

Negotiation and the Coordination of Information and Activity in Distributed Software Problem Management

Robert J. Sandusky
School of Information Sciences
University of Tennessee, Knoxville
+1 865 974-2785
sandusky@utk.edu

Les Gasser
Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
+1 217 265-5021
gasser@uiuc.edu

ABSTRACT

Publicly accessible bug report repositories maintained by free / open source development communities provide vast stores of data about distributed software problem management (SWPM). Qualitative analysis of individual bug reports, texts that record community responses to reported software problems, shows how this distributed community uses its SWPM process to manage software quality. We focus on the role of one basic social process, *negotiation*, in SWPM. We report on the varieties and frequencies of negotiation practices and demonstrate how instances of negotiation in different contexts affect the organization of information, the allocation of community resources, and the disposition of software problems.

Categories and Subject Descriptors

D.2.9 [Software Engineering]: Management – *Software quality assurance (SQA)*; H.1.2 [Models and Principles]: User/Machine Systems – *Human information processing*; K.4.3 [Computers and Society]: Organizational Impacts – *Computer-supported cooperative work*; K.6.3 [Management of Computing and Information Systems]: Software Management – *Software maintenance*; K.6.4 [Management of Computing and Information Systems]: System Management – *Quality assurance*.

General Terms

Management, Design.

Keywords

Software problem management, negotiation, coordination mechanisms, coordination theory, distributed collective practices.

1. INTRODUCTION

Software problems, or bugs, are errors or mistakes of commission or omission inadvertently introduced to software during the software development process [10]. A recent study by the National Institute for Standards and Technology on software testing infrastructure estimates that \$22-60 billion dollars are lost in the U.S. economy each year due to inadequate infrastructure for detecting and correcting software problems, and the subsequent

release of bug-ridden software to users and consumers [13] (p. ES-3).¹ This cost estimate includes both the costs to developers to repair problems and the costs to user organizations to mitigate their effects. Software problems can also have significant social and human impacts. Faulty software used to control radiation therapy machines caused at least six accidents resulting in deaths and other serious injuries in the mid-1980s [12]. Software problems were determined to be a cause of an airline crash in Guam that killed 225 people [14].

It is impossible to create non-trivial software that is completely stable and free of defects for several reasons: the groups that specify and evaluate sophisticated software artifacts comprise multiple participants with diverse views and interests; it can be difficult or impossible to reach consensus on requirements and interpretations of requirements against which to evaluate behavior; even if a software version seems defect-free at some moment, situations of use and bases for interpretation change, so new defects may arise with no change in software codes; software modifications may introduce defects; and so on. When software problems are discovered, it is important that they be managed (evaluated, triaged, repaired, etc.) in the most efficacious manner. The management of software problems is social concern in two senses: SWPM is a collective activity for all but the most localized programs, and the impacts of SWPM extend to user communities and the society at large for critical artifacts such as infrastructure and high-risk systems.

Gaining access to large sets of data has been an impediment to performing this kind of research in the past because of the closed nature of many large software projects. Recently, however, Free/Open Source Software (F/OSS) development communities have begun maintaining open repositories of software design and development activities such as bug reporting and tracking tools, and these are publicly available for study. This paper describes some of the research we've done as part of a larger project on distributed collective practices in F/OSS. We use data drawn from the bug report repository maintained by the Mozilla development community (<http://www.mozilla.org/>).

Background information on related research, bug reports, and SWPM is described in the next section. The method used for the study is described next, followed by detailed analysis of some examples of negotiation in SWPM. Finally, the relationships

¹ This dollar estimate excludes costs "...associated with mission critical software where failure can lead to extremely high costs such as loss of life or catastrophic failure." [13] (p. ES-3).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

GROUP'05, November 6–9, 2005, Sanibel Island, Florida, USA.

Copyright 2005 ACM 1-59593-223-2/05/0011...\$5.00.

between negotiation and information, activity and problem resolution and the future directions for this work are discussed.

