required for a duplicate issue.

[5] Here expectations for SQA review depend on the associated project and issue type. For example the S team is not required to include an SQA reviewer on Enhancement or Defect issues sent to review as of 2010.

[6] Here expectations for inclusion of this reviewer type depend on the associated project and issue type. For example, some test issues, as well as design and code review issues, do not always require test

test participation in reviews where expected as part of ensuring proper communication, minimizing or at least accounting for test coverage, rework and scope change, and consistency in issues as part of agreed upon coordination between verification test and other groups. Each user can configure what users are considered testers or can accept the default configuration. This configuration and logic can be somewhat complex as resources may take on different roles on different teams/projects and at different times.

4. **Other** - These warnings are raised when a variety of other undesirable conditions are detected. They include for example, concerns related to incorrect closure reviewers, missing systems reviewers for requirements changes or risk discovery that may affect the broader system, and so on.

These warnings are important to ensuring process compliance, appropriate communication, and other important aspects of the issue tracking review process. The detection of undesirable conditions and presentation of warnings involves to some extent understanding the issue(s) in question, determining the roles and state of their reviews and reviewers, identifying if/how they may have been redirected[7], examining the users settings and issue related artifacts and version history, and ultimately performing context appropriate reasoning to determine if the expected roles have been satisfied appropriately. If they have not, the agent then responds by not only raising warnings, but explaining (e.g., through detailing the understood and expected roles of existing reviewers and other involved parties such as resolvers for the current issue/scenario), providing links to web-accessible information including the table excerpted in Figure 4.8 and a significant amount of additional content.

---

reviewers. On the other hand, localization related defects and enhancements generally require at least one and sometimes two different test reviewers to ensure appropriate functionality as well as appropriate context availability in non-English languages.

[7] For example, by parsing various free form fields, the agent software is able to identify a number of common types of review redirection such as when a resolver has indicated that review of this issue will be reviewed under another and can perform further checking in such cases to help ensure that review occurs with appropriate timing, related records and so on while relaxing expectations on the redirecting issue.

Situation	Reviewer Role	Field	Responsibility
Normal SW (image)	Self	None (informal)	Self Review (informal)
	Independent	Outstanding Reviewers	Independent Pnit Review
	Test	Outstanding Reviewers	Test Impact Pnit Review
	SQA	Outstanding Reviewers	SQA Pnit Review
	Closure	Closure Reviewer	Closure Review
Duplicate	Self	None (informal)	Self Review (informal)
	Submitter(1)	Outstanding Reviewers	This pnit was understood correctly and is clearly covered by the it is claimed to be a duplicate of. This pnit is not believed to be referenced anywhere in the curre example, there should not be "TODO: PNITXXX" in code or docu is this issue we're about to mark a duplicate.
	SQA*	Outstanding Reviewers	SQA Pnit Review
	Closure	Closure Reviewer	Closure Review
(SDD)Design or Code Review	Self	None (informal)	Self Review (informal)
	Independent	Outstanding Reviewers	Independent Pnit Review This pnit is not believed to be referenced from TODOs in the curr Includes the Review Form and its consistency with the pnit
	SQA	Outstanding Reviewers	SQA Pnit Review
	Closure	Closure Reviewer	Closure Review
Requirement Proposal See also Sm3ChangingRequirements	Self	None (informal)	Self Review (informal)
	Systems	Outstanding Reviewers	Systems review.
	HFE (if UI related)	Outstanding	HFE review.

Figure 4.8: Excerpt from a summary of expectations for issue reviewers

In addition to warning users about missing or incorrect reviewers, support is also provided by usage help and training that describes many of the most common scenarios and the associated expectations for reviewers. For example, in the case of an issue that has associated code changes, an independent reviewer is required (unless review for the changes has explicitly deferred to another issue). Similarly, for issues resolved as duplicates, not repeatable or non-issues, the submitter is required to ensure that the problem was understood correctly; for certain types of requirements change proposals a system engineering reviewer is required; and so on. Several of these scenarios were described in a large table on an internal website and other documentation created by

the PNIT Tiger Team (excerpted in Figure 4.8). Each row of this table describes the scenario, the recommended and required reviewers, rationale, and so on. Some of this was summarized in the usage help annotations themselves and some of it is linked to the relevant area of the Twiki website.

4.4.5 Content and Change Support

As mentioned previously, Content and Change Support was not the primary focus of this thesis and has already been the focus of significant related work. However, a few relatively isolated concerns were addressed in interesting ways due to the findings of the numerous careful reviews and discussions performed and the insights provided by looking at the potential for supporting them through our relatively unique issue focus and given the novel automated issue analysis capabilities created. These focus primarily on support for recognition of and warnings about incomplete work at the time of issue review. We found this to be a particularly interesting and useful vantage from which to identify problems that would otherwise be difficult to detect and resolve at an appropriate time. Consider for example, the use of "TODO" type code or document annotations that are an important practical part of tracking detailed work remaining within product software and documentation being developed. Although this example has been described previously in more detail in Code Annotation (Subsection 2.1.2), it is useful to restate here that checking for these at issue review is one of the novel capabilities implemented in the agent. This, along with checking for unjustified static analysis suppressions, missing security attributes and the like to provide both warnings and informational output, was intended to provide for improved awareness of the state of related artifacts and not only the issues referencing them, but the issues referenced by them.

In addition to this type of analysis and warning support, some interesting capabilities were developed to allow users to rapidly identify and work with what a particular issue was, or group of issues were, waiting on before they could be considered complete. For example, the user could request that an email be sent to those involved with a particular specification, notifying them what is waiting on them and who has requested that it be completed. These sorts of capabilities were aimed at helping coordinate, balance and organize work as well as identify bottlenecks and priorities.

4.4.6 Miscellaneous Support

Some extremely simple support was provided in addition to more sophisticated and major sources of waste focused capabilities to reduce problems with rejection. Perhaps the most significant aspect of this was a change to reduce a small amount of redundantly captured information on review forms that was proposed and put in place for the S team (it had already been in place for the P team) affecting a small fraction of highly complex issues involving certain types of formal review forms.

4.5 PnitAgent Announcement, PNIT Tiger Team Findings, and Related Training

On July 15, 2010 a 1.5 hour training was given by the PNIT Tiger Team described previously. This meeting was required for those involved with the S (n≈20 members) and P (n≈15) projects and open to other managers who heard about it by word of mouth to attend. Ultimately 31 people signed the attendance sheet and about 40 attended. During the meeting some aspects of PnitAgent were demonstrated in the context of the broader findings of the PNIT Tiger Team. Data collected by the agent that supported the topics presented was also summarized (some of this data is shown in SQA Reject Classes (Table 4.1) and Peer/Self Reject Classes (Table 4.2)). During the presentation, a 7 question "PNIT Training Summary" quiz that had been designed by a human factors member of the PNIT Tiger Team was distributed. Nineteen of these were returned completed. The final question on the quiz was "As a result of this training, I will now do the following when resolving PNITS:" Of the 19 responses to this question, 11 directly indicated they would use PnitAgent.

Chapter 5

Advisor Agent Software Implementation

This research involved the development of advisor agent software, called PnitAgent, to support issue tracking. This chapter summarizes the architecture of this agent and some key design challenges faced during its construction. These are part of the context of this thesis and may be relevant to members of the relevant communities interested in practical aspects of the software implementation.

Key design challenges in the design of PnitAgent included:

1. Monitoring the user activities in a way that is non-intrusive, has acceptable performance, is ubiquitous, and is easy for users to control.
2. Abstracting the complexity of user and environment specifics to ensure robust and appropriate operation with different user credentials, preferences, levels of experience, and access to tools, data and services.
3. Building a model of each user, with history and preferences, and creating appropriate analysis tools to enable the agent to support each user.
4. Presenting information, intentions, status, and suggested next actions in a way that is helpful and acceptable to users.

The agent's design and implementation started with research into highly abstract monitoring of the user's actions as the user shifts focus between windows and widgets within those windows [61]. This degree of abstraction was initially pursued to allow

for operation with the various related files, (web-, Eclipse-, and native-Windows-based) issue tracking clients, and related systems.

An early prototype of the agent demonstrated feasibility for cross-technology, probabilistic modelling of sequences of user focus. The agent observes the user and builds a Markov-chain model of the sequence of contexts it observes over a period of time. It then uses the model to recommend next user actions. The model leverages an efficient trie-based data structure and an algorithm for training and prediction that is similar to the Prediction by Partial Match algorithm proposed in [67] and further analyzed in [68]. In this way, the agent can guide users though complex and unfamiliar user interfaces.

This early prototype also demonstrated support for ubiquitous annotation by leveraging a simplified form of programming by demonstration where the agent observes the user as she demonstrates UI focus contexts and associates annotations with them. The agent later recognizes that the user enters one of these contexts and presents the associated annotation content. This capability was eventually simplified in PnitAgent to remove the need for a browser plugin, and was seeded with content from previous training to make it immediately useful without requiring users to train the agent. Advanced users may selectively train, save, combine, and load alternate models as may be appropriate for their team or other specific use. The initial capabilities prototyped, while useful for new users, training, and recall, were too restricted by their exclusive coupling to the sequence of the user's focus. This left the agent only able to react to, perceive, and help with content and problems that were already in the user's focus.

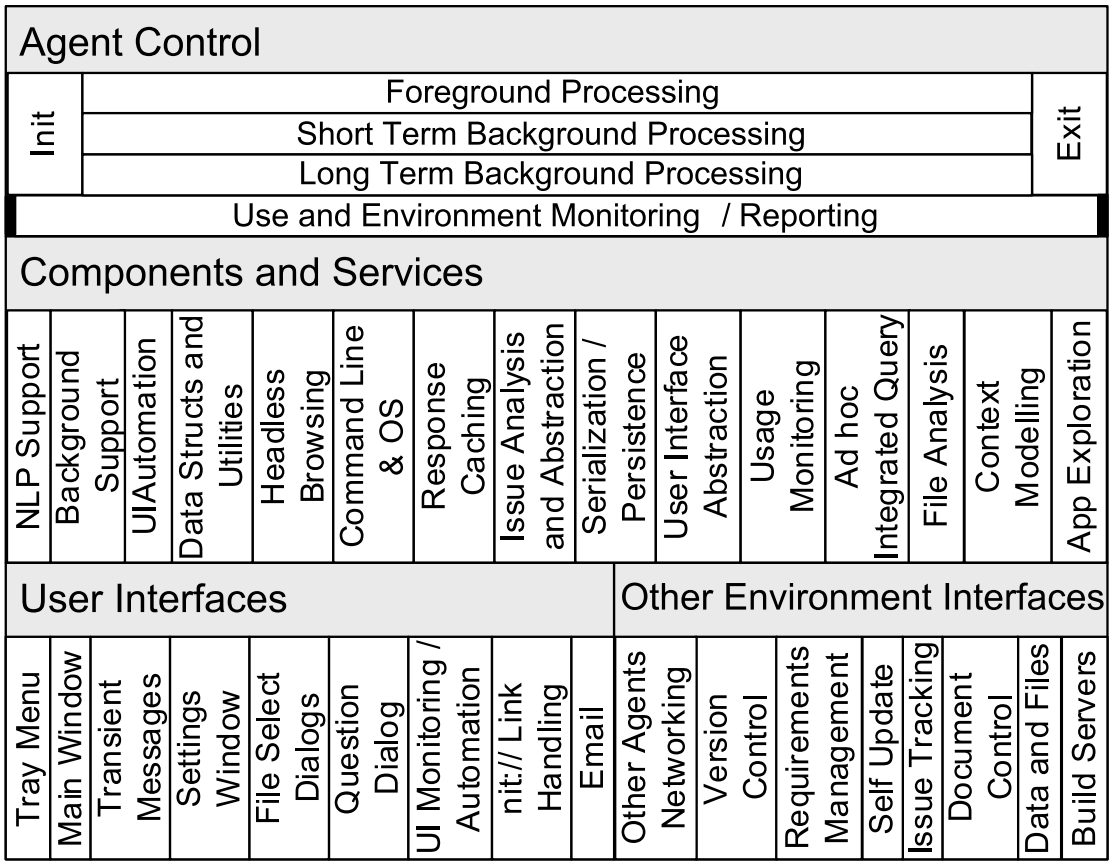

Figure 5.1: Logical architecture diagram.

To address these shortcomings, PnitAgent was re-architected from the early prototype to gather information directly from the various tools and systems available to it, in addition to working through the graphical user interface (see Figure 5.1). The fact that the agent gathers information directly from other systems creates a greater coupling to existing software. This made deployment and maintenance more difficult, but enabled additional capabilities. For instance, access to data, such as textual, unstructured, user-entered descriptions of concerns and resolutions requires the ability to effectively process the contents of the text to provide more proactive, detailed, and scenario-specific support by understanding something about the issue at hand. The agent does not do full grammatical analysis of the English free form text, but uses a series of application-specific pattern matching and basic language processing techniques (e.g., stemming,

proximal text extraction, and replacement of noise words and characters). An example of simple stem matching is given in Listing 5.1. To increase performance, the agent exploits information about the context to guide its process of elimination in extracting and classifying information.

Listing 5.1: Simplified stem match for target modifier

```
if(lowerQuery.Contains("clos"))
  modifiers(ModifierTargetType.ToClose);
```

The agent is built around dynamic dialog with its environment. The agent control layer of the logical architecture (see Figure 5.1) reflects the control and flow of its processing. The agent performs its primary processing on a main thread onto which some key event processing is dispatched and from which other background threads are started and managed. Ongoing monitoring and reporting provide support and input events even during initialization and shutdown. Much of the agent's architecture and processing is structured around interfaces with the user and the environment, that are supported and abstracted by services and components.

Each agent maintains information that is unique to it and its users, including observations, user responses, context associations, and use information. It acts differently based on that information and a number of environmental factors such as who the user is, what folder it was started from, who logged in with it last, what version its software is, what systems it can access and the like. The agent keeps track of its past and current users with separate information associated with each. Unless a user has been explicitly identified to the agent, it typically assumes its user is the one currently logged into its host operating system (OS).

Multiple agents can act on behalf of the same user. Different instances started from the same location share persisted information. If these are started by the same user (or as link handlers on their behalf), they will heavily share information, but may operate on different tasks and observe different user interfaces (e.g., if started on different machines from a common network folder). Multiple instances may be started simultaneously by one user on one or several machines for various reasons. For example, the OS starts new agent instances to handle links activated outside of an active agent and, depending on what is requested, a new agent instance may start, complete the task, and exit automatically.

A key focus of the design was enabling the agent to dynamically alter how it observes and interacts with the user and electronic resources around it. For example, when the agent experiences an unanticipated exception while trying to access the COM interface to the version control system, it keeps a note that it is unavailable and tries the command-line interface. More generally the agent identifies the availability of the interfaces and systems by attempting to find and test them, validating their responses and recognizing failures in this process and during use. If a system is completely unavailable, the agent overrides and hides access to any conflicting settings or actions and explains the situation both immediately and as needed later on (e.g., indicating that reviewer independence can only be partially checked without version control access).

The agent uses information from the OS to identify the local user and, based on that, searches for other information about them. For example, if a user asks the agent to notify others about what is waiting on them (through a link or as shown in Figure 5.5), the agent will not notify all users about everything waiting on them. Instead, it asks the user for context (or retrieves it from the PNIT field if it is not already in focus on the screen). Responses in forms including "4494," "Release X," "everything related to 4494," "waiting on my closure," and many others are understood by the agent. This is accomplished by processing summarized in the following algorithm (see Figure 5.2).

Figure 5.2: Algorithm to get issue context from response R

if R is an issue id **then** ... ▷ "4494", "PNIT 4494"
else if R is a publicly defined query **then** ...
else if R is a *currentUser* defined query **then** ... ▷ "Release X"
else if R is a list of issues ids **then** ... ▷ "1,2" "Pnit1 Pnit2"; see Listing 5.2
else ▷ R is a natural language query or invalid
    if R is a 'waiting' type query **then** ▷ "the stuff waiting on doej1"
        $type \Leftarrow waitingOn$
    else if R is a 'related' type query **then** ▷ "everything related to 4494"
        $type \Leftarrow relatedTo$
...
    end if
...
    if R has a specific issue target **then** ▷ "everything related to *4494*"
        $target \Leftarrow$ issue id from R
        $targetType \Leftarrow issue$
    else if R has a specific user target **then** ▷ "the stuff waiting on *doej1*"
        $target \Leftarrow$ user id from R
...
    end if
    if R has current user target **then** ▷ "... my ...", "... me ..."
        $target \Leftarrow currentUser$
    end if
    if $targetType = specificUser$ or $targetType = currentUser$ **then**
        $targetMods \Leftarrow$ modifiers from R ▷ "... my *closure*"; see Listing 5.1
        if count of $targetMods < 1$ **then**
            $targetMods \Leftarrow resolve + review + close$
        end if
    end if
...
end if

The algorithm uses application-specific techniques such as the regular expression shown in Listing 5.2 to match, classify, and extract information from the user's response.

Listing 5.2: Regular expression identifying lists of issue ids

```
new Regex("(,|;|([0-9]+\\s[0-9]+)|([0-9]+\\spnit))",
```

The use of domain-specific information retrieval techniques is possible because, despite the large variety of information the agent needs to find and process, the domain is

constrained and structured, primarily around the issue tracking interface/process and the issue context.

Even beyond the information retrieval techniques, the design of the agent was heavily focused on applying software engineering and other concepts to uniquely support and leverage the issue tracking structure and context. For example, as described in the following sections, this work re-imagines and applies static analysis (in a number of ways; Section 5.2), artifact annotation (Section 5.3), change sets (Section 5.5), and to some extent role-based reasoning (Section 5.4) in the context of issue tracking and issue review. It also provides a practical automation to better support data tracking, analysis, and presentation to allow for insights into evolving project status related to issue tracking (e.g., effort, rejection, complexity distribution) as well as other key areas including requirements, key documents and other deliverables. All of this is built on top of, and distinct from the commercially available feature sets of related tools and their basic customizations and integrations.

5.1 Existing Implementation Foundations

As an example of the sort of automation already in place before this thesis, it is useful to consider a basic flow of events just before and at the time of issue review. After a set of changes are completed by a developer on the S or P team, they are committed to a development stream in the version control system. Before the version control system will accept such a commit, association of the changeset with an appropriately configured issue from the issue tracking system is required. After commit, a build server detects the availability of new changes and automates build, static analysis, testing, coverage analysis, snapshot creation, metrics collection, triggering of related builds, status communication and other activities. The user then finishes the description of the snapshot and other details in the associated issue, ensures it is ready for review, and marks it as such. The issue tracking system then changes the issue state, preventing further promotes associated with it, and sends emails to the relevant reviewers.

More broadly, the importance of process support within software configuration management has been broadly recognized and pursued. This work it is fundamentally focused on automating support in a comprehensive regulated software development

process. Static (code) analysis tools and practices provide an example of this theme.

5.2 Static Analysis Re-Imagined

Commonly, static analysis tools contribute to the discovery of defects and there is an expectation that issue tracking tools (such as ClearQuest) will be used to manage the defects they helped find [19]. Static analysis has been shown to predict defect density [20] and is often integrated into programming environments. However, the use of static analysis related to issue tracking has not been fully explored.

In particular, this work explores two related topics:

1. the potential for focusing static analysis using for context an issue or set of issues from the issue tracking system, and
2. the potential for performing something like static-analysis on issues themselves (retrospective, non-run-time, external analysis of issues aimed at identifying potential errors in those issues and their related artifacts).

Consider the review of an issue before its closure in which a reviewer examines both the issue and the changes made to software code, documents and other artifacts related to the issue. In this situation, despite the best efforts of all involved to keep mistakes out (of the issues as well as the related artifacts), it makes sense to provide some automated support for identifying them. It would be best if this was provided only, or at least primarily, within the scope of the issue(s) being reviewed. Within this area, there is not only a timely opportunity to look at the contents of the issue and artifacts with a more focused and appropriate lens, but also unique opportunity to leverage the combined data about the issue, process and related artifacts.

5.2.1 Enhanced Support for Analysis Exclusions

On the S and P teams, for example, process requires that all analysis exceptions be justified and all blocks of analysis exclusion must be explicitly defined before they are reviewed under the issue associated with their addition or modification. This makes more explicit the decisions related to suppressing analysis coverage and simplifies the review of those decisions. Generally it is impractical to expect that analysis tools check

all aspects of user requested exceptions to their analysis. Traditionally, these concerns must be checked manually (in the code or tool output) and can be easy to miss or forget to check. To better support these activities, PnitAgent automates the detection of such concerns, for example by using regular expressions like that in Listing 5.3 which can be applied to identify unjustified exemptions to the FxCop static analysis tool, that was one of the tools used by the S and P teams.

```
static readonly Regex UnjustifiedFxCop =
 new Regex(
   "SuppressMessage([\\s\\S\n](?!("+
   "Justification|"+
   "[^\\x5B]\\x5D)))*.[^\\x5B]\\x5D",
  RegexOptions.Compiled
  | RegexOptions.ExplicitCapture);
```

Listing 5.3: Unjustified FxCop suppression regex

This expression basically identifies where it sees the term "SuppressMessage" followed by anything ultimately leading up to a '`]`' character without a '`[`' character or the word "Justification" in between. Another example regular expression is shown in Listing 5.4 for detecting when analysis of a certain rule by the Tiobe ClockSharp[1] static analysis tool has been disabled and remains disabled until the end of the file because it is not explicitly re-enabled. This regular expression basically matches the multi-line rule disable comment and then (in a non-capturing group) everything up to the end of the file that is not followed by a reference to the same rule number as would be required to re-enable checking of that rule in the current file.

```
static readonly Regex UnclosedSuppression =
 new Regex(
   "CLOCKSHARP - ([0-9]*)@([0-9]*)"+
   "(?:[\\s\\S\n](?!\\1@\\2))*$",
  RegexOptions.Compiled);
```

Listing 5.4: Unclosed static analysis regex for ClockSharp

Similar techniques can be applied to support exclusions or suppressions for other static and dynamic analysis tools and extended with further reasoning support. For example, unjustified coverage exclusion attributes as used with the NCover tool can be identified and only raised as a significant concern if (in aggregate) the resulting additional uncovered lines of code cause dynamic analysis coverage to fall short of preset thresholds.

[1] `www.clocksharp.com`

5.2.2 Realizing Advantages of Issue Focus

In large, complex software there can be a great number of exemptions without justifications that are left unnoticed and/or uncorrected. Furthermore, although this checking has value, checking it alone or on a single file is not ideal. It is expected that this and a number of other concerns are checked on many files and reported together. When the goal is to review an issue, this can lead to a significant amount of extra unrelated output to wade through (see for example Figure 5.3). A better way to support such concerns is to organize and present them in the context of the issue in review and as appropriate for their relationship to it.

There may be no file changes or a specific changeset associated with an issue. In the former case, no associated artifact analysis concerns need to be considered and in the latter, only those that are in the changeset are important to the review at hand. Concerns with artifacts not associated with an issue may be de-emphasized, available upon request, not analyzed or ignored as shown in Figure 5.4. This greatly reduces the volume of output a user needs to focus on to complete an issue review. Furthermore, although the risk of false positives is extremely low in this case, by narrowing the scope of concern, not only are there fewer irrelevant true positives for the user to be concerned with, but there are fewer irrelevant false positives.