1.1 Related Research

Until now, little focused research has been done to understand the socio-technical process of managing software problems in distributed development environments. Selected research on SWPM, the theory of coordination, distributed and tool-mediated software development, and negotiation is reviewed here. Some of this research suggests that distributed teams cannot succeed without access to richer modes of interaction (face-to-face interactions).

Negotiation has received some attention from researchers as a basic social process in software problem management and software development in general. Carstensen, Sørensen & Tuikka [1] discuss the role of negotiation in software problem management in general terms, in particular as part of the processes of (1) diagnosing and classifying bug reports and (2) determining which solutions would be acceptable. They state “one out of four bug reports required discussion and negotiation between a tester and designer, or between the spec-team and a designer” (p.10). They do not, however, provide any more detail about the nature, extent, contexts, or outcomes of specific instances of negotiation.

Schmidt and Simone [20], in their seminal paper on coordination mechanisms, state that the artifact and protocol that comprise a coordination mechanism alleviate “the need for ad hoc deliberation and negotiation”, thus reducing “the complexity of articulation work” (p.162). This is true to the extent that the coordination mechanism makes work routine (they cite Carstensen, Sørensen & Tuikka’s [1] description of software problem management to illustrate their theory): negotiation is not needed to route the next bug report through the normative SWPM process. The coordination mechanism, however, does not eliminate negotiation from the process of managing individual bug reports: each bug report presents its own unique circumstances, contexts, and sensemaking trajectory.

Herbsleb and Grinter [8] report upon the serious coordination difficulties experienced by a globally-distributed software development project in a traditional software development organization, noting that team distribution inhibited and limited opportunities for informal contact and that mechanisms for support of spontaneous communication are necessary for project success. Herbsleb et. al. [9] measured the delay caused by the global distribution of software development groups and concluded that cross-site work suffered significant delays compared to same-site work.

Several previous researchers have looked at how software engineering tools are used to articulate / coordinate software development activities. Grinter [7] analyzed one organization’s use of a software configuration management tool, concluding that the tool provided value by storing, organizing and revealing information about the work of other developers, such as change history and code authorship. In another study of tool use, de Souza et. al. [21] looked at configuration management data as well as bug reports and e-mail usage to illustrate how and when transitions between private and public workspaces occur. Yamauchi et. al. [24] examined two open source projects and their use of “lean media” to support coordination of activity. They concluded that the tools used in the projects they studied were

used to coordinate work *after* the work had been done, but they did not include examination of a bug report repository like the one used by the Mozilla project.

Egger and Wagner [4], in one of the most detailed examinations of negotiation in the CSCW literature to date, analyze negotiation of a scarce resource – time – in a surgical clinic. In this setting, they focused on how a “small, semi-autonomous functional group which can be regarded as a social system in itself” (p. 251) manages time. They advocate using Strauss’ [22] theory of negotiation as a way to understand problem solving behavior in an organization because Strauss’ theory doesn’t require a pre-conceived expectation of either the outcomes or the details of the process applied in particular instances (successful, contentious, determined by power / authority relations, etc.)

Larsson [11] focuses on how participants in a globally distributed design team use negotiation to continually construct and re-construct common ground, the shared understanding between people that allows them to conduct effective social interaction. He draws examples from two occasions where the two otherwise distributed teams met face-to-face for the purpose of first, brainstorming, and then, later, developing consensus on their design. While he emphasizes the centrality of negotiation as an active process employed in support of sensemaking during the design process, Larsson’s conceptualization of negotiation is not detailed. The paper has little to say about how negotiations are conducted when the participants are geographically distributed, instead proposing that rich media or periodic face-to-face interaction are critical to supporting distributed design activity.

1.2 Bug Reports

Bug reports are central artifacts within almost all software development communities and are a nexus of information and activity related to both software problem management. A bug report contains:

- General descriptive information about the problem
- Taxonomic entries selected from controlled vocabularies, that classify bug reports (component, priority) and denote current bug report status (status, resolution)
- Instructions for how other people can recreate the bug (reproducibility is a desirable characteristic)
- Descriptions of what work has been done, what work is underway, and what work is planned or committed to in the future
- The identity of people involved in the reporting, triage, analysis, coding, verification and resolution (if any) of the bug
- Timestamps and automatically-inserted comment tags for each change made to the bug report record
- Opinions, hypotheses, clarifications and conjecture, entered as text and made available to anyone in the community
- Attached documents such as patches, screen shots, and stack traces
- References to other bug reports in the repository [19]

A bug report contains tracking, interpretive, and management information as well as factual information. The bug report makes a problem visible to other people and organizations and helps to coordinate and support a chain of activity toward resolving the bug. A community's SWPM process often defines specific roles that are played by individuals in the process. The bug report, the bug report repository, and community conventions comprise a *coordination mechanism* [20], providing an artifact – the bug report – and a related protocol supporting SWPM. In this community, the protocol is embodied in both the community's SWPM conventions and how the repository is programmed. The repository used here relies upon transmission of automatic notifications, by e-mail, of changes to bug reports to a variety of actors associated with each bug report. The roles associated with a bug report include reporter, assignee(s), quality assurance contact(s), etc., in order to support the proper and open handling of bug reports bugs [6].

Also crucial in understanding SWPM is the distinction between a *bug* and a *bug report*. A “bug” is a phenomenological situation, generally lacking tangible form, in which software system behavior fails to conform to some group's expectations and values in a manner judged to have negative sentimental, moral, or practical import. A “bug report” is created to represent a bug, and functions as a formal and tangible representation of the bug. In different situations, members of the community may do work on either an actual bug or on a bug report. If a programmer is making and testing code changes, he is working on a bug. If a manager is evaluating whether a new bug report is a duplicate of another existing bug report, she is reading, evaluating, managing and linking records in the repository. The distinction between bug reports and bugs is critical to our analysis because typical collective repository tools support *bug reports* as first class objects, but the aims of analysis and action are bugs, which are not first-class objects in repository tools. (For example, many bug reports may contribute to the understanding or resolution of a single bug [5, 19].)

Finally, empirical observation reveals that bug reports often represent entities other than software problems. For example, bug reports may represent requests for new features, collections of related bug reports (“meta-bugs” or networks of linked bug reports [19]), or invalid problems (that is, issues that the community decides are not going to be worked on).

1.3 Software Problem Management

Software Problem Management (SWPM) is one constituent part of both traditional or F/OSS development processes. SWPM is typically described as a normative engineering process or a workflow comprising the steps shown in Table 1 [1, 2, 3, 17]. This paper, based upon a re-evaluation of the normative point of view, is focused on activity that usually occurs during phases two (Bug Report Triage and Assignment) and three (Expert Analysis, Fix Development, Testing and Deployment) of the normative SWPM process presented in Table 1.

Evidence from this community's bug reports indicates that SWPM involves many interrelated, nested sub-processes including negotiation (a basic social process), the triage process, as well as sensemaking sub-processes such as individual and collective debugging. SWPM is by itself a single component of a larger, more complex software development process.

Table 1. SWPM Process

Normative Software Problem Management Process
Phase One: Problem Identification and Reporting
Step 1. Problem Identification - User notices anomaly / mistake during normal use or testing - Actors identifying problems may be testers [1, 2] or customers / users [2] <i>Encounter between a human and a problematic phenomenon is subjective, complex, and does not necessarily result in the creation of a bug report</i> Step 2. Problem Reporting - Bug report created, enabling collective management of the SWPM process - Actors creating the bug report may be testers [1] or intermediary (e.g., help desk) [2] <i>Bug reports may represent enhancement requests, accidental submissions, etc.</i>
Phase Two: Bug Report Triage
- Problem re-creation is attempted - Priority and bug report assignment are assessed by a group of software designers whose primary role is the management of software problems [1] or by “marketing engineers” and assignment by “programming managers.” [2] <i>Reducing the number of bug reports needing attention (e.g., identifying duplicate bug reports) is a priority in the Mozilla community. Identification of the small number of high priority problems is also important</i>
Phase Three: Expert Analysis, Fixing, Testing, and Deployment
- Assignee (software designers [1] or software engineers [2]) investigates, determines cause, evaluates repair options, consults with other experts, coordinates the work of multiple experts, etc. - Assignee modifies and tests the software to resolve the bug and deploys the change if successful. <i>Collective debugging, individual debugging, voting, and negotiation are some of the sensemaking, community-specific, and basic social processes employed by the Mozilla community in this phase of the SWPM process.</i>
Phase Four: Bug Report Verification and Closure
Step 1. Fix Verification - Quality assurance verifies that the bug has been corrected (the platform master responsible for the software build [1] or the integration team [2]). <i>In the Mozilla community, the terminal status of most bug reports is “verified.”</i> Step 2. Problem Closure - Bug report is marked “closed” by the central file manager [1] or the software engineer [2].