5.2.3 A Better Time for Analysis

In addition to providing support that reduces the volume of concerns to consider during review, we submit that appropriate automation and thoughtful consideration of the potential advantages of providing such support at the time of issue review can help to address timing concerns as well.

Often examples of how timing can contribute to complexity and challenges with review relate to the fact that there may be several people and issues associated with changes to a single file over the same period or that the fact that problems could have been introduced after the issue in question. Furthermore, they may have been resolved after the issue in question and, while worth noting, may not be worth rejecting an issue. To clarify that the problems found were introduced as part of the changes tracked by the issue under review and have not been resolved since, the agent supports analyzing the

latest version, the basis, a locally modified, or other specific versions, of an associated file or group of files. Furthermore, automation of such analysis and some simple related reasoning can help to identify the more interesting set of static analysis concerns that were introduced by this issue (not present in the basis) and have not been subsequently addressed in the latest version.

As another example, the automated build processes of the S and P teams enforce a number of static analysis checks automatically, but do not address certain coding standards such as related to certain security concerns that are not supported by the static analysis tools and for which compliance is not required until the software is ready for review. An example regular expression used by the agent to support recognition of exposed classes that do not have expected security attributes is shown in Listing 5.5.

```
static readonly Regex NoSecurityDemandRegex =
 new Regex(
   "namespace ([\\s\\S\n](?!"+
   "(SecurityAction|"+
   "UnitTest|generated|TestFixture)))"+
   "{0,500}\n[\\s]*(public|protected).*class",
  RegexOptions.Compiled
  | RegexOptions.ExplicitCapture);
```

Listing 5.5: Missing security demand regex for C#

Though some may desire such concerns to be enforced immediately, consensus within the S, P and other teams is that some should only be enforced at the time of issue review (after some degree of integration and automated build but before more formal integration). This is driven in part by the desire to not delay initial integration too long, but to make sure the standards are high and enforced, automatically where possible, before review.

Other important examples of concerns that are often not addressed before initial integration, but are expected at issue review, are opportunities related to the practice of code annotation and understanding of references to related and remaining work.

5.3 Code and Other Artifact Annotation

Storey et al [21] examine issues related to how to manage annotations and use them to improve tool support. Specifically, they note the relationship between comments such as TODO or FIXME and the links to code in issue tracking systems, as well as automation

support provided by development environments and frameworks. However, they stop short of describing the potential for automating checking for TODO comments at the time of issue review/closure.

We have proposed and implemented guidelines to help relate TODO code annotations with the related issue(s) and automated capability to present related TODO comments (from code as well as documents[2]) which reference the issue under review or are within the scope of the issue under review (and either reference another issue which could be important for understanding the context or do not reference an issue despite guidelines contraindicating this). In particular, we believe that the following capabilities are valuable and novel extensions of automation to support important concerns at the time of issue review:

1. identification and presentation of references to remaining work within an issue;
2. identification and presentation of references to the current issue from other artifacts;
3. identification and presentation of references to work remaining related to (e.g., in files referenced by) issues that are in a post-resolving state;
4. identification of references to work remaining that is not inside of and does not directly reference a tracking issue;
5. identification and presentation of references to nonexistent items across data sources;
6. integration of data and presentation including issue contents retrieved from the tracking system, software product source materials, documentation including spreadsheets and other formats, associated requirements retrieved from a requirement management system.

The identification of remaining work from annotations naturally put into artifacts such as software source code, documents, and the internal fields of one or more issues under review, has not been well described in the literature or supported by current tools. Major editors and integrated development environments have begun to support this kind of concern, but only in a fairly limited fashion (without regular expressions support, lacking support for association with tracking issues, lacking support for several common artifact types such as Microsoft Word documents, and so on).

[2] The detection, reasoning about, and integrated presentation of TODO-type comments even extends to the contents of the related issues in the issue tracking system.

Two key challenges for richer support in this area are being able to access and provide appropriate support for artifacts in different formats and systems (e.g., such as the issue tracking system), as well as correctly detecting not just remaining work and issue references, but the association between the two.

An example regular expression for detecting references to remaining work in basic C-style code (as may be relevant to C, C++, C#, Java and other languages) is shown in Listing 5.6. Line 3 basically matches what is considered to start a comment and text preceding indication of remaining work. In this domain, comments are almost always begun with "//" or use of "MessageBox.Show" (which is never presented to real users of the product, but is sometimes used while the system is under development). With other development languages or domains this matching may need to be changed or replaced (e.g., with "/*" multiline commenting where even newlines may precede the text indicating remaining work, but "*/" cannot).

```
1 static readonly Regex TodoCommentRegex =
2  new Regex(
3    "(MessageBox.Show|//.*"+
4    "(todo|tbd|hack|revisit|fixme|xxx))"+
5    "[\\s\\S\n]{0,120}",
6   RegexOptions.Compiled
7   | RegexOptions.IgnoreCase
8   | RegexOptions.ExplicitCapture);
```

Listing 5.6: TODO regex for C-style code

In xml-based files, which are also used in this domain for several purposes, comments are started with the "¡!–" string and an expression like that in Listing 5.7 is more appropriate.

```
"<!--.*(todo|tbd|hack|revisit|fixme|xxx)"+
```

Listing 5.7: Partial TODO regex for xml-style code

In Microsoft Office documents, which are widely used in this domain, references to remaining work are generally not called out by any special comment characters and may be highlighted, put in a special font, or, more commonly, not distinguished in any fashion (see Listing 5.8).

```
"(todo|tbd|hack|revisit|fixme|xxx|pnits)"+
```

Listing 5.8: Partial TODO regex for documentation

In Listing 5.6, line 4 basically identifies the group that matches the text that indicates remaining work. Line 5 matches additional context that can be displayed to the user.

This context can save the user time with false positives. One can imagine how long, dynamic, and prone to false positives line 4 would have to be if the (not unheard of) practice of using only initials to indicate remaining work were the only common way of indicating remaining work. On the other hand, in the relevant artifacts for the S and P team, searching only for the remaining work indicator "TBD" actually does not yield any false positives. Interestingly, it is preferred to use, and in some coding standards the teams have standardized on the use of, the indicator "TODO" which leads to false positives with relatively simple search expressions and techniques such as seen with "...ToDouble..." (or, as common in this domain, "...ToDose..."). This was commonly used before standardization in part due to default support provided by major integrated development environments. Despite some standardization and plurality preference, consensus is that broader checking for different types of remaining work references at issue review is best.

Put directly into practice, this kind of work can provide useful, but limited indication of all the remaining work associated with an artifact (e.g., see lower portion of Figure 5.3). While this can be organized by and linked to files, shown with context, and presented with varying degrees of finesse, it is not particularly well suited for use at the time of issue review. We submit that, in addition to identifying a reference to remaining work, it is much more useful to also search for and utilize information about an issue which is tracking the resolution of that remaining work.

Figure 5.3: Excerpts of example of unfocused analysis

Searching for such a connection can be difficult, but can be bounded to make that search more expedient and practical. One particularly valuable assertion is that a reference to an issue should occur in the text near the reference to remaining work. This avoids complex analysis of likely unrelated issues and artifacts and allows for search to be constrained to the same comment block or taken a bit farther, even within the context captured by the previous regular expressions. This assertion can be further bolstered by best practices in documentation and coding standards. For example, the standards used by the S, P and related teams require that known remaining work be tracked in the relevant artifact with an issue number. Additionally, some of them require, or provide examples showing, that in software source materials the term “TODO” be within a few

characters before a reference to the issue identifier.

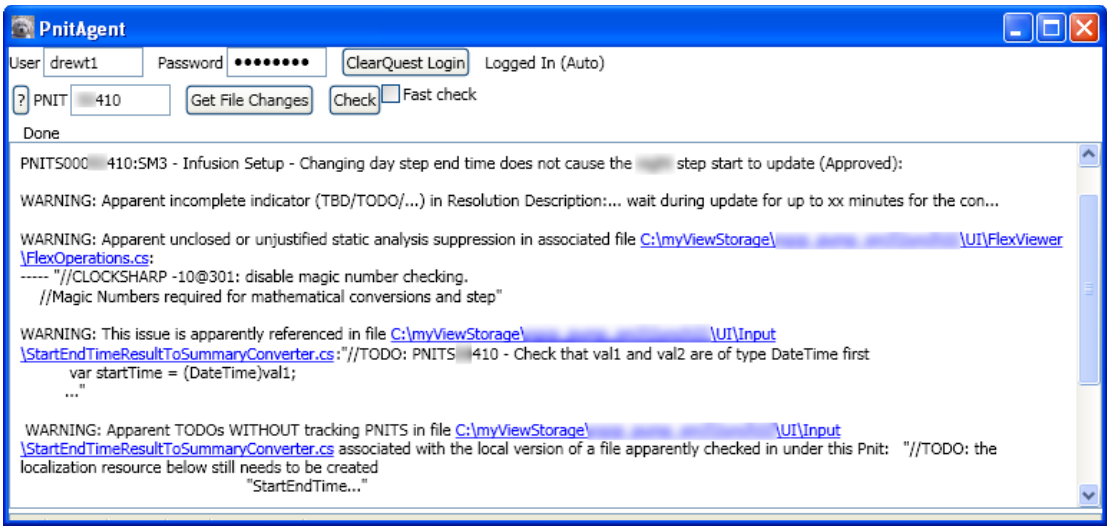

Figure 5.4: Example of issue-focused analysis

As alluded to in Section 5.2, this is even more interesting when organized and presented in the context of the issue under review, for example as shown in Figure 5.4. As described previously, this helps minimize unrelated output. Additionally, in this context by organizing, reasoning about and presenting remaining work information according to its relationship with the issue under test, it can be prioritized and its relationship explained in meaningful ways. For example, references to the issue being reviewed and those with no tracking issue are generally more important than references to other issues. Similarly, references to other issues that are in areas changed as part of this issue's change set are generally more important than references to other open issues outside of the areas associated with this issue. Also, while not directly relevant to the issue under review unless introduced by it, references on the tips of files associated with the issue under review to remaining work tracked by issues that are no longer in a resolving state reflect another sort of concern that should be raised and addressed.

Additionally, this can be extended to not only have a focusing and prioritizing effect, but to have an integrating one as well. For example, analysis of indications of remaining work occurs on the text present in the issue under review itself and these concerns are presented alongside references to remaining work in associated documentation providing

a focused and prioritized, but more complete view of the relevant remaining work. Furthermore, in addition to integration across multiple sources, this is integrated with any other related concerns (such as those related to reviewers as described in Section 5.4).

Put into practice with our software agent (e.g., as shown in Figure 5.4), this checking has lead to the appropriate creation of new issues and the rejections of others at the time of issue review. This may also have further improved understanding of the context of the issue and related artifacts under review in other ways as well.

5.4 Role Analysis for Process Support

A key opportunity and area of focus for our automation efforts relates to detailed process understanding, alignment and application within this domain. While significant process definition exists in Software Configuration Management (SCM) related tools (such as was found in ClearGuide [18] or more recently IBM's Unified Change Management, UCM [24]) and significant best practice guidelines and process definition exists for software development within the medical device industry (e.g., [8, 26]) there is a level of detail at which it is not understood how best to apply process related to issue tracking and associated activities. Indeed, more generally while detailed process support provides value, it also comes with complexity that the market struggles with. As result, fine control general process support tools (like ClearGuide) swing out of fashion and are replaced by more simplified process support (like UCM). There has also been some notice of deficiencies in process support for issue tracking systems that the capabilities we created as part of this research may begin to better address. Delugach, for example, noted that better explicit role support for issue tracking systems is needed "if we want to reason automatically about roles and their appropriateness or legitimacy [15]."

Reviewer Support

Improved role support is of particular importance as an enabler for automation of the process of ensuring the appropriateness of reviewers. The Food and Drug Administration, for example, requires "formal documented reviews" and that related "procedures shall ensure that participants at each ... review include representatives of all functions

concerned with the design stage being reviewed and an individual(s) who does not have direct responsibility for the design stage being reviewed, as well as any specialists needed. [28]" These functional, independent, and specialist reviewer roles and their appropriate satisfaction is important in the review of issues, but not well supported by issue tracking tools.

Based on formative classification of rejections sampled in March 2010, problems with missing or inappropriate reviewers of issues accounted for 11% of the overall and 13% of the (later and more process-focussed) software quality assurance (SQA) rejections. Addressing this involves understanding the situation of the issue at hand (e.g., duplicate, bug in test, requirement change, ...) and the appropriateness of its reviewers. Resolving issues with reviewers, once identified, is generally straight forward, often requiring the addition, removal, change or clarification of one or more reviewers and their roles. However, it can be a serious problem if reviewers are not correct as appropriate review is essential to ensuring compliance, quality, communication, completeness, objectivity, cross-pollination, and so on. For example, depending on the nature of an issue, it can be important to ensure any changes to product have been independently reviewed or that the problem described by the submitter is the same as that addressed by the resolver.

As another example, it can be important to ensure that changes or new additions are visible to those in other roles that may have to create tests for them, consider their broader system impacts, validate their use, and so on. As part of ensuring detailed understanding and alignment on such concerns, the table excerpted in Figure 4.8 was created and advisor agent software was updated to recognize the situation and reviewers associated with the issue at hand and warn if a certain review role appears to be unsatisfied.

Automating recognition of the situation associated with the issue at hand involves natural language processing (NLP) of free form text fields and rules based reasoning including checking the resolution classification selection, related documents, related issues, and other values associated with the issue. For example, the issue resolution classification of a issue may have had "Duplicate" selected identifying the issue as a duplicate and allowing for the expected reviewer roles to be identified and other scenario-specific concerns (such as an associated change set without redirection of review) to be raised. In another example, an issue whose resolution was classified as "New Work Completed"

or "Other" might potentially reflect one of several different scenarios that would have different associated reviewers and need further analysis of the headline, description, related issues and associated documents to identify it as a design review. In addition to identifying it as a design review, the settings associated with its related project would be accessed to determine if a software quality assurance review was needed (depending on their history and other factors, required reviewers vary slightly with teams). More generally, in each scenario a different set of expected review roles might be required and different scenario specific concerns evaluated and potentially raised.

A particularly important aspect of avoiding false positives is using natural language processing techniques to detect when review is deferred to another issue or the role of reviewers is clarified (for example explaining that a certain reviewer is acting as the independent reviewer in addition to, or despite, their normal role as a test impact reviewer). To support this, the agent finds strings that appear to be redirects and then identifies the issue they appear to redirect to so that it can be presented to the user. This is done separately for each of the relevant fields in the issue to allow the user to be referred specifically to what field the apparent redirect appears in (this is done for remaining work references as well based on similar logic; see reference to the field "Resolution Description" in in Figure 5.4).

Once the agent has classified the situation of the issue in question and that it has not redirected its review to another issue, it identifies the expected review roles and attempts to identify a reviewer that satisfies each of these roles. This can involve integrating information from several sources including a database of the jobs of certain individuals, the current project settings, the issue itself, the requirements management system, the document control system, and the version control system. For example, the agent might identify the situation as a product bug fix, use current project settings (or a conservative default if not set by the user) to determine that a software quality assurance reviewer is needed, load the list of reviewers from the issue (e.g., joining a list of identifiers from the Completed Reviewers and Reviewers Outstanding fields) and then use the jobs database to ensure at least one reviewer is a software quality assurance engineer. Checks like this are relatively simple for human reviewers, but often overlooked. To ensure an independent reviewer is present, the agent takes some similar steps, but cannot simply use the jobs database to identify appropriate reviewers.

It can eliminate software quality assurance engineers and others based on their jobs, but that alone would lead to many false negatives. So it also checks the issue contents to rule out those involved with the issue resolution. When configured to do more thorough checking it takes additional steps, for example, working with the version control system to identify the author of each change and eliminate them from the list of potential independent reviewers. As seen, for example, in Figure 5.5, whenever the agent warns about a problem with a reviewer, in addition to briefly identifying its concern, it provides a link to a website containing related guidance including the table excerpted in Figure 4.8 and a significant amount of additional content. In more complex cases, such as with a missing independent reviewer, the agent adds further explanation of its classification of reviewers (e.g., as resolvers, test engineers, or systems engineers).

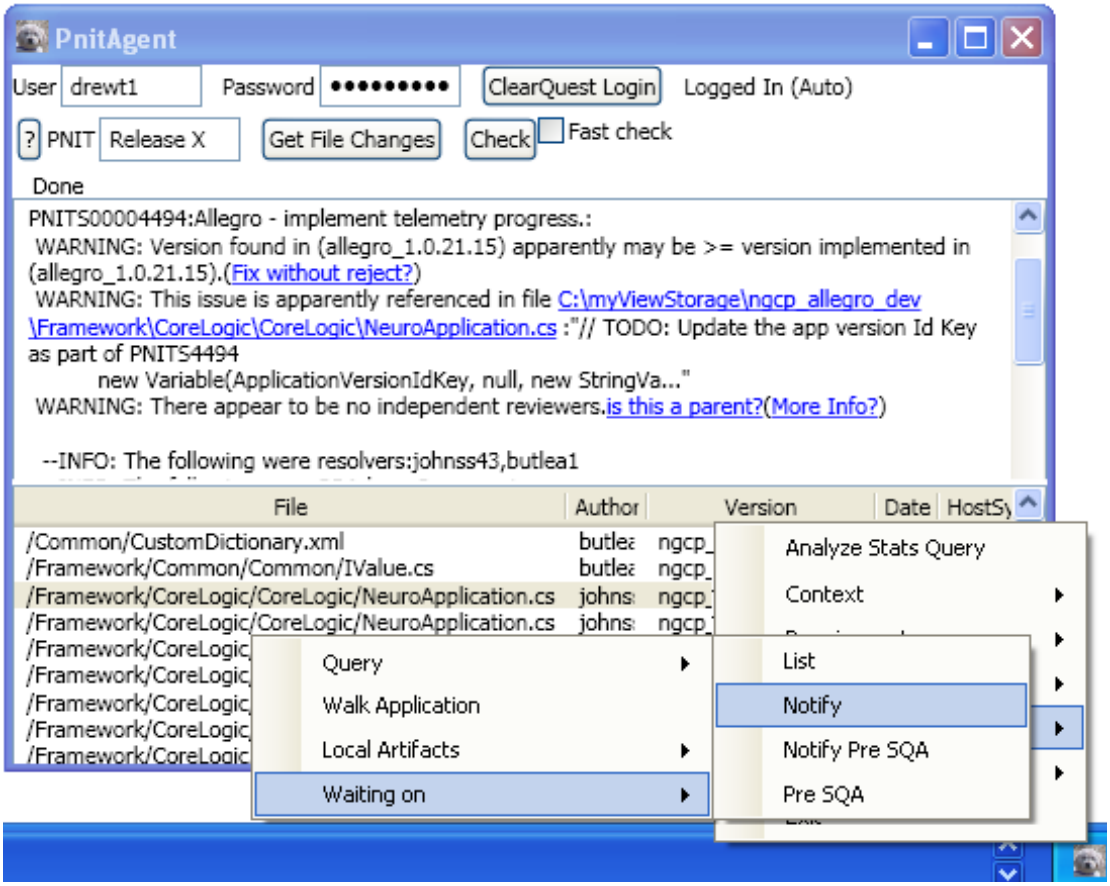

Figure 5.5: Example agent window and tray menu

Collaboration Support

Issue tracking systems are fundamentally systems to support complex collaborative work by teams of users. They provide basic support for this collaboration but allow users to safely and concurrently work with shared issue data. Additionally, as with the issue tracking system used by the S and P teams, they support sending users notifications via email about issues that may be of interest to them. It is not uncommon for a user to receive several, or even several dozen of these emails a day without any clear prioritization or indication of who and what are dependent on them.

While these are helpful to some extent, the task of appropriately remembering and prioritizing reviews requires diligence and continual research. To support this, users often create and save queries to identify different sets of issues they are interested in and use those to recall and browse through issues or report directly on who owns them, their state, their outstanding reviewers, and other information that is needed to tell who they are waiting on and for what. Users then integrate their own knowledge of related process and their colleagues jobs to determine more precisely for each issue who is responsible for the next action(s) on each issue often so that they can then gently encourage, or perhaps just monitor, progress in those areas. Sometimes, users give up on this analysis and simply poll anyone they think may be relevant.

Similar challenges exist with other work, such as the approval of documents in the document management system. Indeed, more generally, research on the information needs of developers concludes that their "most frequently sought information included awareness about artifacts and coworkers ... [and] developers often had to defer tasks because the only source of knowledge was unavailable coworkers"[56]. With research [57] suggesting that such "Interactive activities" occupy more than half of a developers day and are often unplanned, the broader significance of these needs and their effects on interrupting the work of teammates can hardly be understated. Furthermore, there is the challenge of desktop-based communication being "too slow" with the need to "type out a coherent description of the problem." These suggest that a context-aware solution could reduce context capture and communications time for more focused collaboration which is still relatively robust to human resource availability.

To better support these needs, the agent allows user to view, generally in a matter of seconds, an up-to-date list of the issues returned by a certain query. This includes

a summary of who and what each issue is waiting on next. To initiate this, the user selects an appropriate item from the agent's tray menu (see Figure 5.5). The agent then executes the query and, for each issue returned by it, identifies what the issue is waiting on (e.g., resolution, review, closure). It identifies who this action is waiting on and, for review, more specifically what people and roles are next such as independent review, test review, or SQA review (which is only expected after other non-closure review). This involves similar processing to that for checking reviewers.

Additionally the user can ask the agent to notify those for whom issues in the query are waiting. In this case, the agent generates a separate email to each person that one or more issues are waiting on (after generating a unique list of people from the relevant set of needed issue-action-person tuples). It identifies their email address before creating and eventually sending the email telling them who asked that they be notified and exactly what issues are waiting on them for what. This reduces the number of people receiving emails and the burden on each of those people to parse through extraneous information to determine what they need to do.

Finally, the agent supports more flexible and complex batches of queries (through the "Query" menu that is not expanded, but visible in Figure 5.5). This allows users to rapidly gather and at least partially process data from multiple queries potentially involving a number of different systems. This has been used by some users to support relatively sophisticated metrics gathering. In a cross-team development leadership meeting, a principal software quality assurance engineer described her use of it to support issue tracking metrics saying "This will save us at least 3 person-weeks over the next year." Another common use of this is to identify the status of design history file documents of interest to a particular project. Information on (typically a hundred or more) project-specific documents being worked on, reviewed and eventually approved by dozens of individuals is important (and previously time consuming) to track in the days and weeks leading up to major phase reviews.

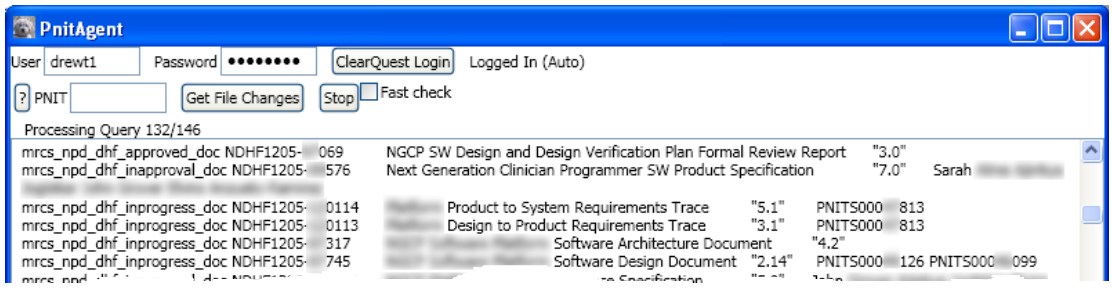

Figure 5.6: Example of Integrated Document Status Query

In addition to system specific queries, this capability allows users to rapidly integrate information from several systems, working on their behalf to collect, analyze, and format data for them in the background. In addition to simple combinations of queries to different systems, more interesting combinations and reasoning are possible. For example, the agent allows for a list of documents from the document control system to be identified and in addition to getting their approval and other status from that system, it searches the issue tracking system to determine if any open issues currently reference them so that it can be ensured that those related issues are closed as appropriate. This allows for a more integrated understanding of document readiness and for that to be tracked more automatically over time as seen in Figure 5.6.

5.5 Change Sets Without Boundaries

Committing atomic changesets affecting several files under a version control system, and association of such changesets with an issue is important and basic software configuration management (SCM) support. Similarly, associating each set of issue field changes, requirements changes, changes within documents in a document management system, and so on with the same issue is desirable and supported to varying degrees in traditional systems. Often, an issue tracks multiple version control changesets as well as changes made to other items such as requirements and documents. The resulting superset of changes, that we will call a 'change set' here, is generally not atomic (which is a problem left for other work), but is generally meant to be a highly cohesive set that is reviewed together at the time of issue review.

This is problematic because of the lack of integrated support to review all of the

changes together in as common a fashion as possible. It seems that document changes from the document management system, are often best reviewed at least somewhat along side related code changes from the version control system. Similarly, for both items, although the steps involved with showing differences is slightly different, it is generally useful to view the differences between each changed document or file version and its predecessor version. Likewise, it can be useful to view the comment associated with the commit of each document or code changeset. For example, if one is trying to determine independence manually, seeing a list of who made changes to documents in the same list as those who made changes to code under the same issue is desirable.

Review of the entire change set of an issue is supported by the agent software primarily in two different ways. The first involves providing and focusing warnings that are presented to the user based on integrated data from various different sources as described previously. The second takes the form of an integrated change listing that shows a single comprehensive list of changes regardless of what content type or host system tracked this change. This includes not only obvious, and automatically associated changes like those made in associated changesets in the version control system, but also those such as from the document control and approval system, MRCS, where design history file documents are kept, but changes cannot automatically be associated with an issue (e.g., see Figure 5.7).

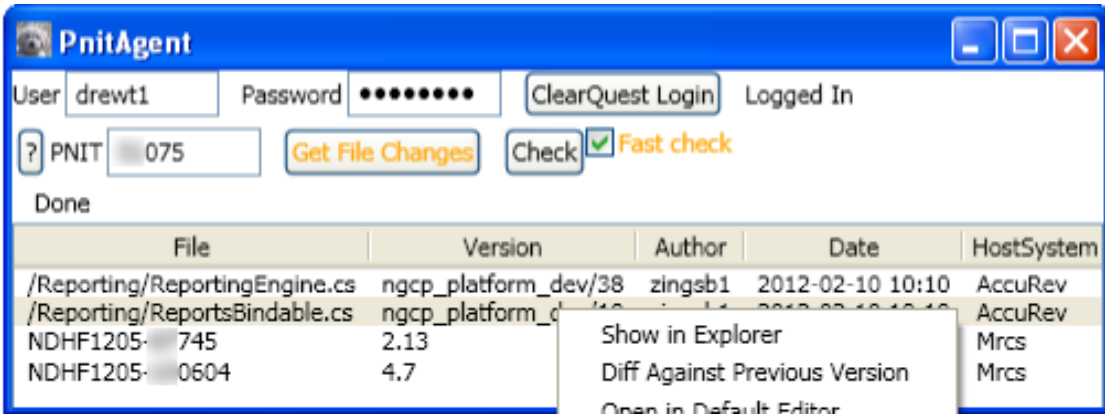

Figure 5.7: Example PnitAgent showing integrated change set listing

To enable this, certain free form text fields of the issue under review are parsed for references to MRCS documents. Double clicking on any change item in the list causes the agent to take the default review support action for that item, which generally takes the form of retrieving the changed and previous version of the files and displaying them in a graphical differencing tool, converting them if needed. Right clicking on any item brings up a context menu which exposes additional support options including those seen in Figure 5.7 and several others. To some extent these actions and options vary, for example with the host system and change item. As shown in Figure 5.8 if the change item was an MRCS document identified by analyzing the issue content, it is possible that some essential information, such as the version number, could not be found. For such items, only very limited actions are supported by the agent. Though something like this often reflects a problem with the documentation of an issue and it may be appropriate to make that apparent, when automating support for systems with natural language inputs, flexibility of this sort is essential.

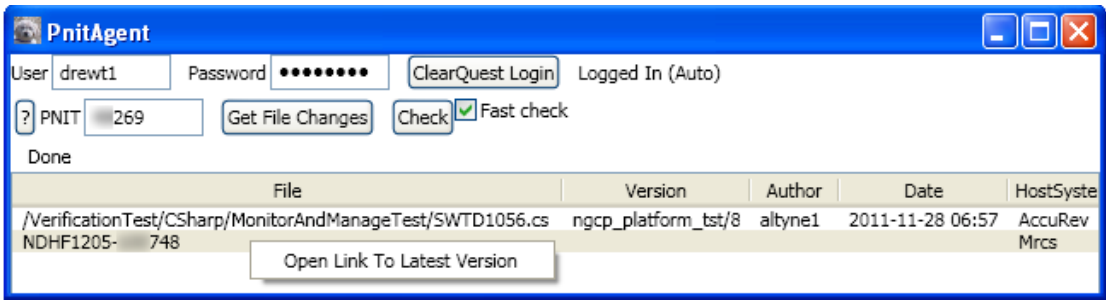

Figure 5.8: Example change set with missing data

Another challenge with integrated data from multiple sources and software that operates on behalf of varied users is the need for robustness despite the complexities of resource availability and environment. In addition to other details described in [66], flexibility in the face of partial failure and well thought out error handling have proven to be very useful in PnitAgent. For example, the agent prefers to access the version control system and issue tracking system through COM and other low level interfaces that tend to result in greater performance, but these have installation dependencies that are not met on some users computers. When these interfaces are unavailable, the agent briefly explains the situation to the user and attempts to use command line clients or

web interfaces through a headless browser. Depending on its level of success, the agent alters its user interface removing options, alerting the user, or taking other actions such as changing the color of user interface elements (see Figure 5.7), to indicate that the associated actions will have slow performance or other limitations.

This may seem like a great deal of flexibility and integrated functionality to provide change set support with as few boundaries as possible, but there is always a desire for more. Important areas of current and future work include improved, integrated support for looking at changes in the content of issues themselves and at sets of changes in requirements. For example, after an issue is rejected and further addressed, it typically returns to the same reviewers. In such cases, users often want to see exactly what's changed in the content of the issue as well as in the related artifacts. In addition to support "since the last time I reviewed this," it is desirable to look over longer periods with more regular measurements to understand the status and flow of work through the issue tracking, requirements management and other systems tracking the work of development teams. Some topics related to these concerns are discussed next.

5.6 Automation to Increase Project Status Accessibility

In this section we describe our experience with automating support for data mining and analysis to provide improved understanding of more general software development status. We attempt to describe some of our efforts to facilitate the mining and refining of information about recent and longer term changes in issues and requirements.

5.6.1 Effort Management

The software agent involves improved support related to the analysis, tracking and management of effort. This goes beyond the details tracked by annotations as described above and into what might be traditionally associated with project management. Prior to the last decade, the use of almost exclusively plan-driven effort management was used within the company. Recently, more agile methods of estimation and effort tracking have played an increasing role with many coming to the conclusion that more empirical effort management is a key part of "a better way" to pursue software development in this domain [22].

Though some teams in this domain have attempted to exclusively use empirical approaches to effort management, ultimately most teams working on significant cross-functional projects return to some hybrid form of empirical and defined/plan-driven effort management processes. This appears to be consistent with the more general SCM literature: "Traditional project management systems are not ideal for software projects ... In contrast, workflow systems [are based on] the request backlog, and the average time to process a request. Some aspects of software projects are best modeled by workflow systems, for example, incoming defect reports [18]."

How to most effectively find this balance between empirical and defined (or workflow and project management) systems and the complexity and overhead of maintaining and pursuing them both effectively in practice is often not clear. As the domain in question requires that all or nearly all work done on nearly all major artifacts (including code, review, and documentation) be tracked by the issue tracking system, we submit that the issue tracking system provides an existing mechanism and data repository for most of the major items associated with effort in our area of focus.

Furthermore, while the issue tracking system contains a mix of issues associated with scheduled and unplanned tasks and has not be designed to most effectively support effort management activities, it can be used for this purpose in combination with our software. In particular, the agent can support effort analysis, tacking and management effectively with relatively low overhead for its users.

PnitAgent supports a range of activities from tracking resolution and review velocities within areas of interest to identifying changes and imbalances in team and individual effort backlogs, to reporting accomplishments at a high level of detail, to identifying who and for what a body of work is waiting, to supporting collaborative team work through communicating requests and priorities and so on. This involves, for example, allowing users to provide effort estimates in the free-form text of their issues, the agent parsing this information out and, when such information has not been provided, the agent estimates remaining effort. Agent estimation can involve attempting to determine the scenario of the issue, who owns it, who is expected to review it and so on. In this way it allocates effort for the resolver if no estimated effort has been identified and for each outstanding reviewer regardless. The agent then allows for aggregates of these efforts to be reviewed across certain groups of issues or the people who currently own

(or it expects will own) the relevant activities.

5.6.2 Open Product Code Issues

Although issues provide a measure of the known work remaining to be completed as part of a project, it is often valuable to examine only the portion of those issues that are still open and have caused, are causing, or will cause changes to the product. These reflect the amount of known change left for the product as well as the visible risk of further defect introduction through resolution of these issues. Viewed as a time series, for example as shown in Figure 5.9, this may be a helpful indicator in gauging the convergence of a software product's readiness for release. This is not trivial to automate.

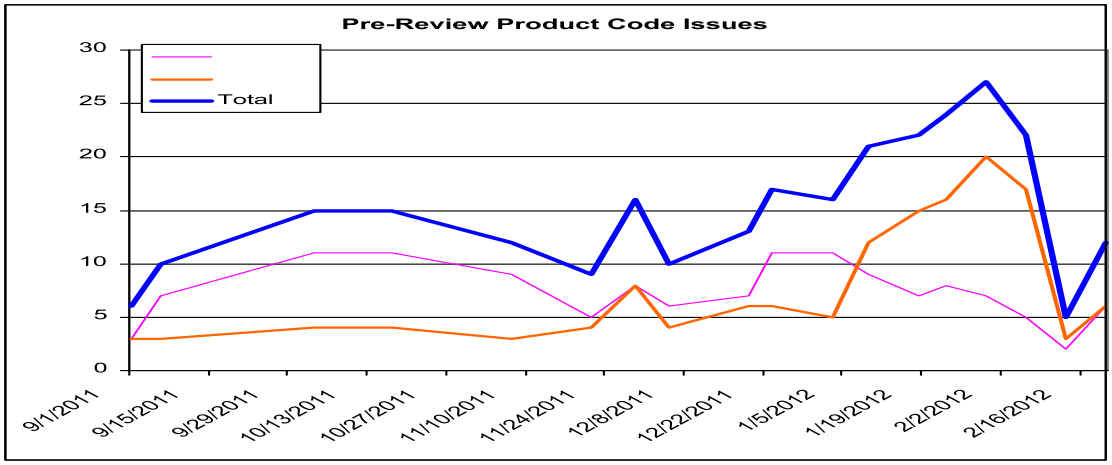

Figure 5.9: Example plot of open product code issues from two related projects

Issues that are sufficiently far along in their lifecycle can be readily analyzed to determine if they have had associated code changes. Identifying which of these code changes are in the product and which are not can be leveraged from scenario determination analysis described previously. With less mature issues (such as those that have just been submitted and/or are largely incomplete), it is more difficult even though these typically more recent issues are of great interest to current decision making. To allow for this additional analysis, the agent seeks advise from the user attempting to keep related user burden to a minimum. To do this, the agent gets the latest data on

each issue and classifies all sufficiently mature issues and those that are not mature but already have product code changes made against them. It then looks for a cached answer to whether the issue is expected to affect product code. If found, that is used to classify the issue until it is more mature. If no cached answer is available, the agent prompts the user and persists their response in the cache.

In other situations, the agent caches answers to questions that come from intensive processing or interactions with other systems (e.g. forensic audit trail access and parsing). In addition to persisting simple answers, the agent applies the same principle with somewhat more complex data and techniques to capture "baselines" of issue tracking, requirements and other information. This has several advantages including the ability to be more efficient or robust in the face of resource access challenges and even bridge between tools systems due to its abstraction and persistence of relevant information. Additionally, it can later use baselines to provide capabilities that are difficult (or impossible) to reconstruct with the underlying systems alone. This includes, for example, support for rapidly determining what has changed in a group of issues or requirements week to week as described in the following sections.

5.6.3 Issue Change Tracking

Using the agent's support, high level information such as that shown in Figure 5.10 can be synthesized to support intuitive measures of the growth rate and volume of overall issue tracked work, unresolved issue or review debt a project has and how many reviews and rejections are being performed by a team (even when that does not immediately translate into changes in high-level issue state). Used at and even higher level of abstraction and aggregation, these are useful for not only consistent presentation of related concerns across projects, but for resource levelling and insight into the impacts of project priority calls. Such information is also further extrapolated to determine if a team is keeping up in different areas or even predict a rate of convergence (e.g., based on recent rate of change or more complicated models of issue discovery and completion).

These can be intuitive and useful, but not trivial to automate. For example, rapidly analyzing and sub-classifying a set of issues over time can be difficult such as when interested in issues related to a feature or discipline that is not well classified by the system or as described when working to observe only Open Code Issues. Additionally,

the agent supports rapidly and retrospectively.

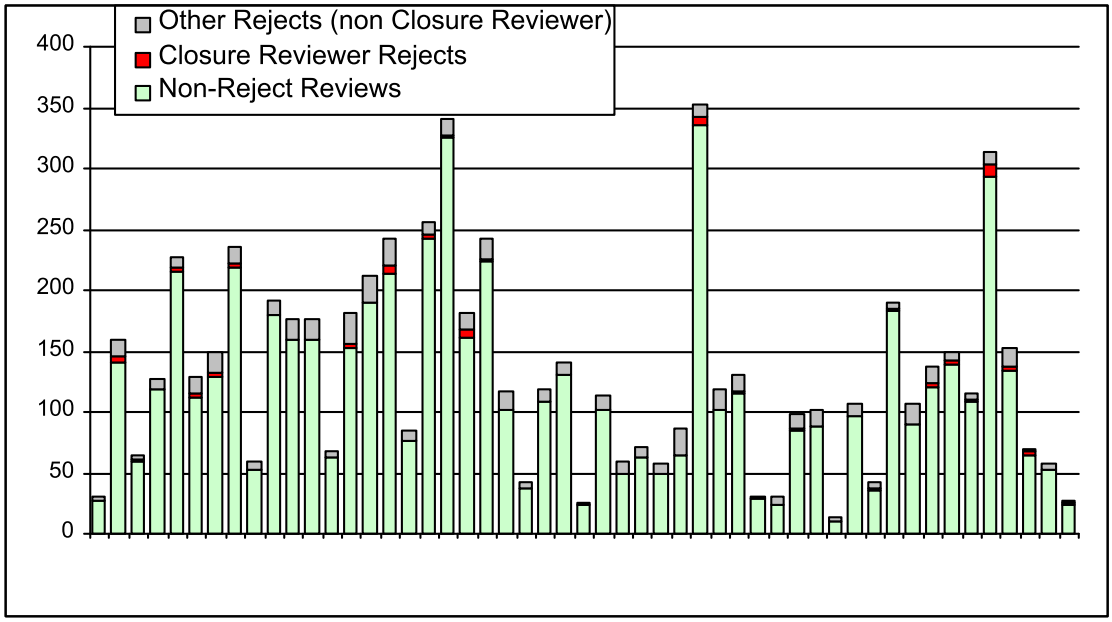

Figure 5.10: Example plot of review rates each week

Similarly, the agent supports much more detailed analysis of change in a set of issues. For example, the agent is used weekly by the S and P teams to classify and summarize rejections that occurred and major changes in issues. This has been particularly useful when concerns are less focused on issue debt (which the team may be fast enough to keep at zero) than the rate of incoming product code issues that may keep the team busy and the schedule slipping even when the total count of issues is always low. Similarly keeping up with recent rejections allows software leadership to recognize themes and respond to process concerns in a timely fashion.

5.6.4 Requirements Change Tracking

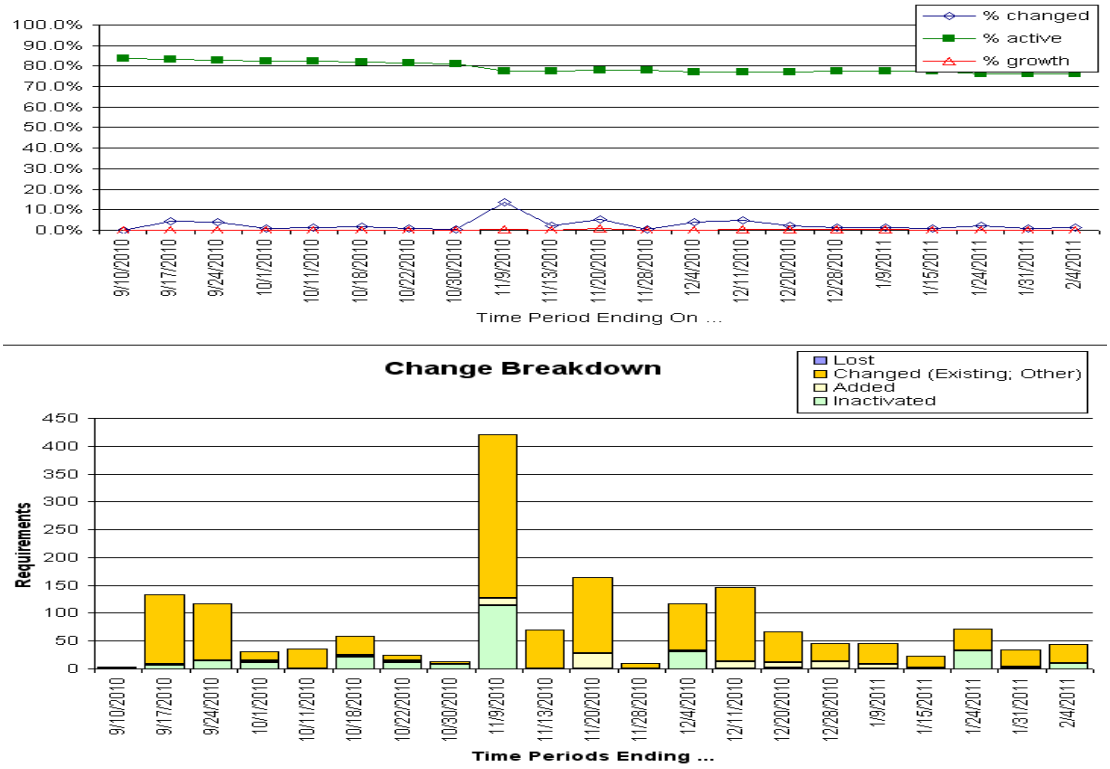

Figure 5.11: Example change in all requirements

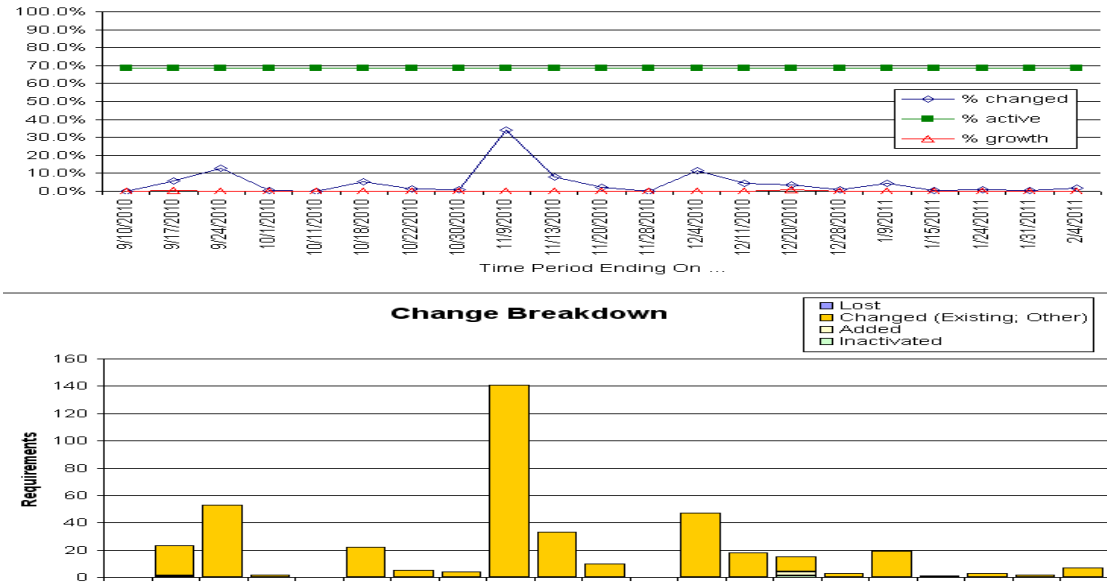

Figure 5.12: Change in low level requirements only

In much the same way that it supports tracking change in issues, the agent automates analysis of change in requirements. This involves identifying, classifying and quantifying different sorts of change over time. This is applied across all traceable items managed by the requirements management system, which can be viewed overall such as seen in Figure 5.11. This is decomposed into groups of interest such as inputs, system-level requirements, low level requirements (see 5.12), design, and test records. These can be used to observe scope change, churn, the ripple of changes, convergence at different levels, and more generally the contribution of a more specific set of traceable items to the overall picture. At a more detailed level, the agent helps to support review of change in tracing, attributes and requirement content for a specific requirement or group of requirements.

Chapter 6

Data, Analysis and Evaluation of Impact

Formative rejection data and related classification analysis used in identifying major themes in sources of waste were previously discussed in the Identifying Sources of Waste section. This section focuses on other data that is relevant to validating (or invalidating) the proposed sources of waste including additional data taken from the issue tracking system and software created to address the identified sources of waste. The following subsections present data and basic analysis of that data, followed by an evaluation that ties together the more detailed data and analysis, and finally additional data related to issue complexity and its potential relationship with rejection.

6.1 Data

This section presents a variety of data and basic analysis. First, this includes background on sample selection criteria and definition of key time periods. Then, it presents Basic Issue and Reviews Data (Subsection 6.1.3) as well as Usage (Subsection 6.1.4) data reported by the software agents before examining their relationship in Use vs Basic Review Data (Subsection 6.1.5). Some additional perspectives on issue tracking data are then presented in Volume of Reviews (Subsection 6.1.6) and Relative Timing of Rejections (Subsection 6.1.7). Finally, rejection classification data related to the nature of rejections and anecdotal data are presented in Change In Rejection Classification

Data (Subsection 6.1.8) and Anecdotal Data (Subsection 6.1.9).