2. METHOD

A systematic random sample of 385 bug reports was drawn from a F/OSS bug report repository containing approximately 182,000 bug reports. The sample size was determined using an approach reported by Powell [15] (p.75). A conservative sample size is suitable here because we did not have complete information about the variability of all characteristics of the bug reports at the time the sample was drawn. A large, conservative sample size helps ensure that the sample is representative of the variety of bug reports found in the repository. The bug reports in our sample had been initiated over a four and half year period. The bug report is the primary unit of analysis in this study.

Each bug report in the sample was treated as a text and was read and analyzed using a content analytic approach [25]. Concepts, phenomena, and relationships between phenomena were inductively derived and refined as they emerged from the bug reports during data analysis using grounded theory [23]. As the analysis proceeded, the conceptual labels were combined into a smaller number of categories. The properties and dimensions of each of the categories were also identified during open coding. This coding was performed by a single researcher, so intra-coder reliability is a concern. Reliability concerns were mitigated here by the development and use of a consistent coding procedure and codebook.

The grounded theory procedure also includes the frequent creation of memos, which are short, focused, and informal write-ups of the analytic process, provisional findings, new questions, etc. The memos were shared with other project members as they were written, and the critiques generated by the other members of the project team were incorporated into subsequent memo revisions and helped direct and focus subsequent analysis.

3. NEGOTIATION IN SOFTWARE PROBLEM MANAGEMENT

The F/OSS community studied here is engaged in “distributed collective practice” (DCP) [5], making sense of bug reports. The study of a DCP focuses on examining the activities, objects, social and technical contexts, and common problems and methods within large-scale socio-technical systems. Negotiation [22] is a basic social process frequently employed to address uncertainty and complexity, coordinate activity, and support distributed collective sensemaking, using the bug report as its locus.

Negotiation is a core concept that has emerged from this analysis. Strauss [20] defines negotiation as a process:

“...when individuals or groups or organizations of any size work together ‘to get things done’ then agreement is required about such matters as what, how, when, where, and how much. Continued agreement itself may be something to be worked at.... Putting the matter thus suggests that negotiations pertain to the ordering and articulation of an enormous variety of activities (p. ix).”

There are multiple levels at which negotiation may be used in SWPM within the F/OSS community. At the higher, structural level [22], are issues that are infrequently or never the subject of negotiations in the bug reports. Examples of these are the format of the bug report or the SWPM protocol in use, which comprise

the coordination mechanism [20]. The normative SWPM process (Table 1), based upon practice in traditional software development organizations, is not questioned or subjected to negotiation within bug reports. The particular values available in controlled vocabulary fields within the bug report, like status and resolution, are not commonly negotiated; on the other hand, the names of the products and components are more likely to be changed in order to keep pace with the changing structure of the software itself, and thus could be negotiated issues (although the sample reported on here provides no evidence that issues such as these are negotiated within bug reports).

SWPM in any organization, whether traditional or F/OSS, is a negotiated order: it could always be otherwise. Note, for example, the differences between the processes described in [1] and [2] regarding who has responsibility for problem prioritization. Crowston [2] also points out that the coordination mechanism used to support similar processes can vary from organization to organization. Within an organization, different choices can be made, for example, about how programmers are assigned to particular bug reports. The focus of this paper is not on negotiation in the large, or at the structural level, but on negotiation in the small, as it pertains to the management of information, resources, and activity related to individual bug reports.