6.1.1 Issue Subset Selection Criteria

This thesis examines and compares various aspects of different sets of issues. The set selection criteria create primarily two groups: those from before the bias of the targeted education and agent support provided as part of this research, and those from after the benefit of them.

As a primary focus of this thesis is on the review and rejection of issues, we attempt to account for the fact that some issues are submitted and closed in less than a day while others are not reviewed, or are only partially reviewed, for a period of months or even years. This variability can complicate the definition of criteria used to partition subsets of the issues. To address this, the method for selection used in this thesis is based primarily on the union of three criteria:

- **First Review After** - The date of the first review performed on each of the issues included in the group must have been completed after this date. This first review could have been a rejection, self-review, peer review, or any other kind of review. This is used to exclude issues that are more mature on, or at least partially reviewed before, a certain date. For example, the *after* period uses this to exclude issues that were first reviewed before 16 July 2010.

- **First Review Before** - The date of the first review performed on each of the issues included in the group must have been completed before this date. This first review could have been a rejection, self-review, peer review, or any other kind of review. This is used to exclude issues that were less mature on, or had not been at least partially reviewed before, a certain date.

- **SQA Review Before** - The date of the first SQA review performed (by a software quality assurance engineer) on each of the issues included in the group must have been completed before this date. If no SQA review had been performed at that time (or to date) then the issue is excluded. This is used to exclude issues that were less mature on, or had not been SQA reviewed before, a certain date. For example, the *before* period uses this to exclude issues that were not SQA reviewed before 1 January 2010.

6.1.2 Definition of Time Periods Including *Before* and *After*

In this thesis, various quantitative data and results are discussed in the context of different time periods. This section defines these key time periods.

The period *before* the impact of the agent establishes a baseline from which changes can be measured. Although there are decades of earlier issue tracking and several dates might be considered as the start of the impact of this research, we more precisely bound the *before* period as follows: issues in the before period include the oldest in the project database (in May of 2006) through those that were first reviewed by SQA before 1 January 2010.

The PnitAgent software continues to be used, there may be other ongoing impacts of the activities associated with this thesis, and the additional support and education associated with this thesis were not introduced or adopted overnight. Despite this, we more precisely bound the *after* period as well. Issues first reviewed after PnitAgent was formally announced on 16 July 2010 (see Section 4.5) that were reviewed by SQA before 14 March 2011 are considered to be part of the *after* period.

In addition to the *before* and *after* period this thesis examines some additional time periods:

- **Beta testing** - Reviewed before 16 July 2010 - Issues reviewed before the start of the *after* period, but potentially biased by the activities associated with this thesis. During this period there was limited use of the evolving prototype PnitAgent software (primarily by the author of this thesis) before it was formally announced (see Section 4.5).

- **First month after** - First reviewed between 16 July 2010 and 16 August 2010 - A subset of the *after* period, comprising the first month after of the PnitAgent software was formally announced (see Section 4.5).

- **Last three months after** - Issues in the *after* period first reviewed after 14 December 2010 - A subset of the *after* period including only issues first reviewed after 14 December 2010 and SQA reviewed before the end of the after period (14 March 2011).

6.1.3 Basic Issue and Reviews Data

This section presents basic information about the number of issues and some details of their reviews. It examines the data overall, as seen in the last row of Table 6.1 and across different subsets.

Table 6.1: Issues identified by timing criteria and their related reviews for the S team sampled up through March 14, 2011. The last row, row 6, reflects all issues that have had at least one SQA review. The previous rows reflect a chronological progression of interesting subsets of these issues including those: prior to bias from this work (row 1); prior to PnitAgent Announcement, PNIT Tiger Team Findings, and Related Training (row 2); first reviewed in month afterwards (row 3); first reviewed any time afterwards (row 4); and (in row 5) first reviewed in the last three months of the sample (after SQA has stopped reviewing most issues).

	Issue Selection Criteria			**# of Issues**	**Reviews**				
	First Review		SQA Review		Total	Rejects		Before Reject	
	After	Before	Before			#	%	Pass	Reject
1	1/1/01	1/1/10	1/1/10	355	1954	336	17.2%	3.545	0.875
2	1/1/01	7/16/10	7/16/10	1086	6531	975	14.9%	3.870	0.778
3	7/16/10	8/16/10	3/14/11	106	539	63	11.7%	3.857	0.349
4	7/16/10	3/14/11	3/14/11	517	2668	267	10.0%	3.599	0.352
5	12/14/10	3/14/11	3/14/11	44	198	18	9.1%	3.056	0.167
6	1/1/01	3/14/11	3/14/11	1862	10729	1436	13.4%	3.787	0.651

Note that in Issues and their Reviews from Various Times the trend in rejection rates shows improvement. Also, the number of passed reviews prior to a rejection is getting smaller (possibly reflecting earlier and more consistent rejection). The average number of rejections prior to the current rejection is also getting smaller (reflecting fewer repeat rejections).

There was a significantly lower average reject rate in reviews performed on issues first reviewed *after*[1] the announcement of PnitAgent when compared to those reviews that were completed on issues that were SQA reviewed *before*[2] the announcement of PnitAgent (see H_1). Here the null hypothesis is that there is no difference in the average number of rejections per review in these two populations of reviews. This is expressed in the equation below where x represents the *before* population of reviews, y represents the *after* population of reviews, and μ_x and μ_y are the respective means.

$$H_0 : \mu_x - \mu_y \leq 0$$

The research hypothesis is that the average reject rate in the reviews of issues from *before* is greater than from *after*.

$$H_a : \mu_x - \mu_y > 0$$

The test statistic is calculated as follows yielding a p-value of $7.5921 * 10^{-12}$ indicating extremely high confidence:

$$\left(\frac{(\overline{x} - \overline{y}) - 0}{\sqrt{\frac{s_x^2}{n_x} + \frac{s_y^2}{n_y}}} \right)$$

Here, n_x and n_y represent the sample sizes of the *before* and *after* population samples respectively. Additionally, $\overline{x}$ and $\overline{y}$ represent the sample averages, and s_x and s_y represent the sample standard deviations. More precisely:

$$\overline{x} = \frac{\sum x}{n_x}$$

and

$$s_x = \sqrt{\frac{\sum (x - \overline{x})}{n_x - 1}}$$

[1] These are reflected in the fourth row of Table 6.1.
[2] These are reflected in the first row of Table 6.1

Table 6.2: Issues identified by timing criteria and their related reviews by software quality assurance for the S team sampled up through March 14, 2011. See also Issues and their Reviews from Various Times.

Issue Selection Criteria			**SQA Reviews**		
First Review		SQA Review	Total	Rejects	
After	Before	Before		#	%
1/1/01	1/1/10	1/1/10	469	93	19.8%
1/1/01	7/16/10	7/16/10	1325	186	14.0%
7/16/10	8/16/10	3/14/11	114	7	6.1%
7/16/10	3/14/11	3/14/11	559	39	7.0%
12/14/10	3/14/11	3/14/11	49	5	10.2%
1/1/01	3/14/11	3/14/11	2190	271	12.4%

As with the entire set of reviews, there was a significantly lower average SQA reject rate in reviews performed on issues first reviewed *after*[3] the announcement of PnitAgent when compared to those from SQA reviewed *before*[4] that period (p-value $4.7335 * 10^{-7}$; see H_{1a}).

Table 6.3: Issues identified by timing criteria and their related closure reviews for the S team sampled up through March 14, 2011. See also Issues and their Reviews from Various Times.

Issue Selection Criteria			**Closure Reviews**		
First Review		SQA Review	Total	Rejects	
After	Before	Before		#	%
1/1/01	1/1/10	1/1/10	403	48	11.9%
1/1/01	7/16/10	7/16/10	1184	99	8.4%

Continued on next page

[3] These are reflected in the second row of Table 6.2.
[4] These are reflected in the fourth row of Table 6.2

Table 6.3 – continued from previous page

Issue Selection Criteria			Closure Reviews		
First Review		SQA Review	Total	Rejects	
After	Before	Before		#	%
7/16/10	8/16/10	3/14/11	110	4	3.6%
7/16/10	3/14/11	3/14/11	504	16	3.2%
12/14/10	3/14/11	3/14/11	40	1	2.5%
1/1/01	3/14/11	3/14/11	1972	140	7.1%

As with the entire set of reviews, and the set of SQA reviews, there was a significantly lower average closure reject rate in reviews performed on issues first reviewed *after*[5] the announcement of PnitAgent when compared to those closure reviewed *before*[6] that period (p-value $1.8908 * 10^{-6}$; see H_{1b}).

This measurable improvement is suggestive that the support provided has been effective in reducing rejections and, presumably, therefore waste in this domain. However, based on these significant improvements in average rejection rates alone, the benefit of PnitAgent is not clear. One might imagine that these improvements could have been caused completely by factors other than those related to the PnitAgent support software intended to address the causes of waste in issue tracking. To address this, the following sections introduce usage data.

6.1.4 Usage

This subsection introduces data collected about use of the agent software provided. By the time it was announced to its potential users, the agent software was capable of tracking basic aspects of usage and attempting to email related information back for analysis. An example of such a report can be seen in Figure 6.1. Various situations could prevent this data from being reported (e.g., users could disable or interfere with this reporting, at least one very early user did not update their software to a version that supported reporting, and so on). However, a number of emails were received from

[5] These are reflected in the second row of Table 6.3.
[6] These are reflected in the fourth row of Table 6.3

agents running on the machines of users. Though it is to some extent incomplete and limited, it can be thought of as a conservative reflection of the actual use of the agent and subsequent descriptions of agent use will be based solely on this data, except were explicitly noted.

```
From: Pnit Agent [mailto:drewt1@medtronic.com]
Sent: Friday, March 18, 2011 12:45 PM
To: drewt1@medtronic.com
Subject: Some Friendly Thoughts About PNITS

Messages for drewt1:
[username removed]
0.133
||||| Start Log.txt|||||
12:44:27|Debug|Initializing...|
12:44:27|Debug|Initializing CQ Client|
12:44:27|Debug|Initializing CQ login|
12:44:41|Debug|Initializing Settings|
12:44:41|Debug|Initializing AccuRev Client|
12:44:41|Debug|Initializing tray icon|
12:44:56|Debug|Notifying usage @ 41|

||||| End Log.txt|||||
||||| Start Resources\Use.dat|||||
<PnitAgent.LocalMachine.UseData _assembly="PnitAgent;mscorlib"
IssuesChecked="186" IssueArtifactChecks="53" LastUsed="2011-03-18"
DaysUsed="41" ReportsGenerated="6" LinksFollowed="1" DaysCollected="1"
BaselinesCreated="9" BaselinesCompared="9" AnnotationsShown="33" />
||||| End Resources\Use.dat|||||
```

Figure 6.1: Example of an email received from an agent reporting on its usage.

As of March 18, 2011 emails reflecting use by more than 20 unique users had been received. The majority of these users were, or had been full time employee developers on the S and P teams. A few were contractors, from other teams, or were non-developers (e.g. testers, managers, or human factors engineers). Although one human factors engineer on the S team would be considered a "regular user," the eight "frequent users" were all S and P developers (see Use vs Basic Review Data (Subsection 6.1.5)). For all but one of the users there was an email reporting use within the last two weeks (between March 10, 2011 and March 18, 2011). Their median reported days of PnitAgent use was 84 'work days' (where a new 'work day' of use was defined to start with any detected use more than 12 hours after that of the start of the last 'work day' of use).

6.1.5 Use vs Basic Review Data

In this section, we consider four different ways to divide populations of reviews based on how much their resolvers had used PnitAgent as 14 March 2011(see H_2, H_{2a}, ...).

1. **ever vs never** - this separates reviews into two populations based on whether there was *any* (agent email) report of the person who resolved the issue under review using PnitAgent.

2. **regular vs not** - this separates reviews into two populations based on whether there was enough (agent email) reported PnitAgent use by the person who resolved the issue under review to qualify them as a regular user. A subset of those who ever used PnitAgent qualify as regular users, because their agents had reported more than 5 days of use and 12 issues checked. About half of users were regular users. For these regular users the primary use appears to have involved Automated Issue Analysis (directly and/or through Hybrid Interactive Support) with a median rate of 2.04 issues analyzed per work day although all of them also used agent for issue related artifact review and all but three for Annotation and Sequential Support.

3. **frequent vs not** - this separates reviews into two populations based on whether there was enough (agent email) reported PnitAgent use by the person who resolved the issue under review to qualify them as a frequent user. A subset of those regular users of PnitAgent qualify as frequent users, because their agents had

reported more than 10 days of use and 20 issues checked. There were eight frequent users. Their primary use also appears to have involved Automated Issue Analysis (directly and/or through Hybrid Interactive Support) with a median rate of about 3.7135 issues analyzed per work day although all of these 8 also used the agent for issue related artifact review and all but three for Annotation and Sequential Support.

4. **power vs not** - this separates reviews based on whether there was enough (agent email) reported PnitAgent use by the person who resolved the issue under review to qualify them as a power user. A subset of those frequent users of PnitAgent qualify as a power users, because their agents had reported more than 25 days of use and 100 issues checked. There were six power users. Their primary use also appears to have involved Automated Issue Analysis (directly and/or through Hybrid Interactive Support) with a median rate of about 5.5145 issues analyzed per work day although all of these six users also used the agent for issue related artifact review and all but one for Annotation and Sequential Support.

Table 6.4: Issues identified by timing and resolver use of PnitAgent criteria and their related reviews for the S team sampled up through March 14, 2011.

Resolver	**Issue Selection Criteria**		**# of**	**Reviews**				
by use of	Reviewed...		**Issues**	Total	Rejects		Before Reject	
PnitAgent	First After	by SQA Before			#	%	Pass	Reject
ever	7/16/2010	3/14/2011	466	2386	225	9.4%	3.724	0.316
never	7/16/2010	3/14/2011	51	282	42	14.9%	2.929	0.548
regular	7/16/2010	3/14/2011	264	1390	110	7.9%	3.982	0.291
not regular	7/16/2010	3/14/2011	253	1278	157	12.3%	3.331	0.395
frequent	7/16/2010	3/14/2011	204	961	85	8.8%	3.682	0.329
not frequent	7/16/2010	3/14/2011	313	1707	182	10.7%	3.560	0.363
power	7/16/2010	3/14/2011	171	816	72	8.8%	3.653	0.292
not power	7/16/2010	3/14/2011	346	1852	195	10.5%	3.579	0.374

Within the set of reviews first reviewed after the announcement of PnitAgent, there was a significantly lower average reject rate in reviews performed on issues that had been resolved by someone who had used PnitAgent[7] when compared to those that had been resolved by someone who had never used PnitAgent[8] (p-value $6.6418 * 10^{-3}$). Similarly, those in the (resolved by) regular PnitAgent users population had a significantly lower average reject rate than those (resolved by) users that were not regular PnitAgent users (p-value $9.3261 * 10^{-5}$). Although with much lower confidence (significant with $\alpha = 0.1$ but not $\alpha = 0.05$), the data also suggests that there may be a lower average reject rate for those resolved by frequent users (p-value $6.2188 * 10^{-2}$) and even power users (p-value $8.1587 * 10^{-2}$).

Table 6.5: Issues identified by timing and resolver use of PnitAgent criteria and their related software quality assurance (SQA) reviews for the S team sampled up through 14 March 2011.

Resolver by use of	**Issue Selection Criteria** Reviewed...		**SQA Reviews** Total	Rejects	
PnitAgent	First After	by SQA Before		#	%
ever	7/16/2010	3/14/2011	499	30	6.0%
never	7/16/2010	3/14/2011	60	9	15.0%
regular	7/16/2010	3/14/2011	277	11	4.0%
rare	7/16/2010	3/14/2011	282	28	9.9%
frequent	7/16/2010	3/14/2011	216	10	4.6%
not frequent	7/16/2010	3/14/2011	343	29	8.5%
power	7/16/2010	3/14/2011	179	7	3.9%
not power	7/16/2010	3/14/2011	380	32	8.4%

When further constrained to SQA reviews, those reviews in the *after* period with resolvers that are 'ever,' 'regular,' 'frequent,' or 'power' users of PnitAgent appear to

[7] These are reflected in the first row of Table 6.4.
[8] These are reflected in the second row of Table 6.4

have lower average (SQA) reject rates (with p-values of $2.9742 * 10^{-2}$, $2.6456 * 10^{-3}$, $3.2805 * 10^{-2}$, and $1.3373 * 10^{-2}$ respectively).

Table 6.6: Issues identified by timing and resolver use of PnitAgent criteria and their related closure reviews for the S team sampled up through 14 March 2011.

Resolver by use of	**Issue Selection Criteria** Reviewed...		**Closure Reviews** Total	Rejects	
PnitAgent	First After	by Closer Before		#	%
ever	7/16/2010	3/14/2011	452	14	3.1%
never	7/16/2010	3/14/2011	51	1	2.0%
regular	7/16/2010	3/14/2011	248	4	1.6%
rare	7/16/2010	3/14/2011	255	11	4.3%
frequent	7/16/2010	3/14/2011	188	4	2.1%
not frequent	7/16/2010	3/14/2011	315	11	3.5%
power	7/16/2010	3/14/2011	158	3	1.9%
not power	7/16/2010	3/14/2011	345	12	3.5%

The data on reject rates in the closure reviews of issues that had been resolved by team members who used the agent to a greater extent was limited. Although the data may seem suggestive, the significance of greater improvement was less clear (with p-values of $7.0374 * 10^{-1}$, $3.6440 * 10^{-2}$, $1.7810 * 10^{-1}$, and $1.4138 * 10^{-1}$).

One might imagine, that some characteristic of those who used PnitAgent and used it more frequently, might have caused them to have lower rejection rates even if they had not used PnitAgent. For example, one might imagine that those who chose to use PnitAgent were simply more savvy, familiar, and/or methodical to begin with and that difference would have caused them to have lower rejection rates to begin with.

This research aims to explore the possibility that use of PnitAgent is just correlated with, rather than responsible for, reduced rates of rejection. To this end, it is useful to consider if there is any greater improvement in the rate of rejection from *before* and *after* in the population of those issues resolved by individuals who more heavily used

the agent software[9] when compared to the improvement in issues resolved by of those who did not.

Table 6.7: A summary of the rates of improvement in average rate of rejection from *before* to *after* by relative use of PnitAgent.

Resolvers	**Improvement In Reject Rate**	
by use of PnitAgent	Overall	SQA
ever	44.3%	69.1%
never	20.9%	33.1%
regular	50.9%	78.9%
rare	38.2%	57.5%
frequent	44.9%	73.6%
not frequent	44.2%	66.9%
power	41.0%	78.2%
not power	42.4%	59.8%

To determine significance in this case, the null hypothesis is that the improvement (from *before* to *after*) of heavier users is the same as that of those who used PnitAgent less frequently. The research hypothesis is that the heavier users had a greater rate of improvement.

For the overall review, it appears that there may have been significantly greater improvement where resolvers had ever (as opposed to never) used PnitAgent (p-value of $5.3827 * 10^{-2}$). There was no statistically significant increase in rate of improvement for other partitions based on resolver usage of PnitAgent (p-values of $3.0670 * 10^{-1}$, $8.4292 * 10^{-1}$, and $9.0086 * 10^{-1}$ for the regular, frequent and power usage distinctions respectively).

For the SQA review, it appears that those that ever used PnitAgent had greater improvement in rejection rate that was of borderline statistical significance with $\alpha = 0.1$ (p-value of $1.0904 * 10^{-1}$). As with the overall reviews, there was no statistically

[9] Here, as with previous discussion, we are referring to use in the *after* period only.

significant increase in rate of improvement for other partitions based on resolver usage of PnitAgent (p-values of $2.8689 * 10^{-1}$, $9.5667 * 10^{-1}$, and $2.6089 * 10^{-1}$ for the regular, frequent and power usage distinctions respectively).

In this context, note that while it may not be statistically significant in some cases, use of PnitAgent by resolvers appeared to only further improve or leave approximately the same the rate of improvement in rejection when compared to those who used the agent less.

6.1.6 Volume of Reviews

It is interesting to consider the number of reviews per issue that occurred and whether these were greater or less *after*[10] the announcement of PnitAgent when compared to those from *before*[11] .

Table 6.8: The volumes of different types of reviews per issue from *before* and *after* introduction of PnitAgent to the S team sampled up through March 14, 2011.

	Before		**After**		**Change**	
	Average	Dev.	Average	Dev.	Direction	p-value
Total Reviews	5.504	3.518	5.161	1.939	Less	$4.7020 * 10^{-2}$
SQA Reviews	1.321	0.779	1.081	0.287	Less	$1.4308 * 10^{-8}$
Self Reviews	1.014	1.618	0.983	1.234	Similar	$3.7825 * 10^{-1}$
Test Reviews	1.073	1.551	1.629	1.383	Greater	$2.8786 * 10^{-8}$
Total Non-Rejects	4.558	2.523	4.644	1.480	Similar	$2.8095 * 10^{-1}$
SQA Non-Rejects	1.059	0.299	1.006	0.116	Less	$6.8913 * 10^{-4}$
Self Non-Rejects	0.828	1.311	0.801	1.045	Similar	$3.7128 * 10^{-1}$
Test Non-Rejects	0.854	1.182	1.458	1.063	Greater	$5.4401 * 10^{-15}$
Total Reviewers	3.459	1.161	3.940	0.894	Greater	$2.3401 * 10^{-11}$

[10] Here *after* refers to issues whose first recorded issue review occurred after 16 July 2010.
[11] Here *before* refers to those issues that were SQA reviewed before 1 January 2010

There was a significant decrease in the total number of reviews per issue. Additionally, looking at the break down of those reviews, there was a significant increase in test reviews and significant decreases in SQA reviews. The overall decrease and changes within the breakdown of issues is interesting and suggests improvement. The increase in test reviews arguably reflects better communication between development and test with additional focus on verification and understanding of changes. Similarly, the decreases in SQA reviews may reflect the voluntary withdrawal of SQA from reviews as SQA rejection and the need for review of process and other non-product concerns decreased.

Even though there was a significant decrease in the total number of reviews, there was a slight (not statistically significant) increase in non-reject reviews and decrease in the variability of this number. As with the overall reviews, the non-reject reviews breakdown mirrored the significant increase in test reviews and decrease in SQA reviews.

There was also a significant increase in the number of different reviewers per issue. This may reflect improvements in inclusion of appropriate reviewers. This is presumably more apparent in the time period after the announcement of PnitAgent but before SQA requested that they no longer be involved with review of many of the issues. The exclusion of SQA reviewers in the period after SQA began spot checking issues would presumably have served to decrease, rather than increase, the average number of reviewers.

6.1.7 Relative Timing of Rejections

It is difficult to capture the relative timing of rejects and non-rejected reviews as impacted by PnitAgent. This is in part because the hope is that PnitAgent be used by the resolver prior to the first review to avoid avoid associated rejections entirely. Similarly, the meaning of the relative order of reviews associated with an issue without any associated rejections is unclear. It is somewhat interesting, however, to look at the timing of a rejection relative to other rejections and non-rejections associated with the same issue. In particular, consideration of the relative timing within the 336 issues involving at least one rejection *before*[12] the introduction of PnitAgent in contrast with the 267 issues involving at least one rejection *after*[13] .

[12] Here *before* refers to those issues that were SQA reviewed before 1 January 2010

[13] Here *after* refers to issues whose first recorded issue review occurred after 16 July 2010.

Table 6.9: The relative timing of rejections, and other reviews associated with the same issue, from *before* and *after* introduction of PnitAgent to the S team sampled up through March 14, 2011.

	Before		**After**		**Change**	
	Average	Dev.	Average	Dev.	Direction	p-value
Passed	3.545	1.824	3.599	1.284	Similar	$3.3332 * 10^{-1}$
Rejected	0.875	1.366	0.352	0.621	Less	$2.0193 * 10^{-10}$

There was not a significant change in the average number of non-rejected reviews proceeding a rejection of the same issue. However, the average number of rejections preceding a rejection of the same issue showed significant reduction.

6.1.8 Change In Rejection Classification Data

To help examine the hypothesis that the major sources of waste identified through analysis of classified rejects were valid and could have been reduced by the support aimed at addressing them, a set of subsequent rejections were classified and the themes extracted from them validated as described in Rejection Classification (Subsection 3.6.1). A summary of this second set of classifications showing their short descriptions and frequencies is seen in Table 6.10.

Table 6.10: The classified causes of rejection for the S team sampled where first review later than September 16, 2010 and SQA review before March 14, 2011.

Cause of Rejection	**SQA**		**Self/Peer**		**Total**	
	#	%	#	%	#	%
Change During Review definition			3	2.46%	3	2.0%
Change During Review workload management			2	1.64%	2	1.3%

Continued on next page

Table 6.10 – continued from previous page

Cause of Rejection	SQA		Self/Peer		Total	
	#	%	#	%	#	%
Comments not recorded			1	0.82%	1	0.7%
Content: caught by slow check - static analysis			1	0.82%	1	0.7%
Content: caught by slow check - unresolved TODO pointing to this			2	1.64%	2	1.3%
Content: incorrect	1	3.4%	34	27.87%	35	23.2%
Content: incorrect: report	2	6.9%	2	1.64%	4	2.6%
Content: investigate			2	1.64%	2	1.3%
Content: missing/incomplete	1	3.4%	26	21.31%	27	17.9%
Content: suggested change/refactoring			1	0.82%	1	0.7%
Description: incomplete/typo/unclear	2	6.9%	1	0.82%	3	2.0%
Inconsistency: related record			2	1.64%	2	1.3%
Inconsistency: report vs PNIT	10	34.5%	3	2.46%	13	8.6%
Inconsistency: resolution vs content	1	3.4%	2	1.64%	3	2.0%
Inconsistency: resolution vs reproducibility			1	0.82%	1	0.7%
Inconsistency: version vs content	1	3.4%	5	4.10%	6	4.0%
Incorrect Version Number			2	1.64%	2	1.3%
Missing related records	1	3.4%	4	3.28%	5	3.3%
Missing Reviewers			5	4.10%	5	3.3%
Related records usage			1	0.82%	1	0.7%
Resolution incorrect			1	0.82%	1	0.7%
Resolution vague/missing details	5	17.2%	15	12.30%	20	13.2%

Continued on next page

Table 6.10 – continued from previous page

Cause of Rejection	SQA		Self/Peer		Total	
	#	%	#	%	#	%
Version information incomplete	2	6.9%			2	1.3%
Version missing or incorrect format	3	10.3%	4	3.28%	7	4.6%
Total	**29**	**100%**	**122**	**100%**	**151**	**100%**

The individual classifications were subsequently grouped into themes including the major sources of waste previously identified and described in Identifying Sources of Waste (Section 4.1). All of the original major sources of waste remain, with those primarily targeted to be addressed by this research reduced to smaller fractions of the total as shown in the Pie chart accounting for major sources of waste September 2010 through March 2011 (Figure 6.2).

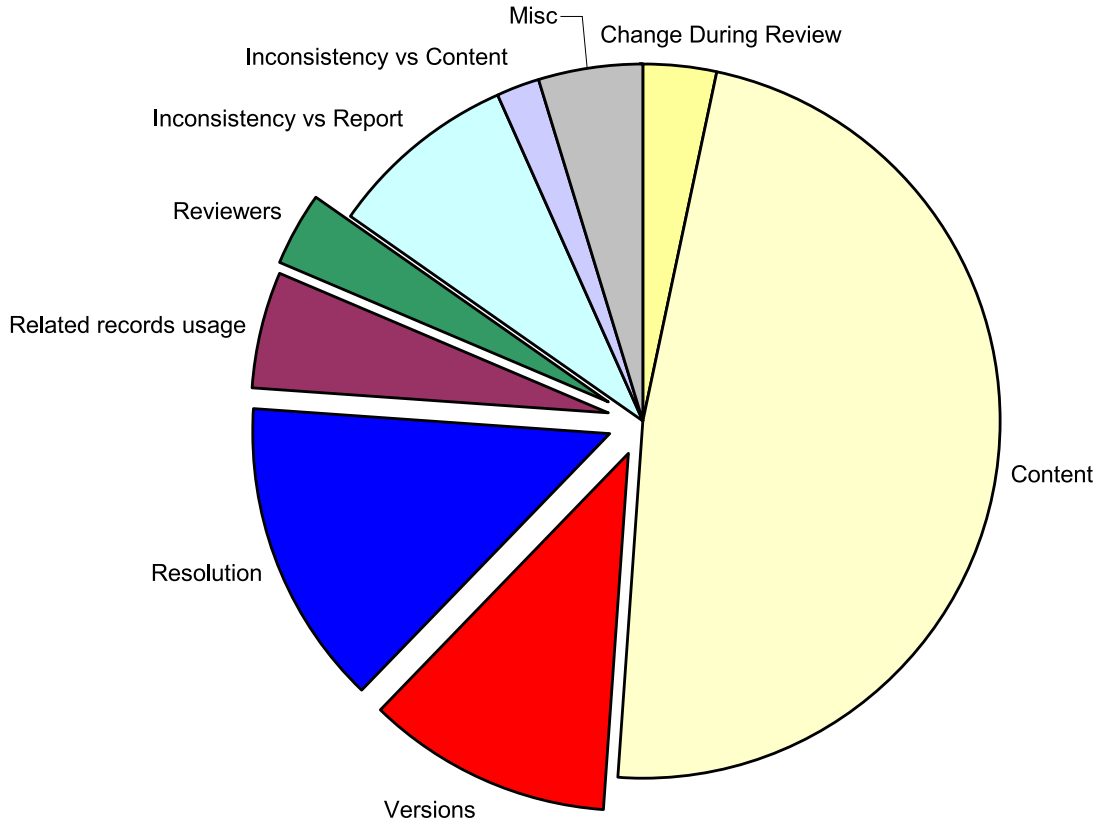

Figure 6.2: Pie chart accounting for major sources of waste September 2010 through March 2011

This can be contrasted with the Pie chart accounting for major sources of waste March 2010 as seen in Figure 6.3. Note that the four major sources of rejections targeted by this thesis shrunk significantly (both in number and as a percentage) in the sample from September 2010 through March 2011 (shown together in the bottom left side of the charts). Thus, there was not only an overall reduction in the rate of rejection, but there was a greater reduction in the areas that the agent was designed to address.

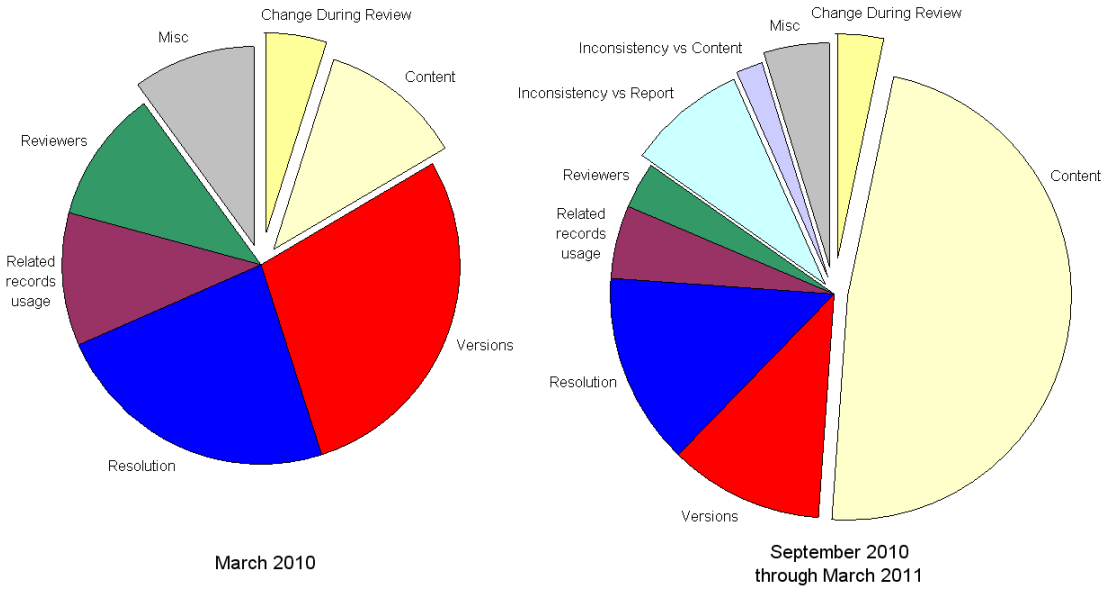

Figure 6.3: Rejections by classification before and after. Relative pie size reflects the total number of issues in the sample period.

The four major sources of waste identified from rejection classification in March 2011 accounted for about 73% of rejections with Versions (Subsection 4.1.2), Resolutions (Subsection 4.1.3), Related Records (Subsection 4.1.4), and Reviewers (Subsection 4.1.5) contributing about 28%, 23%, 11%, and 11% respectively. These were reduced in the sample taken from September 2010 to March 2011 by more than half to about 34% in total with each taking about 11%, 14%, 5%, and 3% respectively. At the same time, rejections related to Content and Change (Subsection 4.1.1) went from less than 25% to more than 50%.

6.1.9 Anecdotal Data

In addition to the objective data collected as part of this thesis, a good deal of positive unsolicited feedback was also received. For example, on April 30, 2010, a developer on another team that was exposed to PnitAgent sent an email CCing a Sr. Manager saying "I just wanted to let you know I found the PnitAgent tool you created extremely useful when doing reviews. Not only have I been able to complete them faster, but the tool

found problems I may have otherwise missed. I also found it helpful in analyzing my own issues prior to routing them for review, which should help lower my rejection rate. Nice job!" This sort of anecdotal feedback not only provides another unsolicited form of evidence supporting the value of the capabilities provided and the sources of waste they aimed to address, but helped to bring users and user interest to support this research. In response to the email, the Sr. Manager asked the PNIT Tiger Team to show her entire team the tool and encourage them to use it.

Not all feedback was positive, but the small amount of negative feedback was constructive and generally an explanation by a potential user as to why they would not be using or continuing to use the agent. For example in January 2011, one previous user explained they would not be using it anymore unless it "could work without relying on installation of the clear quest client." Similarly, the test lead for the S team asked that a version of the agent be created that could work via proxy to another copy of the agent or other software so that it could be more useful to contract test resources based in another country. Though this was not realized directly, it was ultimately allowed by having some users remotely control a computer in the United States. This feedback was also interesting, because it became clear that this test lead was personally using a copy of PnitAgent, but later review showed that their agent never successfully sent an email reporting its use. On two occasions, one infrequent user commented that they did not think the (optional) capability for it to monitor their actions should have been incorporated (though that capability was turned off by default in all but very early prototypes of the software). A more frequent user, commented on occasional pauses while the agent was monitoring and how "basic" the usage information was.

Most of the positive feedback was fairly general, but in some cases highly detailed or requesting very specific support. On June 17, 2010, a human factors engineer wrote, "I am finally getting back into ... PNIT mode ... and PNIT agent is very cool! Thanks for all the work, you are saving me a bunch of time." On a few occasions, one of the software quality assurance engineers used the agent to analyze all of the open reviews so that issues with a common problem she had noticed could be identified and communicated together.

6.2 Basic Evaluation

In order to evaluate the impact of the support provided in reducing rejection in the major sources of waste initially identified, this thesis considers together the usage, issue tracking, and other data to answer a series of detailed research questions to elucidate the nature and magnitude of this impact.

As a starting point, it is useful to ascertain whether waste associated with unneeded review and rejection was improved. This has to do with rate of rejection, the timing of rejection, the number of reviews, and so on.

As described in Basic Issue and Reviews Data (Subsection 6.1.3), there was a significant improvement in rejection rates following the introduction of the targeted support. This is an encouraging improvement when considered in the context of the relatively small change in rejection rates over the months prior as seen in High Level Rejection Graph June 2009 To March 2010 (Figure 1.4).

As described in Volume of Reviews (Subsection 6.1.6), the average number of total reviews per issue was also reduced significantly. Without a significant reduction in non-reject reviews, much of this was due to the relatively direct effects of improvements in rejection rates causing fewer rejections per issue. These gains were heavily concentrated in SQA reviews and peer reviews as there was a similar number of self reviews and an increase in the number non-reject test reviews. Additionally, the perception of significant and maintained improvement was so great that each of the software quality engineers associated with the S and P teams ultimately withdrew from reviewing all of the issues to only spot checking certain issues. This decision not only reduced the number of reviews resulting in rejection, but is also the cause of reduction in the number of SQA non-reject reviews.

Furthermore, as described in Relative Timing of Rejections (Subsection 6.1.7), there was little change in the average number of non-reject reviews prior to each rejection but a significant reduction in the number of preceding rejections. Thus, rejections occurred relatively earlier (in terms of the number of reviews preceeding them) and the frequency of repeated rejection of the same issue was lower.

Additionally, there were fewer overall reviews, lower average rejection rates, and rejections that did occur, did so earlier on average. All of these imply improvements

and reduction in waste.

It is also interesting to examine the degree that this improvement appears to relate to the major sources of waste identified and the support provided to address them. To this end, it is useful to consider the degree to which the new software support was used, possible correlation between that use and change in basic review data, and the degree to which the sampled classification of rejections changed.

As described in Usage (Subsection 6.1.4), the PnitAgent software capability was used by several users and used more heavily by several developers on the S and P teams as reported by the software agents.

As this system was used by some but not all users, as presented in Use vs Basic Review Data (Subsection 6.1.5), it is possible to divide and contrast the performance of the issue tracking system users on the S and P teams by their degree of use of the support provided in the advisor agent that was aimed at reducing the major sources of rejection. Those who used PnitAgent had rejection rates that not only improved, but appear to have generally improved at an equal or greater rate than others.

Thus, there appears to have been significant use of the software, improvements in rejection, and some generally positive relationship between the two. The question remains, however, as to how this improvement was distributed with respect to the nature of rejections and more specifically the major sources of waste identified by, and made a focus of improvement in, this research. Thus, if greater improvements had been seen in areas that were not targeted (i.e. those that would fit under the rubrics of Miscellaneous (Subsection 4.1.6) or Content and Change (Subsection 4.1.1)) then either the identification of the major sources of waste or the capabilities provided to address them were likely faulty. However, as described in Change In Rejection Classification Data (Subsection 6.1.8), the data suggests that greater reductions were seen in identified sources of waste, adding some additional validation and evidence of the positive and specific impact of this research.

Finally, it is useful to consider, at least anecdotally, some unsolicited user feedback that was provided. On one hand, these have suggested specific limitations and opportunities for future improvement of this research and the advisor agent. On the other hand, some feedback has also added further narrative around, if not support for, the case to be made for the utility of the agent software capabilities in reducing waste

and improving quality in the completion of issue tracking activities especially related to review. Together these seem to reflect a perception of value in the PnitAgent software, especially automated issue analysis capabilities, and areas for improvement in the agent's ubiquitous monitoring, complexity, and other technical such as dependence on other software.

6.3 Issue Complexity

One might hypothesize that if a greater volume or complexity of work is documented by a particular issue (the 'issue complexity'), that issue is more likely to be rejected. As one might imagine, a certain volume and complexity is required in the body of issues that document the corresponding work, but that complexity can be divided in different ways across a body of issues affecting the number of issues and the distribution of complexity across them. The choice in how to control this is distributed among the users of the issue tracking system as they decide when and how to create issues and provide feedback to other users on such concerns. With this in mind, it is interesting to consider if issues of higher complexity have higher rejection rates, and, furthermore, if choices could be made differently by users in the division of such complexity of issues to effect lower overall rejection or waste.

This possibility is somewhat orthogonal to the primary method of classifying rejections to identify sources of waste that in turn are abstracted into themes and validated to some extent by improvement after the application focused support. Rather than being focused on reducing the more specific sources of waste discussed and evaluated earlier in this chapter, this section focuses on issue complexity and the extent that it may be related to rejection and waste. This is interesting to examine as it could allow for greater improvements in issue tracking practices and understanding.

Anecdotally, such considerations are consistent with the practices of issue tracking system users. Comments made by some users suggested the importance of separating even closely related concerns into separate issues based on team, function, or relation to the originally described concern. Several users on the S and P teams developed patterns for how to create and relate new issues. For example, members of the S team would create a "parent PNIT" for each new feature and associate two additional issues,

the “design review PNIT” and the “code review PNIT.” Similarly, when a project scope change was decided during a “system design team” meeting, the decisions of the meeting would generally be documented in one issue tracking record, any requirements changes would be proposed in a related issue, any implementation change would be made under a third related issue, and any user interface design or additional concerns might be addressed under additional issues. Also, members of The PNIT Tiger Team (Section 3.5) expressed on several occasions the importance of not “putting too much in a single PNIT” and the challenges that presented for review and rejection. It was apparent that many, if not most, members of the team felt they could recognize an issue as very simple or very complex but there was less certainty and consistent understanding between the extremes. Furthermore, there was general support for greater alignment on how to separate concerns between issues, but there was no attempt to quantify complexity or arrive at specific and generalizable rules for understanding or addressing division of issues.

Means to quantify the volume and complexity of product work have previously been discussed in the literature and general practice. These include measures such as cyclomatic complexity of software code, words in a design document, and unique requirements in a specification. While these are all relevant to issues, none of them address all of the key facets of issue complexity. Not only do issues relate to all of these, but they also have other distinct aspects relevant to issue complexity such as measures of the amount of content documented in an issue itself and the amount of review a particular issue is subject to.

Prior to this thesis, there appears to have been no significant published work on quantifying issue complexity. To allow for consistent and objective measurement, this thesis proposes a specific definition of issue complexity. This measure appears to be one of many possible measures and although it was presented to The PNIT Tiger Team (Section 3.5) and made available to users in the form of warnings about “high” complexity in Automated Issue Analysis (Subsection 4.3.2) without any reported concerns, it has not been validated. It appears to be a reasonable measure and a good starting point based on somewhat limited feedback, but it may not be the simplest, most accurate, precise or optimal in other regards. To this extent, it is useful to understand the nature and components of how the proposed issue complexity measure is calculated

before reviewing data expressed in terms of it.

6.3.1 An Issue Complexity Metric

Although the details of how issue complexity is calculated in this thesis are shown below in Listing 6.1, it is perhaps even more useful to understand the more general concerns that may be relevant to this sort of measurement. The manner in which these concerns are integrated and scaled is intentional, and will be discussed to some extent in this section, but these are less of a focus of this thesis and likely of less conceptual significance.

The following are some key concerns that may be relevant to the calculation of issue complexity:

1. **Related Issues** - As described previously, Related Records (Subsection 4.1.4) can be essential to understanding the issue at hand in the greater context of the broader set of issues and the bigger picture. A simple bug may be relatively isolated with only one, or without any, explicit and detailed relationship to previous issues captured. In a more complex case, an issue may have dozens of related issues that may overlap with, and rely on, each other, and the issue at hand, in complex ways. For example, similar or overlapping concerns with sometimes even contradictory suggested observations or suggested resolutions may have been reported a number of times and from a variety of sources and the process of ensuring they are all resolved and woven into a clear, concise, consistent, complete and correct narrative is not always simple. Future work may examine more detailed means to quantify the nature and magnitude of the effect of related issues and their relationships with the issue at hand, for example by calculating their complexity, classifying and scoring their relationships and so on.

   For the purpose of complexity analysis, the related issues that have been captured in the issue tracking system and identified as Related Records of the issue at hand are counted and added to the complexity value of the issue at hand as seen on lines 6 through 9 of Listing 6.1. Thus, related records simply increase the measured complexity or the issue at hand by one each.

2. **Issue (and Resolution) Description** - In the documentation of a specific issue,

there is a description of the original concern and how it was addressed. This may involve a number of separate descriptions such as the description of the concern, instructions to reproduce, investigation, steps taken to resolve the issue and respond to review comments, and so on. In the issue tracking system configuration used by the S and P teams, the documentation of the issue is broken up across a number of fields including the (original issue) description, instructions to reproduce, resolution classification, review comments, a large free-form text Resolution Description field, and several others. Many of these are somewhat relevant to issue complexity or have some degree of overlap in their use (e.g., with review comment resolution sometimes being added as additional review comments and sometimes being addressed in the Resolution Description).

For the purpose of this thesis, only the Resolution Description field is considered such that increasing its length beyond a certain point adds to the issue complexity, but in a scaled and exponentially diminishing fashion, as seen on lines 10 through 13 of Listing 6.1. This calculation is simple and does not appear to sacrifice too much by neglecting more complex consideration of the additional fields. For example, this avoids concerns with details of the issue description and instructions to reproduce that appear to often overlap, neglect to address concerns identified during the investigation and resolution of the issue, and may reflect a complex set of symptoms or steps to reproduce a problem that turns out to be relatively simple to identify, resolve, and document.

3. **Review and Reviewers** - As described previously, Reviewers and review (see also Issue Tracking in Section 1.2) are an essential part of the regulated issue tracking process. The number and nature of reviewers and reviews are a component of the complexity of an issue. For example, simpler issues, such as duplicates tend to have fewer reviewers and low review related complexity. However, issues that involve review by a number of different parties tend to be be more complex and further complicated by an increased volume and diversity of review and rejection comments. Depending on the goal of issue complexity measurement, it may be important to consider the complexity of the review process and details documented in a particular issue even after it has been closed.

For the purpose of this research, only the number of non-closure reviewers is considered such that each unique reviewer, tracked by fields such as Reviewers Outstanding and Reviewers Completed, adds one to the issue complexity of an issue, as seen on line 14 of Listing 6.1. As closure reviewers are required, but may not yet have been assigned, consideration of them does not clearly add value in differentiating issues by complexity. For this thesis, it was thought to be somewhat undesirable for issue complexity to change simply with the completion of successful reviews and the review comments. Therefore, the nature and content of the reviews has been ignored.

4. **Related Software** - The product or other software changes associated with an issue are interesting to consider in the context of issue complexity. Generally issues that have no related software changes lack an aspect of complexity in that they do not need an understanding of such changes. Similarly, those with very extensive and significant changes are more complex in that they reflect the process and required understanding associated with the changes. There are numerous ways to measure the volume/complexity of software changes such as the number or percent of lines of code changed, the change in a complexity measure of the code (e.g., change in cyclomatic complexity), the number of items (lines, files, modules, variables, classes, or other component) affected, the number of different versions or change sets, and so on.

   For the purpose of this research, only the number of related files is considered such that each unique file from version control that had a change associated with the issue adds to the issue complexity with diminishing impact as shown on lines 15 through 18 of Listing 6.1. For the issue tracking system in question, this can be calculated quickly and without the overhead of accessing the version control system.

5. **Related Documents** - Related plans, reports, specs, design documents and other documents are a critical aspect of the work associated with the issue tracking system. The complexity of related documents could be considered in a number of ways such as based on the amount document change (e.g., number of lines or words changed) and nature of the document changes (e.g., changes in high-level

requirements often involve more complexity than changes in reports, especially late in a project).

For the purpose of this thesis, once over a certain threshold, the amount of text used in the documents affected field of the issue adds to the issue complexity with scaled and diminishing impact as shown on lines 19 through 22 of Listing 6.1. For the issue tracking system in question, this can be calculated quickly and without the overhead of accessing the document control systems, natural language processing or other more complex techniques.

6. **Related Requirements** - Related Risks, Design Inputs, System Requirements, Product Requirements, Test Designs, and other tagged items that are tracked by a requirements management tool and/or have associated coverage, traceability and other properties/concerns are relevant to issue tracking work and understanding related ripples of consequences. Adding, changing, retracing, removing, and other types of changes to such items may all have different impacts on the complexity of the issue at hand. Furthermore, the nature (type, level, source, history, maturity, connectedness, descendants, and so on) of such items could be considered or combined in a number of ways to contribute to an associated issue's complexity.

   For the purpose of this thesis, simply the amount of uniquely tagged items associated/referenced from the requirements management system adds to the issue complexity as shown on lines 23 through 26 of Listing 6.1. For the issue tracking system in question, this can be calculated quickly and without the overhead of accessing the requirement management system or other more complex techniques.

7. **History and Miscellaneous** - In addition to the factors considered in the PnitAgent implementation of an issue complexity measure, there are almost certainly others factors that may be relevant. For example, the history of an issue may be of particular relevance. If an issue has been left open for years or if it has already been rejected several times, then its complexity may be higher.

```
1  public double EstimatedComplexity
2  {
3    get
4    {
5      double result = 0;
6      if(RelatedRecords != null)
7      {
8        result += Math.Sqrt(RelatedRecords.Count);
9      }
10     if(!string.IsNullOrEmpty(Resolution))
11     {
12       result += Math.Sqrt(Math.Max(0, Resolution.Length - 100)) * 0.25;
13     }
14     result += NumberOfNonClosureReviewers;
15     if(FileRecords != null)
16     {
17       result += Math.Sqrt(Math.Sqrt(FileRecords.Count));
18     }
19     if (!string.IsNullOrEmpty(DocumentsAffected))
20     {
21       result += Math.Sqrt(Math.Max(0, DocumentsAffected.Length - 100)) * 0.25;
22     }
23     if (RelatedRequirements != null)
24     {
25       result += Math.Sqrt(Math.Sqrt(RelatedRequirements.Count));
26     }
27     return result;
28   }
29 }
```

Listing 6.1: Issue Complexity Property Excerpt

6.3.2 Issue Complexity Data and Analysis

The average rate of rejection appears to have generally increased with complexity from simple issues (with issue complexity less than 8) to complex issues (with complexity greater than or equal to 8) in the *before*, and to a lesser extent in the *after* period as seen in Average Rates of Rejection: Simple, Overall and Complex (Figure 6.6). The average complexity also appeared to increase with the number of times an issue was rejected as can seen in Average Issue Complexity vs Rejections Per Issue: Before (Figure 6.4) and, perhaps even more clearly, in the data from *after* depicted in Average Issue Complexity vs Rejections Per Issue: After (Figure 6.5).

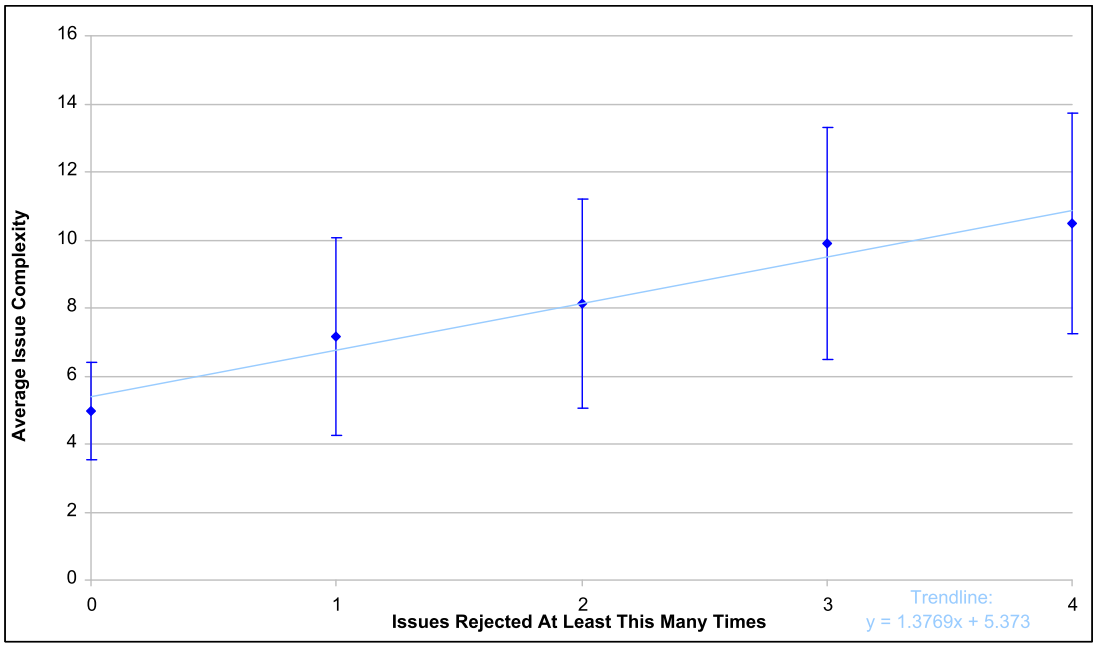

Figure 6.4: Average Issue Complexity vs Rejections Per Issue: Before

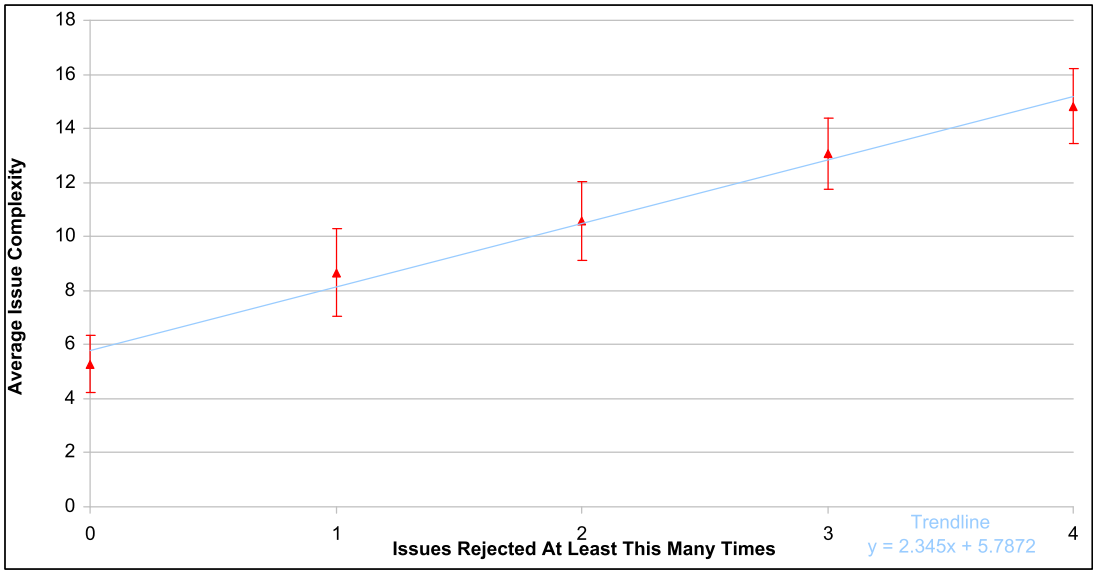

Figure 6.5: Average Issue Complexity vs Rejections Per Issue: After

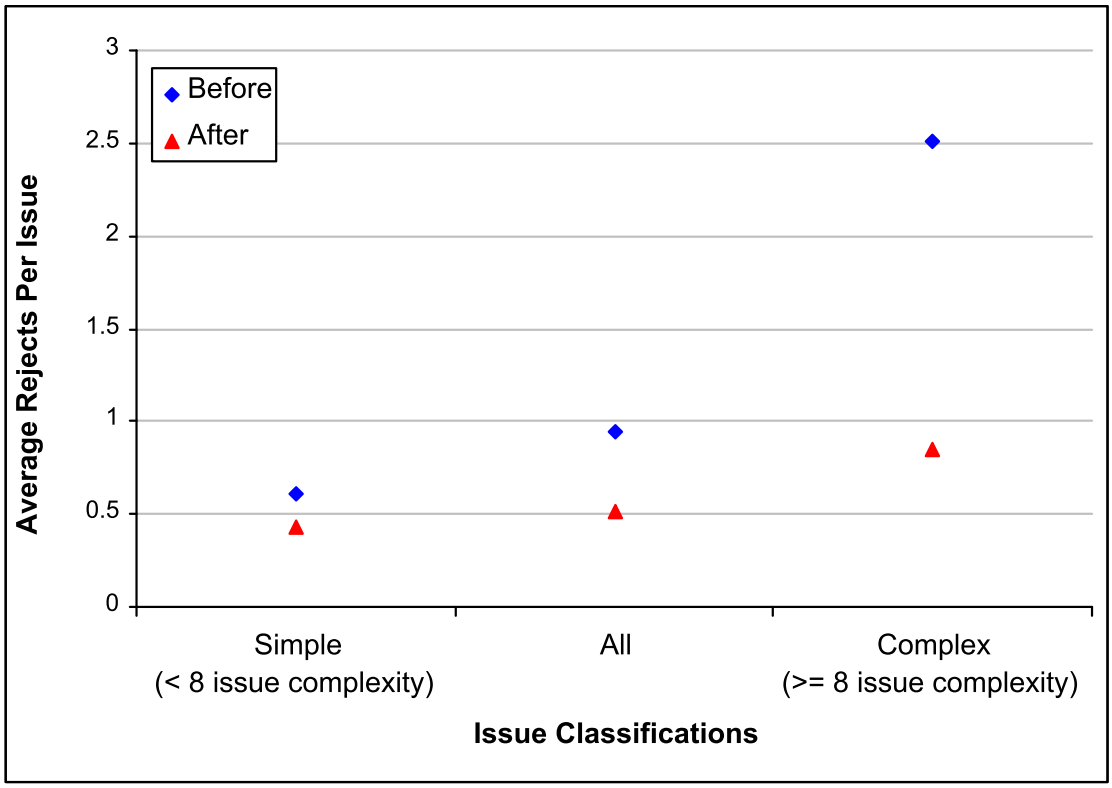

Figure 6.6: Average Rates of Rejection: Simple, Overall and Complex

This implies that there is a greater chance of rejection, and/or repeat rejection, with greater complexity both in data from *before* and in data *after*. Interestingly, as can be seen in Average Rates of Rejection: Simple, Overall and Complex (Figure 6.6), the support provided as part of this research may have helped to reduce magnitude of the negative effects of issue complexity as it translates to rejection.

The average issue complexity significantly increased (p-value of $3.1029 * 10^{-3}$) from about 6.0828 to about 6.5302 (from *before* to *after*). While warnings provided by the PnitAgent for "complex issues" may have contributed to more careful reviews, it does not appear to have prompted users to have distributed complexity across a greater number of simpler issues as the proportion of issues at, or over, the warning threshold increased from about 17.75% to about 21.08%.

With this increase in issue complexity and complex issues in mind, it appears that

change in issue complexity did not contribute to, and might even have been expected to have worked against, the improvements in rejection described earlier in this section. Indeed, it may be possible that the support provided by this research may have reduced the rate at which high complexity increases rejection rates.

Chapter 7

Standardization of Language

When considered more abstractly, this thesis explores concrete examples of problems manifest from complex, dynamic, collaborative work that were successfully addressed and measured in practice through the application of novel information technology and other means to make work associated with issue tracking more explicit and standardized. In this light, this chapter exposes this study as a uniquely tangible, measured and detailed exploration of the structuring and standardization of language to improve collaborative outcomes.

Here the term "language" is used to encompass both the literal language used in communicating through the issue tracking system and more broadly. For example, on the literal end, this work lead to agreement on the meaning of the term "Fixed" as described in Resolutions (Subsection 4.1.3) and Resolutions Support (Subsection 4.4.2). Similarly, warnings are provided by PnitAgent about words like "crash" because their meaning is vague and is often used in lieu of more structured language like 'automatically restart,' 'black screen,' 'blue screen,' or 'blue screen of death,' all of which are more meaningful descriptions. On the less literal end, this research also helped drive alignment on the major classes of rejections and materialize the review process to become so explicit that some of it became more automated.

This thesis contributes interesting detail to, and expands on, some widely cited concepts raised in related business and even software literature. In his hallmark paper, *Software processes are software too*[55], Osterweil concludes "the greatest advantage offered by process programming is that it provides a vehicle for the materialization of the

processes by which we develop and evolve software." This thesis realizes such materialization and its related advantages in the regulated issue tracking process through

1. 'process programming' to the point that parts of it are
   (a) documented in linked and detailed English text, tables and figures that are not unlike pseudo-code, or
   (b) automated and exposed as agent generated advice to users
2. abstraction of key concepts and concerns from ethnographic work and empirical data such as
   (a) Identifying Sources of Waste (Section 4.1),
   (b) the role of Issue Complexity (Section 6.3), or
   (c) the general need for in context reminders/collective knowledge such as
      i. ubiquitous annotation and usage help, or
      ii. next step sequence recommendation support.

The better part of a decade after Osterweil's paper, Monteverde presented evidence to the *Management Science* community that the success of fabless firms within the semiconductor industry relates to their ability to minimize their need to share a "form of interpersonal communication between engineers in the design and fabrication stages" referred to as *unstructured technical dialog*[69]. This work fundamentally aims to minimize such unstructured technical dialog in the issue tracking process by structuring or materializing it into more explicit knowledge (see Explicit Knowledge and Knowledge Classification (Section 7.2)).

Mahoney[70] briefly discusses common "language or coding," excerpting Williamson's explanation of the advantage of firms in forming "efficient codes"[71] and identifies accounting systems and blueprints as examples of "standardization of language." This thesis adds evidence that, in addition to more mature practices, such as those at the heart of architecture or accounting, greater standardization of language can be implemented to improve the work of regulated software development teams. In addition to more traditional face to face conversations, documentation, and group mode communication, novel information technologies (software support from advisor

agents) can be implemented and adopted in practice where their mechanism of action in producing successful results may lie in their contribution to greater standardization of language.

7.1 Knowledge Management

As described in Information and Decision Sciences (Section 2.3), knowledge management (KM) is the process of acquiring and applying knowledge to allow for more effective work. This thesis is an exercise in KM. It is focused on improving a KM (issue tracking) system in large part with the application of a novel KM system (PnitAgent) designed for this purpose.

To elucidate this, it is useful to consider more specifically where the work associated with this thesis is on the KM continuum. This can be examined in the context of "the knowledge management spectrum" Binney[72] describes from transactional to innovation/creation knowledge management including his identification of how KM applications, technologies, and other observations map to this spectrum. This spectrum breaks down in the following order and relates to the applications and technologies of this thesis as described below:

1. transactional - PnitAgent allows for knowledge to be "embedded in the application of technology" and "presented to the user of [the issue tracking] system in the course of completing ... a unit of work." This support is consistent with the expert system enabling technology.

2. analytical - This work involves the interpretation of "large amounts" of data from "disparate sources" to support the decisions of issue tracking system users to support deeper understanding of their collective work consistent with several of the identified enabling technologies including "Intelligent Agents" and "Data Analysis and Reporting Tools."

3. asset management - The work described in this thesis also involves "the management of explicit knowledge assets," for example, by allowing issue tracking users to more readily access relevant issues as well as associated artifacts and best

practice documentation and the context relevant information within these assets consistent.

4. process-based - "The codification and improvement of process" is at the heart of this work. It involves everything from definition of best practices to process mapping to workflow management (e.g., supporting email notification of other issue tracking system users when and what important work is waiting on them).

5. developmental - Especially through "Computer-based Training" within context support from an advisor agent, a key aim of this thesis has been on improving the issue tracking related competencies and capabilities of the S and P teams.

6. innovation/creation - Fundamentally, this work includes not just "training in explicit knowledge," but also facilitation of the collaborative learning process that the issue tracking system is designed to support. Ultimately, improvements in team collaboration and communication are achieved by improving the issue tracking system and process that are central to supporting them.

Though this thesis involves a balance across the entire spectrum, it is interesting to consider the breakdown and directional observations made across the spectrum. For example, the work in this thesis applies to the higher modality developmental end of the spectrum and adds several novel modalities for learning in this domain through in-context annotation, automated issue analysis and so on. Similarly, this research provides this developmental support in a manner that allows adoption and use to be purely optional, which is consistent with the observed high degree of optionality and more cultural adoption models at this end of the spectrum. Furthermore, by covering much of the spectrum, this study provides intellectual capital through both the structural capital of the organization and the human capital of increased employee competence. Finally, the work in this thesis reflects a spread of investment from explicit knowledge at the transactional end of the spectrum to "encouraging the flow of tacit knowledge" through the issue tracking system at the innovation/creation end of the spectrum.

7.2 Explicit Knowledge and Knowledge Classification

As described in related KM work (e.g. [52, 53]), much of the change driven by this thesis falls under the rubric of making more accessible knowledge and transforming it from implicit, fragmented, and often inconsistent *tacit* knowledge of individuals into more *explicit* knowledge that is encoded, communicated and applied more effectively across the organization.

Jasumuddin et al[73] noted that "there are still many gaps in our understanding of organisational knowledge and its implications for firms and managerial practices... Among the questions that still remain unresolved is one that arises from the categorisation, and subsequent utilisation, of tacit and explicit knowledge... two dominant perspectives on the relationship between tacit and explicit knowledge have emerged: knowledge as a category and knowledge as a continuum." They further summarize the relative advantages of both types (or ends of the spectrum) of knowledge (e.g., tacit knowledge is more easily lost and difficult to communicate, while explicit knowledge can be more easily imitated and is therefore less secure) and conclude that they are inextricably related and should be considered on a continuum defined by their mix. They go on to describe how this mix can be seen in communities of practice where their unique language is used effectively to make the knowledge internally explicit, but externally tacit, further suggesting that innovative culture is "inherently local" and dependent on standardization of language. They primarily advocate for a strategy where "on the one hand codifying knowledge internally, but, on the other, treating the codes themselves as personalised and tacit."

This thesis fundamentally allows for that strategy to be applied within the regulated software development community of practice. It fosters standardization of language in that group by defining, reminding, warning about, and automating aspects of collaborative work within that domain. To a great degree, it makes the knowledge captured in the work more explicit which makes it easier to understand, imitate and work with as part of the community of practice.

Perhaps in conflict with traditional business strategy though, the research associated with this thesis is aimed at making it easier to join and engage in the S and P communities of practice, not just for internal parties that collaborate through issue tracking,

but also to external auditors and others. To this extent, in terms of the dimensions of knowledge described by Spender[74], the efforts associated with this thesis intentionally move knowledge to be more objectified; that is more explicit, less individual and "essentially public knowledge." In contrast, [74] implies that a "theory based on objectified knowledge depends on effective use of the institutional mechanisms, such as patents and registered designs." In the situation of concern to this thesis, however, concerns that the knowledge is not protected by such means are instead minimized as it is unlikely that a competitor will need to collaborate on the same product or have access to the same systems and cultural codes. Furthermore, denying competitors, or at least outside agencies, the benefit of review and general presentation of the applied best practices in this area would be considered unethical or inconsistent with the company's mission and the realities of its operating environment.

In addition to the dimensions of tacit, implicit, explicit, individual, social and various transitions between these, the literature also proposes some other models and classifications of knowledge, including notably Boisot's model. Boisot [75] describes a model with three dimensions from concrete to abstract, codified to uncodified, and diffused to undiffused whose flows are used to describe the Social Learning Cycle. On some level, these are similar to other models/dimensions discussed previously[76]. For example, this learning cycle describes flows from diffuse/uncodified knowledge through abstraction and diffusion to others who absorb it and apply it concretely which might also be described as a tacit-explicit-tacit and individual-social-individual flow of externalization and internalization.

Such flows or cycles are an interesting serialization of states along the different axes of knowledge classification and, in cases like the social learning cycle, or simple knowledge management strategies, appear to represent a linear value stream progression. Previous work, however, does not appear to set out a single, simple, linear series of well described conceptual states along a knowledge spectrum that maps well to some of the details of this thesis.

In particular, it is useful to consider in detail where the support for issue tracking associated with this thesis might lie on such a sequence. In continuums with a single classification axis, such as the single tacit-explicit mixture continuum advocated in [73], there is no clear increasing value or temporal flow from one end of the spectrum to the

other and the spectrum does not seem to effectively differentiate the different aspects of the work. This is especially apparent at the more explicit end of this spectrum where investments associated with this thesis in novel information technology have been greatest. Also the learning cycle and other discovery-application flows seem to break down in some cases at the extremes or at sufficient detail. For example, where automation executes the learning cycle or brings support directly into context at which point control transfers, some steps or concepts like absorption or internalization (or the move from explicit to tacit) seem to breakdown.

7.3 Knowledge Maturity

It may be useful to think of the continuum of knowledge mono-dimensionally but more concretely and in a slightly different dimension. For this purpose, it is useful to propose a slightly different knowledge spectrum, called 'knowledge maturity,' ranging from the least to the most mature.

1. **Undefined** - Knowledge that has not even been conceived of, let alone defined or applied in any practical way, is perhaps the least accessible to a firm.

   Although it is far from explicit, it does not make sense to say it is the most tacit because it is not in the minds of individuals, or that it is the most implicit because it cannot be used automatically or collectively. Although it is certainly not codified, it is not any less codified than knowledge that is Unknown. Similarly it is neither abstract nor concrete, diffused nor undiffused, individual nor social.

2. **Unknown** - Knowledge that has been described but not discovered. For example, the answer to a question such as 'what material makes a good electric light bulb filament?' was unknown before years of Edison's empirical experiments, but the question itself reflects some definition that makes it possible for it to be measured and searched for methodically by any number of people who understand and can act to explore the question. Such information can be sought over time by those with prepared minds, but it cannot be immediately acquired from anyone at any monetary price.

3. **Individual** - Knowledge that is locked up in the minds of people. Knowledge that can only be applied by and may conflict between individuals, and which is not transferred to a broader group or firm. This appears to be what is typically considered the extreme of tacit knowledge in the continuum from tacit to explicit knowledge described in the knowledge management literature. This knowledge may easily be lost with or forgotten by the individuals that possess it, and may be Unknown or even Undefined to others and even at the conscious level to those who possess it.

4. **Tribal** - Knowledge that is locked in the minds of individuals, but some part of which is implicitly passed on or re-learned through experience in the area or practice. If the knowledge is no longer passed on for whatever reason, it may become individual knowledge. If the entire *tribe* with the relevant knowledge is lost or does not maintain the knowledge it may become Unknown or Undefined.

5. **Taught** - Knowledge that is intentionally and methodically taught directly by existing members to new members of a community of practice.

6. **Encoded** - Knowledge that is encoded into documents, processes, tools, practices, or other means that can be experienced and used by others without direct personal transfer of knowledge. This form of knowledge is less subject to loss and unintended evolution.

7. **Accessible** - Knowledge that is defined centrally, made intuitive, and/or encapsulated sufficiently in accessible impersonal media that it can be consistently interpreted and applied by many with a variety of past experience. This appears to be what is often considered the extreme of tacit knowledge in the continuum from tacit to explicit knowledge described in the knowledge management literature.

8. **In-Context** - Knowledge that is not only accessible, but readily accessible, tailored to, and proactively presented in the specific and detailed context where it is applied. For example, networked location aware mobile devices and supporting systems (like global positioning satellites) allow geo-spatial knowledge to be so accessible in their current locations that one might say it is *in context*.

9. **Automated** - Knowledge that is successfully and automatically applied without the need for supervision or understanding of the one who initiated it. For example, if a person needed to navigate to the nearest gas station they might be able to naturally communicate that goal to a smart car that would initiate an automated process for determining their context (e.g., location, nearest gas station, seat belt status, speed limits, and so on), determining actions to take to achieve the goal (e.g., planning a route), and taking those actions, all without the user needing to understand the knowledge associated with forming and executing that plan successfully.

10. **Inherent** - Knowledge that is 'built in' to such an extent that no specific goal formulation, communication/initiation, or user understanding of that knowledge is required. For example, for most people, the perception of pain and temperature from their body and determination if these are beyond a reasonable range is inherent.

This is certainly related in many regards to concerns in previous work including previously discussed knowledge axes or flows, but with a few key differences.

1. It simply defines a complete maturity scale for knowledge.

2. It concisely covers some interesting and challenging extremes that were seen in the work associated with this thesis.

3. Finally, it differentiates successfully, but simply, some different areas for support described by this thesis.

Within this framework it is interesting to consider development work and firms, or even sub-teams within such firms, that do innovative work within the medical device industry. Firms have a fundamental interest in developing the knowledge that underlies their products. More specifically, they should be interested in moving such knowledge down this spectrum from undefined towards inherent. Some part of the firm must identify what needs exist for a medical device, defining and bounding the problem(s) to be solved. While the content and details of the device and its supporting materials, such as the evidence of its development and performance, remain unknown, definition

of their general nature or some of the questions that must be to achieve them is the requisite first step towards along the spectrum. One or more of their employees must then at least sub-consciously recognize or otherwise form in their individual mind(s) the internal understanding required to pursue achieving the desired nature or answering such questions. To sustainably achieve anything but the simplest devices, the firm must realize the benefits of (at least internal) collaboration and reach beyond the limitations of isolated individual minds. Such transfer of knowledge may begin with observation and implicit education. For products and supporting activities to be accepted by regulatory agencies, for intellectual property protection, and by other stakeholders, they must be methodically and intentionally explained and encoded in persistent and accessible language. If topics of interest in such documentation are not sufficiently accessible to them, regulatory agencies and other stakeholders will require them to be further refined, explained and exposed to them before they is broadly accepted. To achieve greater efficiency and effectiveness that knowledge can be made available in the context where it is needed, can be leveraged to some extent automatically without requiring as much human supervision, or ultimately inherently achieved appropriately without any opportunity for error.

More narrowly, issue tracking involves similar progression down this spectrum and PnitAgent, as well as this work more broadly, facilitates this progression with measured positive impact in practice. It appears more precisely that this work appears to have contributed to the standardization of language, which in turn accelerated this progression and improvement. At a very high level of abstraction, for example, work associated with this thesis fundamentally:

1. identified knowledge that was *unknown* (i.e. 'What significantly contributes to issue rejection in this domain and what might be done to address it?');

2. broke it down into smaller pieces through measurements, discussion and alignment on a new language of classifications, best practices and descriptions (e.g. Versions (Subsection 4.1.2));

3. and then took steps to advance knowledge in those sections (e.g., through training (see Section 4.5) and agent-based automation and reinforcement such as Versions Support (Subsection 4.4.1))

7.4 Resolution Classification

By setting or changing the Resolution field of an issue, the user selects a classification of the resolution of the issue at hand. There was little alignment on detailed definitions of the different resolution classifications and the process for selecting appropriate values for this field before this research. Related challenges and rejections were often believed to reflect poor understanding of the relevant classifications and the terms used to describe them. This lack of detailed alignment lead to some confusion and change as issues went through the review process and different users looked at them. An example of this was discussed at the beginning of this chapter and earlier in Resolutions (Subsection 4.1.3) and Resolutions Support (Subsection 4.4.2) relating to the term "Fixed" which is one of the options which may be selected in this field.

Using audit trails provided by the issue tracking system, it is possible to look at how many times this field changed in an issue after it was first reviewed by another user. Looking at the average rate of these changes in the *before* period ($\approx$0.076) and *after* ($\approx$0.019) period reveals that there were significantly fewer average changes of this sort afterwards (p-value $8.961 * 10^{-4}$). Furthermore, within the *after* period, there was a significantly (p-value $3.022 * 10^{-2}$) lower average rate of change for regular users ($\approx$0.008) compared to other users ($\approx$0.030). These differences appear to reflect the effect of the efforts associated with this thesis in standardizing language in this area both overall and to an even greater extent within those using the agent.

Improvements in the rates of change to issue resolution classification after review reflect a concrete example of standardization of language and the effects of this research in intentionally driving greater knowledge maturity. The PNIT Tiger Team (Section 3.5) held a series of meetings that specifically discussed and brought the leadership of the S and P teams into alignment, correcting *individual* knowledge, and establishing common *tribal* knowledge, at a detailed level about each of the relevant classification options/terms. The relevant knowledge was eventually *taught* by that group to others and *encoded* in detailed documentation (available from a website and document control system) as well as Usage Help (see Annotation and Sequential Support (Subsection 4.3.1)) that made the knowledge *accessible* and available *in-context*. Finally, support for detecting, warning about, and explaining some incorrect use of such resolution classifications

was *automated*; see Resolutions Support (Subsection 4.4.2).

7.5 Annotation of Remaining Work

Knowledge of work that was once recognized to be remaining, inadequate, or otherwise in need of change is an important part of development in this domain and in general. This need is one important motivation for the issue tracking system itself, but in many cases is not efficiently handled by current issue tracking systems. For example, issue tracking systems *encode* and make more *accessible* such concerns, but they are not good at bringing it further down the knowledge maturity spectrum.

To address this residual need, other techniques are often applied. One such technique is the use of annotations or comments such as "TODO: this needs to be fixed" inserted into the code or documentation itself. These have the advantage of being immediately *in the context* they apply to rather than in a separate issue tracking database with some description of how to find the place where they apply. Unfortunately, such approaches have other limitations such as producing duplication with issues tracked in the issue tracking system, being in many ways less accessible because they require the relevant code context to be seen or at least searched. Existing tools (such as the "Task List" in Microsoft Visual Studio and analogous support in Eclipse and other integrated development environments) already provide some support for leveraging such comments, but are brittle (e.g., recognizing "TODO" but not "TBD") and limited.

Support was built into PnitAgent with the aim of helping to address these limitations and the residual need. This includes support for flexibly recognizing, filtering, and integrating such comments from various types of software code, Microsoft Word documents, fields within issues themselves, and other sources. This also includes presenting the most relevant of these *in-context* and refined by *automated* analysis to provide support while completing certain issue tracking activities. For example, drawing greater attention to comments referencing the issue in question or warning about "TODO" type references to issues that are closed or in review (while filtering out references from review reports that are not immediately important to the artifacts and issues in their current state). Some time after this support was provided by the agent, the concerns raised by it were discussed and eventually there were some related changes *encoded*

in coding standards and discussed with other leaders from software development and other functions. Thus, although it was not previously discussed, documented, or taught beforehand, it was implemented in the agent and as a result, the relevant knowledge recognized, matured, refined, and shared.

In addition to behavioral changes seen on the S and P teams and more broadly, the agent's support lead directly to the detection of several related concerns and rejections. Though these presumably increased the number and relative rate of rejections, they resulted in earlier detection and prevented the creation of additional issues. Additionally, a couple of users have also reported that this support prevented them from sending issues to review prematurely.

7.6 Inclusion of Appropriate Reviewers

A major source of waste found in the initial rejection classification was related to inclusion and understanding of appropriate issue reviewers. This is likely because there was not a common understanding of the details of review roles and who should fill them. As described in Reviewers Support (Subsection 4.4.4) this research produced the first documentation of detailed issue scenarios and the different reviewers that were expected for each of them. The PNIT Tiger Team (Section 3.5) aligned on these rules and PnitAgent provided automated support to recognize and warn about when they were violated providing links to more detailed descriptions. There was significant improvement in this category from 11% overall before to 3.3% after, as well as in SQA rejections from 13% to 0% as described in Change In Rejection Classification Data (Subsection 6.1.8). It seems plausible that the improvement in understanding and involving the appropriate reviewers is an example of standardization of language facilitated by the efforts associated with this thesis.

7.7 Broader Standardization

In addition to specific examples of standardization of language such as that related to Resolution Classification (Section 7.4), it is useful to consider measures of this effect more broadly. One such measure would be to examine the cohesiveness or similarity of

the user entered text that makes up much of the content of the issues.

One might expect that measuring the similarity between the contents of the issues would increase with greater standardization of language. A simple way to measure the similarity of two short documents is to look at how much overlap they have in their terms. To "significantly improve matching coverage" [77] between the relatively short text content of issues and because all entries are in English, stemming was used in this analysis with an implementation derived from Porter's Stemming Algorithm [78, 79]. In this way we looked at the overlap of stems of terms rather than exact terms. Further ideas for implementation details such as "noise word" removal where used based on [80]. This was calculated on the Description, Resolution Description and other free form text fields of all unique pairs of issues and then averaged to determine the *stem overlap* in a group of issues. These fields are often not written or changed by the same or a single person and because author determination is difficult, it is also difficult to compare regular and non-regular users in this area. However, the stem overlap in the period *before* the influence of this research and in the period *after* were both calculated, with average stem overlaps of approximately 0.1255 and 0.1663 respectively. This showed significantly (p-value $\ll 0.1$) greater average stem overlap in the *after* period, consistent with the notion of standardization of language. This seems consistent with the idea that greater similarity or consistency is achieved as users better understand what they are expected to write to satisfy the agent software and other users.

Although it is difficult to compare regular and other users with respect to average similarity in the combination of the various free form text fields of issues, it is relatively straight forward to compare them with respect to average similarity in the Resolution Description field only. Interestingly, although regular users showed an increase in average similarity or resolution description from *before* to *after* (p-value $3.1699 * 10^{-6}$), they showed slightly, but significantly, less similarity than other users (p-value $3.1699 * 10^{-6}$). With free form text, as in this case, the significant effort put into the natural language processing techniques employed by the agent allow for equally successful support despite substantial flexibility. It seems plausible that despite helping to improve their relative rejection rates, this did not constrain the natural language of regular users to the extent that they had significantly greater similarity in their resolution descriptions than other users. Furthermore, non-regular users use of unclear, incomplete, or vague language

that would be detected by the agent likely have this corrected through rejection after agent-assisted review. It is also possible that regular users are actually encoding their resolution descriptions more efficiently than other users and as a result have relatively less unneeded, but similar, content in their descriptions.

Along these lines, a potential measure of broad standardization of language is issue content encoding efficiency. One might imagine that standardization of language in this domain would take the form of communication efficiency optimization resulting in more "efficient codes[71]." Furthermore, this efficiency might manifest shorter descriptions leveraging more precise language. One might expect this to be reflected in shorter resolution descriptions in the *after* period and to an even greater extent in regular users of the agent software. The average overall length of resolution description *after* ($\approx$400.6 characters) was significantly (p-value $4.324 * 10^{-3}$) less than that *before* ($\approx$634.6 characters). This may reflect an overall improvement in efficiency in encoding of the resolution description.

Interestingly, however, average resolution description length with resolvers who were regular users of the agent ($\approx$489.1) was significantly (p-value $1.813 * 10^{-2}$) greater than that of other users ($\approx$308.3). These findings are also surprising when considering that rejections decreased but that increased volume of description would have presumably allowed for greater chance for error or contradiction to be found in review resulting in rejection.

It seems plausible that the descriptions have so much additional valuable information that they are longer despite more efficient encoding of that information. Perhaps this reflects standardization of language in the very inclusion of important and clarifying information that in turn helps reduce the rate of rejection. Despite the increased potential for error or contradiction with the larger volume of information presented, this explanation also seems consistent with the notion that the support provided in this research may have reduced the effects of Issue Complexity (Section 6.3) on rejection. Thus, perhaps error and complexity tolerance along with consistent understanding or at least acceptance of encoded knowledge are all additional measures of standardization of language. Thus, the improved rejection rates and other findings, described in earlier presentation of Data, Analysis and Evaluation of Impact (Chapter 6), also reflect broad standardization of language.

Another potential measure of standardization of language is subjective and anecdotal feedback. Though it was never presented as such to them, the language standardizing effect of this work has not escaped those involved with the S and P team. Many PnitAgent users and others aware of this work have commented on its role in driving alignment and consistency. For example, referring to the PnitAgent software, the software quality assurance engineer responsible for the P team said, "This is good work you've done. Ninety-five percent of the time when we now say blue we mean blue. Not light blue or dark blue, but blue ... we are talking the same language now."

While these statements generally are not phrased in the same terms used in this and other academic work on knowledge management, process programming or standardization of language, it seems apparent that their relevance and intent are clear. Indeed, they seem all the more meaningful because they are unsolicited, and reflect unfamiliarity with this area of research. It is possible that in addition to standardizing language specifically around issue tracking, work associated with this thesis has helped structure by example the more abstract process and concept of this standardization in the minds of those exposed to it. With this perspective in mind, it seems possible that this thesis might be helpful in supporting standardization of language in completely different applications and domains as well.

Chapter 8

Conclusions and Discussion

This thesis analyzes waste in the form of issue tracking rejections during the development of medical device software. Ethnographic and empirical data relating to the reasons for rejection were collected. Themes were abstracted from this data revealing major sources of waste, most of which were not directly related to the content of the product artifacts. These themes were validated by an experienced software quality assurance engineer and other members of a team of multi-disciplinary experts familiar with these issues. Steps, including novel software capabilities built into advisor agent software, were proposed and implemented to address the major sources of waste identified. Later, a second set of rejections were collected, classified and validated.

Analysis of classified rejections, as well as data collected by the issue tracking system and agent software from *before* and *after*, revealed that there were significant improvements in rate and nature of rejection. Furthermore, greater improvements were seen in groups that were observed to use the advisor agent software. These results provide evidence to validate the major themes of waste identified as well as the steps taken to address them.

8.1 Generalization and Abstraction

This thesis may be of greater interest to the extent that it can be considered more abstractly to apply to broader topics relevant to communities such as those involved with information and decision sciences in business as well as software engineering and

artificial intelligence. The following subsections describe aspects of this thesis in the context of more general themes relevant to some of the current discourse in these areas.

8.1.1 Issue Tracking

Issue tracking as examined by this thesis is a collaborative engineering practice involving the distributed, user-driven documentation, review and collaborative judgement of various concerns. This is applied in various cases to specific content in the form of potential defects, enhancements, and other concerns relating to specific requirements, source materials, design considerations and other materials. More abstractly, though, it is a process for a small community to methodically create and align on a narrative of these concerns. In the activities required to support each issue, it adds value by incrementally defining, aligning and refining this narrative. If the initial resolution of each issue was reliably good enough, only the defining step of this process would be needed. The refining and aligning work done in the review of each issue is overhead. The generation of rejections in the peer review process at the heart of the issue tracking review step reflects the need for this overhead and defines a measure of waste in the process.

This thesis examines the causes of these rejections through abstracting classification. It then proposes broader causal themes in this data and invents means to partially address them. These means were implemented and some measures of their use and impact were measured. These measures provide evidence that rejections were reduced and that to a rejections associated with the targeted causes and use of the implemented means to address them were reduced to a greater extent. This provides evidence not only of the success of the implemented rejection reducing means, but also a degree of validation of the causal themes and the method used to find and address them. Anecdotal feedback further suggested that the changes made helped to eliminate waste, not simply shift it elsewhere.

This thesis supports the refinement and alignment of the incremental narrative of development teams in issue content that is managed through peer review feedback cycles that are part of the issue tracking process. On this level, parallels with this research appear to exist much more broadly. Conference proceedings, or even dissertations and more generally a technical community's literature, for example, might be considered to

be an incremental narrative of that community. With papers, rather than issues, there is generally a similar peer review, feedback and acceptance process that helps to refine and brings papers into alignment with the community. The causes of paper rejection (or change) might be classified, abstracted into themes, means to address those themes proposed and implemented, and that evidence of improvements might be seen. Even more specifically, it might be interesting to implement an advisor agent that supports paper writing and reviewing activities (e.g., not just checking spelling and grammar but with warnings about common causes of paper rejection/change including vague phrases like 'this work', long sentences, formatting, references, and more complex concerns). Similarly, one might imagine that, analogous sources of waste may apply to papers:

1. Content and Change (Subsection 4.1.1) - There may be flaws in the methods and means (content) associated with a paper. Change in the technical community may similarly obviate, diminish or alter the perception of a paper's subject or conclusions, which, for example, may no longer be as novel given recent work.

2. Versions (Subsection 4.1.2) - The technical details of a paper's reference to its associated work may be incomplete, inconsistent, or incorrect. For example, a paper may not say how many participants were involved, when the work took place, which sub-population it is discussing, and so on.

3. Resolutions (Subsection 4.1.3) - A paper may have missing, vague, and incorrect information or it might be misclassified (e.g., submitted to the wrong track/conference/community or described inconsistently).

4. Related Records (Subsection 4.1.4) - A paper may not cite, or incorrectly cite, related papers. The relationship to referenced work may not be clear.

5. Reviewers (Subsection 4.1.5) - Some communities may struggle to objectively or consistently review papers. Papers may not be distributed to the most appropriate reviewers. A paper may rely on other work that has not been sufficiently exposed for review.

8.1.2 Standardization of Language

This thesis describes a real-world, but relatively well controlled, experience with the application of enhanced knowledge management (KM) support in identifying areas for improvement, further deploying support in these areas, and measuring the use and impact of that support. It proposes an abstract, linear scale of knowledge maturity to help describe the status and improvement of these areas and the role of KM technologies and their further development in supporting this progression. It further elucidates a narrative and set of quantified benefits from standardization of language with concrete empirical and anecdotal support.

As described in Standardization of Language (Chapter 7) and previously in this thesis, software supported means of extracting and classifying rejections illuminates areas where challenges in anticipating and aligning on appropriate unstructured technical communication and documentation lead to waste, inefficiency and confusion. By organizing, naming, discussing, and agreeing on norms for improving these areas, best practices and detailed descriptions were encoded and shared electronically as well as though group mode presentation. Finally, they were partially automated into additional KM software support in the form of PnitAgent capabilities that were widely adopted and benefitted from despite controlled evolution to limit the complexity of impact evaluation. In addition to empirical data reflecting improvement and greater consistency, positive comments about the value and standardizing role of this process and software support were provided by users as well as non-users.

It seems feasible that the potential for similar means to identify, steps to mature, and techniques for measuring areas of immature and inconsistent technical knowledge and communication exist more broadly in other firms and domains. The agent has recently begun to be further extended into related areas such as design history document approval (where plans, reports, specifications, and other documents are tracked and approved) and functional test execution (where procedures for testing product and their results are tracked, executed, and so on). These and other areas appear to be ripe with opportunities for the standardization of technical language and maturation of associated knowledge using some similar techniques.

8.1.3 Innovative Application of Artificial Intelligence

The advisor agent software developed to support this thesis grew out of more general work on layering advanced UI functionality (supporting next step recommendation and flexible annotation with context demonstration) over an existing virtual environment leveraging UI monitoring along with Markov and non-Markov user modeling as published in [61]. Although this early agent work aligned well with mainstream AI with its probabilistic machine learning and agent-based architecture, it was too cumbersome to apply effectively to many of the detailed challenges presented by the issue tracking activities of the S and P teams. To address this, the agent software encorporated a variety of techniques to allow it to dynamically couple with its environment. It used clues from ongoing general UI and virtual environment monitoring and/or free form textual user inputs to identify an appropriate context in which to gather more specific inputs, perform more specific processing, and provide more specific outputs. Instead of more abstract natural language processing techniques like full grammatical analysis, it leveraged virtual context information (such as the structure of issues, artifacts, and its own interface to users) and a series of more specific pattern matching and basic language processing techniques to extract relevant information. This allowed it to be successful despite the variety and challenging nature of some of its inputs (e.g., with context being demonstrated graphically, provided by the user as a number, or extracted from a Microsoft Word document or a source code file written in one of several different languages).

Despite its complexity and mixture of traditional AI and more application specific techniques, part of this research has been accepted as an award winning innovative application in artificial intelligence (AI) by the Association for Advancement of Artificial Intelligence presented at AAAI-12 [66]. While these techniques have been applied in a relatively specific domain, this recognition aligns with our understanding that they appear to be applicable more broadly. Within Medtronic, the advisor agent software has been the starting point for a variety of work on other features and entirely new applications. These include:

1. **TrajectoryAgent** supporting the interrelated tracking, planning and reporting of tight and dynamic trajectories that equipment, participants, facilitators and

other resources take through time and various tasks primarily during validation activities such as the summative usability study of the S and P teams that spanned dozens of people, hundreds of pieces of equipment and months of time.

2. **TestAgent** supporting the efficient distributed execution and data management and analysis of automated testing, for example, for the formal test integrations and dry runs of the S and P teams where several additional machine weeks of after hours and weekend test execution were leveraged from systems used for other purposes during the day. This system relied on autonomous agent coordination, suffered from less environmental failures than formal centralized infrastructure, and provided the first automated debugging classification and context extraction for failed tests, which it performs automatically as test completion finishes.

3. **ApplicationExplorer** (also called AppWalker) supporting autonomous (unscripted) exploration (primarily for "free form" stress testing and screen capture) of user interfaces requiring no prior knowledge. In the first 2 days of operation the ApplicationExplorer software identified several previously unknown crashes in the "stable" software being developed by the P team and was able to use that software to configure printers and system time in a manner unanticipated by its users. Starting in September 2012, a contract developer was funded and began working full-time to re-implement and extend ApplicationExplorer.

The work associated with, and subsequently derived from, this thesis is an example of successful design through the agent lens to achieve flexibility in integrating information under different environments from multiple sources and formats, leveraging pragmatic approaches to context detection and language processing and development of a framework to support generalization to other application. One key area of ongoing work within Medtronic is the extension of the advisor agent software to incrementally tackle the broader problem of tool integration and process automation such as deliverable creation, tracking, and approval support.

8.1.4 Application of Automated Software Engineering

Issue tracking and the related activities are a part of software engineering. This research provided means for at least partially automating the measurement and analysis of these

software engineering tasks in some more generalizable as described in [65]. In effect, this thesis makes visible specific examples of broader software engineering concerns relating to:

1. current development/issue debt,
2. reflections of project maturity and change,
3. opportunities for automation at issue review time, and
4. beyond-version-control change sets with automated integration across multiple software engineering systems.

It not only demonstrates issue review and niche defect detection in product and non-product artifacts, but more generally illustrates the automation of novel static analysis, especially at the time of issue review such as related to remaining work in source code as well as design and other documentation. It also embodies some forms of improved collaboration automation in software engineering tasks such as communicating requests and priorities in review processes.

8.2 Future Work

Extension of the agent to provide additional support and capabilities related to issue tracking and other challenges faced in this domain are perhaps the most obvious and tangible areas of future work. PnitAgent was intentionally frozen during the *after* period while the impact of this work was being analyzed, but it has since evolved extensively and become the basis of and inspiration for interesting derivative work (including that described in Innovative Application of Artificial Intelligence (Subsection 8.1.3)). Within the narrow domain of this work, there are also some interesting questions left unanswered. For example, those related to issue complexity, its most appropriate measurement, and the means to better manage this complexity. Furthermore, although significant improvements have been seen in the rate and nature of rejection, the remaining sources of rejection and other dimensions of waste (such as the more careful measurement of time and effort associated with issue tracking) represent opportunity for future work. Adaptation and evaluation of this research to the issue tracking of other groups and domains is also interesting.

There is additional opportunity to improve the agent's user interface, deploy it in other groups, and refine its ability to detect problems in issues. Although some effort has been spent there, most of the recent work has been in other areas. Some of this more recent work has spread into more distantly related areas, such as the daily work of document, deliverable, and design history file analysis conducted extensively by quality and reliability engineers and verification infrastructure at the heart of software verification activities. As one interesting example, a spin-off project is currently funded (independent of any projects) to further develop the agent's early 'AppWalker' capability to autonomously explore software interfaces (described further in Innovative Application of Artificial Intelligence (Subsection 8.1.3)).

More generally, some of the interesting ideas applied and explored by this this thesis merit further consideration. For example, the agent's dynamic blend of coupling to users and underlying systems is interesting and more broadly applicable. This ranges from robust and abstract with minimal coupling to extensive and capable, but highly detailed and brittle. As another example, the agent's support for data integration and pragmatic free form language understanding through application of a series of specific techniques are effective and worth broader consideration. Similarly, as an example of directed, technology-supported standardization of language aimed at increasing knowledge maturity in a complex, variable, collaborative area of practice, this thesis provides an interesting example in a sea of potential. It seems apparent that this opportunity lies in at least two major veins. First, the application of similar techniques to provide detailed support that is personalized to the situation and user enabling them to work more effectively in their community of practice. Second, in providing additional research and evidence to demonstrate and reveal the specific relationship between such techniques and the more abstract standardization of language and maturation of knowledge.

8.3 Detailed Summary of Innovations

This thesis involves the development of local theories about the sources of issue rejection, targeted support to address those, and evaluation of the impact of the support provided. Significant improvement is demonstrated and even greater improvement is linked to the use of the advisor agent software deployed as part of this research.

This section attempts to enumerate the various detailed innovations, potential contributions and new ideas that are part of this research.

1. Identification, execution, and successful results of a process for reducing waste in issue tracking. This includes specifically performing rejection classification, abstracting those classifications into sources of waste, implementing software capabilities and other steps to address those sources of waste, and subsequently repeating at least part of this process and comparing rejection classification and other information across cycles to assess the impact (i.e. on quantity and quality of rejection) of changes made to address sources of waste and gain greater understanding. See Central Strategy (Section 3.1).

    (a) Exposure of rejections that are not related to product content as a key measure of waste in this domain

    (b) Identification of several key sources of waste (and validation thereof through expert review, ethnography, and associated improvement and use of support means aimed to address them)

    (c) Proposal, documentation, and exploration of specific means to address the major sources of waste identified

2. Novel software support

    (a) Exploration of agent-based meta-software support; see Annotation and Sequential Support (Subsection 4.3.1)

        i. Proof of concept design of a meta-software agent for issue tracking systems

        ii. The novel suggestion that widget and window in focus are the two key features for user interface grounded meta-context observation and user support

        iii. Identification and exploration of ubiquitous annotation support and next context prediction as two key modalities of dynamic context sensitive support that can be provided with minimal coupling (based only on widget and window in focus)

iv. A system implementation with

A. ubiquitous support for monitoring and transitioning between active elements in Web[1], application, file system, Eclipse, and other software;

B. constant user observation and support with user acceptable performance

C. a sophisticated multi-modal agent UI for direct two-way user interaction, notification of context relevant information, and configuration and interaction with model and data

D. ubiquitous, virtual, context-sensitive annotations using context descriptors that enable polymorphic context matching

E. a method for ubiquitously predicting/recommending and presenting what virtual context a user will/should proceed to next

(b) Initial conception, design, implementation, deployment, use of automated issue-focused analysis

i. Issue reference/annotation analysis in code, design documents, and other artifacts

A. Identification and presentation of references to remaining work within an issue;

B. Identification and presentation of references to the current issue from other artifacts;

C. Identification and presentation of references to work remaining related to (e.g., in files referenced by) issues that are in a post-resolving state;

D. Identification of references to work remaining which is not inside of and does not directly reference a tracking issue;

E. Identification and presentation of references to nonexistent items across data sources;

F. Integration of data and presentation including issue contents retrieved from the tracking system, software sources, documentation

[1] Web support was prototyped and tested but not deployed

in spreadsheets and other formats, associated requirements retrieved from a requirement management system.

ii. A novel approach to static analysis of source code at the time of issue review, that includes integration of data about the issue, process and related artifacts;

(c) Enhanced role analysis and support for collaboration;

(d) Support for flexible, cross-data-source, batch query and automated analysis;

(e) Support for broad issue-focused change sets that works across system boundaries;

(f) Hybrid support; see Hybrid Interactive Support (Subsection 4.3.3)

3. Refined and more explicit understanding of issue tracking in this domain

(a) Initial comprehensive description of *the story* told by issues and its key attributes

(b) Initial comprehensive articulation of how domain constraints can lend themselves to the use of an advisor agent for support and exploration of a successful adoption of supplemental functionality in such a form.

(c) Empirical evidence that major sources of rejection in issue tracking are largely not related to the content of product artifacts and documentation, but the information captured in the issues themselves. See also Item 1a.

(d) Issue complexity

i. Definition of and implementation of automated means to measure

ii. Exploration of relationship with issue rejection and repeat rejection

iii. Exploration of change relationship with issue rejection and repeat rejection with improved issue tracking execution and support

4. Exploration of the structuring and standardization of language to improve collaborative outcomes

(a) Concrete and detailed relation of relevant concepts/language from Business and Management communities to those of the Software Engineering community.

(b) Detailed examination of the work associated with this thesis in the context of knowledge management (KM) and the KM spectrum

(c) Detailed examination of the work associated with this thesis in the context of tacit and explicit knowledge and the knowledge spectrum

   i. Exposure of the work associated with this thesis as an example of where some primary security/inimitability advantages of tacit knowledge do not seem to apply

   ii. Proposal and discussion of an alternate knowledge spectrum

      A. With broader range and more concrete breakdown than typically considered in the literature

      B. Consideration in the context of this work and the various software information technology investments made as part of it

(d) Detailed examination of the work associated with this thesis and its various needs, means, and outcomes of increasing standardization of language in practice.

References

[1] N. G. Leveson and C. S. Turner. An investigation of the Therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[2] Touby Drew and Steve Goetz. Vision of the virtual programmer - steps towards change in instrument systems for implantable medical devices. In *Proc. Int'l Conf. on Biomedical Electronics and Devices*, volume 1, pages 156–159, 2008.

[3] Medical imaging safety | COPD news of the day. `http://copdnewsoftheday.com/?p=3203`.

[4] Medtronic. `http://www.medtronic.eu/about-medtronic/medtronic-europe/index.htm`.

[5] Twitter / medtronic CRDM mx: Every 4 seconds... 400,000 ... `http://twitter.com/ritmocardiaco/status/27381846104`.

[6] List of medical device recalls > medtronic announces nationwide, voluntary recall of model 8870 software application card. `http://www.fda.gov/MedicalDevices/Safety/RecallsCorrectionsRemovals/ListofRecalls/ucm133126.htm`, September 2004.

[7] Mike Stobbe. FDA stepping up oversight of drug pumps - yahoo! news. `http://news.yahoo.com/s/ap/20100423/ap_on_he_me/us_med_drug_pumps`.

[8] ANSI/AAMI/IEC 62304:2006, medical device software—Software life cycle processes, 2006.

[9] Jennifer Oliver. Sr. director quality systems, August 2011.

[10] Jacky Estublier. Software configuration management: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 279–289, Limerick, Ireland, 2000. ACM.

[11] Minna Mäkäräinen. *Software change management processes in the development of embedded software*. PhD thesis, Technical Research Centre of Finland, August 2000.

[12] Jacky Estublier, Jean-Marie Favre, and Philippe Morat. Toward SCM / PDM integration? In *Proceedings of the SCM-8 Symposium on System Configuration Management*, pages 75–94. Springer-Verlag, 1998.

[13] Adrian S. Choo, Kevin W. Linderman, and Roger G. Schroeder. Method and context perspectives on learning and knowledge creation in quality management. *Journal of Operations Management*, 25(4):918–931, June 2007.

[14] Jad El-khoury. Model data management: towards a common solution for PDM/SCM systems. In *Proceedings of the 12th International Workshop on Software Configuration Management*, pages 17–32, Lisbon, Portugal, 2005. ACM.

[15] Harry S. Delugach. An evaluation of the pragmatics of web-based bug tracking tools. In *Proceedings of the 2nd International Conference on Pragmatic Web*, pages 49–55, Tilburg, The Netherlands, 2007. ACM.

[16] Andrew Meneely, Mackenzie Corcoran, and Laurie Williams. Improving developer activity metrics with issue tracking annotations. In *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*, pages 75–80, Cape Town, South Africa, 2010. ACM.

[17] Jacky Estublier, David Leblang, André van der Hoek, Reidar Conradi, Geoffrey Clemm, Walter Tichy, and Darcy Wiborg-Weber. Impact of software engineering research on the practice of software configuration management. *ACM Trans. Softw. Eng. Methodol.*, 14(4):383–430, 2005.

[18] David B. Leblang. Managing the software development process with ClearGuide. In *Proceedings of the SCM-7 Workshop on System Configuration Management*, pages 66–80. Springer-Verlag, 1997.

[19] Matthew Hayward. Static analysis vulnerabilities and defects: Best practices for both agile and waterfall development environments.

[20] Nachiappan Nagappan and Thomas Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th International Conference on Software Engineering*, pages 580–586, St. Louis, MO, USA, 2005. ACM.

[21] Margaret-Anne Storey, Jody Ryall, R. Ian Bull, Del Myers, and Janice Singer. TODO or to bug: exploring how task annotations play a role in the work practices of software developers. In *Proceedings of the 30th International Conference on Software Engineering*, pages 251–260, Leipzig, Germany, 2008. ACM.

[22] Jon W Spence. There has to be a better way! In *Proc. Agile Development Conference*, page 272–278, Washington, DC, USA, 2005. IEEE Computer Society.

[23] Agile project management, scrum, and agile development tool | VersionOne. `http://www.versionone.com/`.

[24] David Bellagio and Tom Milligan. *Software configuration management strategies and IBM® Rational® ClearCase®: a practical introduction, second edition.* IBM Press, 2005.

[25] IEC 62366:2007 medical devices - application of usability engineering to medical devices, 2007.

[26] IEC/TR 80002-1 ed.1: Medical device software - guidance on the application of ISO 14971 to medical device software, 2009.

[27] Dr. dobb's | beyond version control | may 1, 2001. `http://www.drdobbs.com/184414733;jsessionid=FBJ4OA1S21E4TQE1GHPSKH4ATMY32JVN?_requestid=372621`.

[28] Code of federal regulations title 21 section 820.30, April 2011.

[29] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of duplicate defect reports using natural language processing. In *Proceedings of the 29th International Conference on Software Engineering*, pages 499–510. IEEE Computer Society, 2007.

[30] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 52–61, 2008.

[31] Christine A. Halverson, Jason B. Ellis, Catalina Danis, and Wendy A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, pages 39–48, Banff, Alberta, Canada, 2006. ACM.

[32] Jürgen Münch, Ye Yang, Wilhelm Schäfer, Patrick Knab, Martin Pinzger, and Harald Gall. Visual patterns in issue tracking data. In *New Modeling Concepts for Today's Software Processes*, volume 6195 of *Lecture Notes in Computer Science*, pages 222–233. Springer Berlin / Heidelberg, 2010.

[33] A.J. Ko, B.A. Myers, and Duen Horng Chau. A linguistic analysis of how people describe software problems. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 127–134, 2006.

[34] Michael Wooldridge and Nicholas R. Jennings. Agent theories, architectures, and languages: A survey. In *Intelligent Agents*, volume Volume 890/1995 of *Lecture Notes in Computer Science*, pages 1–39. Springer Berlin / Heidelberg, 1995.

[35] David Canfield Smith, Allen Cypher, and Jim Spohrer. KidSim: programming agents without a programming language. *Commun. ACM*, 37(7):54–67, 1994.

[36] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages*, pages 21–35. Springer-Verlag, 1997.

[37] Stuart J. Russell and Peter Norvig. *Artificial intelligence: a modern approach.* Prentice-Hall, Inc., 1995.

[38] H. Lieberman and Ted Selker. Agents for the user interface. In *Handbook of Agent Technology*. MIT Press, 2003.

[39] Henry Lieberman. Integrating user interface agents with conventional applications. In *IUI '98: Proceedings of the 3rd International Conference on Intelligent User Interfaces*, pages 39–46, San Francisco, California, United States, 1998. ACM.

[40] Vittorio Castelli, Lawrence Bergman, Tessa Lau, and Daniel Oblinger. Layering advanced user interface functionalities onto existing applications. `http://domino.research.ibm.com/library/cyberdig.nsf/1e4115aea78b6e7c85256b360066f0d4/fe8651c2fe63110a85256fe2005ceece!OpenDocument`, April 2005.

[41] Marcelo G. Armentano and Analia A. Amandi. A framework for attaching personal assistants to existing applications. *Comput. Lang. Syst. Struct.*, 35(4):448–463, 2009.

[42] Anton N Dragunov, Thomas G Dietterich, Kevin Johnsrude, Matthew McLaughlin, Lida Li, and Jonathan L Herlocker. TaskTracer: a desktop environment to support multi-tasking knowledge workers. In *Proc. 10th Int'l Conf. on Intelligent User Interfaces*, page 75–82, New York, NY, USA, 2005. ACM.

[43] Lawrence Bergman, Vittorio Castelli, Tessa Lau, and Daniel Oblinger. DocWizards: a system for authoring follow-me documentation wizards. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, pages 191–200, Seattle, WA, USA, 2005. ACM.

[44] Tessa Lau, Lawrence Bergman, Vittorio Castelli, and Daniel Oblinger. Sheepdog: learning procedures for technical support. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, pages 109–116, Funchal, Madeira, Portugal, 2004. ACM.

[45] Tessa Lau. Why PBD systems fail: Lessons learned for usable AI. In *CHI 2008 Workshop on Usable AI*, Florence, Italy, 2008.

[46] Karl Anders Gyllstrom, Craig Soules, and Alistair Veitch. Confluence: enhancing contextual desktop search. In *Proc. ACM SIGIR Conf. on Research and Development in Information Retrieval*, page 717–718, New York, NY, USA, 2007. ACM.

[47] Anind K. Dey, Raffay Hamid, Chris Beckmann, Ian Li, and Daniel Hsu. a CAPpella: programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 33–40, Vienna, Austria, 2004. ACM.

[48] Touby Drew and Maria Gini. Implantable medical devices as agents and part of multiagent systems. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1534–1541, Hakodate, Japan, 2006. ACM.

[49] Maryam Alavi and Dorothy E. Leidner. Knowledge management systems: issues, challenges, and benefits. *Commun. AIS*, 1(2es):1, 1999.

[50] Jungpil Hahn and Mani R. Subramani. A framework of knowledge management systems: issues and challenges for theory and practice. In *Proceedings of the Twenty First International Conference on Information Systems*, pages 302–312, Brisbane, Queensland, Australia, 2000. Association for Information Systems.

[51] Robert Grant. Prospering in dynamically-competitive Environments: Organizational capability as knowledge integration.

[52] Ikujiro Nonaka. A dynamic theory of organizational knowledge creation. *Organization Science*, 5(1):14–37, 1994.

[53] Michael Polanyi. *The Tacit Dimension.* University of Chicago Press.

[54] V. Sambamurthy and Mani Subramani. Special issue on information technologies and knowledge management. *MIS Quarterly*, 29(1):1–7, March 2005.

[55] L. Osterweil. Software processes are software too. In *Proceedings of the 9th International Conference on Software Engineering*, pages 2–13, Monterey, California, United States, 1987. IEEE Computer Society Press.

[56] Andrew J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, pages 344–353. IEEE Computer Society, 2007.

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

[58] Olga Baysal, Reid Holmes, and Michael W Godfrey. Situational awareness: personalizing issue tracking systems. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1185–1188. IEEE Press, May 2013.

[59] Yuqing Ren, Sara Kiesler, and Susan Fussell. Multiple group coordination in complex and dynamic task environments: Interruptions, coping mechanisms, and technology recommendations. *J. Manage. Inf. Syst.*, 25(1):105–130, 2008.

[60] S. Jalali, M. Shafieezadeh, and M. Naiini. Using knowledge management in DMAIC methodology of six sigma projects. In *Information Technology, 2008. IT 2008. 1st International Conference on*, pages 1–4, 2008.

[61] Touby Drew and Maria Gini. MAITH: a meta-software agent for issue tracking help. In *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems*, pages 1755–1762, Toronto, Canada, May 2010.

[62] Ritch Macefield. Usability studies and the hawthorne effect. *Journal of Usability Studies*, 2(3):145–154, May 2007.

[63] J. Carreira and J.G. Silva. Computer science and the Pygmalion effect. *Computer*, 31(2):116 –117, feb. 1998.

[64] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[65] Touby Drew and Maria Gini. Automation for regulated issue tracking activities. Technical Report 12-010, Univ. of MN, `http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=12-010`, 2012.

[66] Touby Drew and Maria Gini. Advisor agent support for issue tracking in medical device development. In Markus P. J. Fromherz and Hector Muñoz-Avila, editors, *IAAI*. AAAI, 2012.

[67] John G. Cleary and Ian H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Trans. on Communications*, 32(4):396–402, April 1984.

[68] Ron Begleiter, Ran El-Yaniv, and Golan Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.

[69] Kirk Monteverde. Technical dialog as an incentive for vertical integration in the semiconductor industry. *Management Science*, 41(10):1624–1638, October 1995.

[70] Joseph T Mahoney. A resource-based theory of sustainable rents. *Journal of Management*, 27(6):651–660, 2001.

[71] Oliver E Williamson. *Markets and Hierarchies: Analysis and Antitrust Implications: A Study in the Economics of Internal Organization.* Free Press, New York, NY, USA, 1975.

[72] D. Binney. The knowledge management spectrum - understanding the KM landscape. *Journal of Knowledge Management*, 5(1):33–42, 2001.

[73] S. M Jasimuddin, J. H Klein, and C. Connell. The paradox of using tacit and explicit knowledge: strategies to face dilemmas. *Management Decision*, 43(1):102–112, 2005.

[74] J.-C. Spender. Making knowledge the basis of a dynamic theory of the firm. *Strategic Management Journal*, 17:45–62, December 1996.

[75] M. Boisot. *Knowledge assets: securing competitive advantage in the information economy.* Oxford University Press, 1999.

[76] Knox Haggie and John Kingston. Choosing your knowledge management strategy. *Journal of Knowledge Management Practice*, June 2003.

[77] Donald Metzler, Susan Dumais, Christopher Meek, Giambattista Amati, Claudio Carpineto, and Giovanni Romano. Similarity measures for short segments of text. In *Advances in Information Retrieval*, volume 4425 of *Lecture Notes in Computer Science*, pages 16–27. Springer Berlin / Heidelberg, 2007.

[78] MF Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[79] Martin Porter. Porter stemming algorithm. `http://tartarus.org/martin/PorterStemmer/`.

[80] Mark Watson. *Practical Artificial Intelligence Programming with Java*. Mark Watson, 3rd edition, December 2008.

Appendix A

Glossary and Acronyms

Care has been taken in this thesis to minimize the use of jargon and acronyms, but this cannot always be achieved. This appendix defines jargon terms in a glossary, and contains a table of acronyms and their meaning.

A.1 Glossary

- **Knowledge Management** – See Information and Decision Sciences (Section 2.3) and Knowledge Management (Section 7.1)
- **Medtronic** (**MDT**) – Medtronic, Inc. www.medtronic.com The world's largest medical device manufacturer and the company where sources of waste and capabilities to address them were examined for this thesis.
- **Neuromodulation** – Control of the nervous system (e.g., through electrical stimulation or drug delivery).
- **P team** – A software development team at Medtronic Neuromodulation that works closely with and is similar to the S team. This team is not typically referred to as the P team, but has been referred to as that for the purpose of this thesis.
- **Pnit** or **PNIT** – Often used in place of *issue*. A term used to refer to an issue in the Production Neuromodulation Issue Tracking System or (less frequently) the associated process of issue tracking.

- **PnitAgent** – Agent-based software created as part of this research to provide various kinds of support to issue tracking system users at Medtronic.
- **PNIT Tiger Team** – A multidisciplinary team formed to understand and help address issue tracking challenges and practices for the S and P teams. See The PNIT Tiger Team (Section 3.5).
- **Production Neuromodulation Issue Tracking System** (**PNITS**) – The issue tracking system used in the groups of interest within Medtronic.
- **S team** – A software development team at Medtronic Neuromodulation described in Pilot Users and Use Environment (Section 1.3) that is not typically referred to as the S team, but has been referred to as that for the purpose of this thesis.
- **Software Configuration Management** – See Software Configuration Management (Subsection 2.1.1).
- **Issue Complexity** – The complexity of an issue and its associated work. See Issue Complexity (Section 6.3).
- **Waste** – See the description of waste in the Introduction (Chapter 1).

A.2 Acronyms

Table A.1: Acronyms

Acronym	Meaning
AI	Artificial Intelligence
COM	Component Object Model
FDA	Food and Drug Administration
HCI	Human Computer Interaction
KM	Knowledge Management
MRCS	Medtronic Revision Control System

Continued on next page

Table A.1 – continued from previous page

Acronym	Meaning
OS	Operating System
PNITS	Production Neuromodultion Issue Tracking System
R&D	Research and Development
SCM	Software Configuration Management
SQA	Software Quality Assurance
SQE	Software Quality Engineer
UI	User Interface