The nature of this F/OSS development community and F/OSS communities generally is another aspect of the structural context surrounding SWPM and negotiations therein. For example, F/OSS communities are often characterized as meritocracies [16] in contrast to traditional, bureaucratic software development organizations. While examination of these factors is beyond the scope of this paper, it is useful to acknowledge that factors seemingly far removed from the management of a particular bug report might play a role in shaping the community’s actions and the outcomes.

Negotiation occurs frequently as members of this F/OSS development community manage software problems. Table 2 shows that a majority (61%) of the bug reports in the sample contain instances of negotiation. 27% of the sampled bug reports contain negotiation on more than one issue.

Table 2. Frequency of negotiation

Distinct Issues Negotiated per BR	Percentage of BRs	Cumulative Percentage
6	<1%	<1%
5	1%	1%
4	4%	5%
3	10%	15%
2	12%	27%
1	34%	61%
0	39%	100%

Based upon the empirical investigation described here, instances of negotiation vary along several dimensions:

- the issue negotiated (see Table 3)
- level of intensity (civil vs. confrontational)
- duration (measured either by the number of distinct contributions to the negotiation sequence or the elapsed time from the start to the end of the negotiation)
- number of participants (number of distinct contributors)
- roles of the participants
- number of potential outcomes (distinct points of view, etc.)
- effect (impacts of distinct outcomes)
- status (the disposition of the negotiation)

For example, the level of intensity of an instance of negotiation can be more or less civil, democratic, or even sometimes rude. The tone is typically civil, but in one bug report (bug report 40), one negotiator begins his comment with “What’s wrong with you people?” In terms of duration and number of participants in a single instance of negotiation, we have found examples in our sample of 12 contributions from 5 people and 17 contributions from 4 people.²

Table 3. Issue frequency

Issue Negotiated	Percentage of BRs
Is there a problem?	25%
What is best design?	22%
How should the BR be managed?	14%
Is this a duplicate or dependent BR?	13%
Is this problem fixed?	8%
Who’s responsible for fixing this?	7%
When should fix be scheduled?	5%
What is this BR’s scope?	4%
What is this BR’s priority?	3%
Is the fix’s code acceptable?	2%
Bug, feature, or enhancement?	2%
Is bug’s cause determined?	1%
What type of BR is it?	1%

Table 3 summarizes the percentages of bug reports that contain instances of negotiation on various issues. While F/OSS community members engage in negotiation about many aspects of the SWPM process, negotiation of (1) whether or not a problem exists and (2) issues of system design (insofar as design relates to

the management of software problems) are the most frequently negotiated issues. Bug reports are rich in proposals for designs that can be used to fix the bugs represented by the bug reports. The participants are almost always geographically distributed. Thus, design negotiations in support of SWPM are open, distributed, and collaborative in support of the continuous (re)-design of the F/OSS system.

In some cases, a bug report contains a single statement related to a negotiable issue. No one else ever comments on the statement, whether to support it, propose an alternative or present an argument against it. We code these as instances of negotiation, using Strauss’ notion of *implicit* negotiation [22]. In such cases, the lack of a specific response represents the community’s implicit agreement with the original statement. In some of these cases, the bug report is subsequently marked as a duplicate of another bug report and thus no further comments are likely to be added to that bug report. The same issue may, however, be negotiated in the related bug report.

In the following examples, identities of individuals are changed and some system-related names are disguised (framed by angle brackets: thus Microsoft Windows becomes <Commercial Operating System>). The first example shows a simple design negotiation that includes the rationale behind the two points of view expressed by the participants. Example 2 is drawn from a bug report that represents a request for an enhancement rather than something in the system that is broken. This example also shows the influence of the community’s values, in this case ease of system use by multiple user constituencies, on the design. The third example is more complex and includes evidence of concurrent negotiation on four different issues: What is best design?, Who’s responsible for fixing this?, How should the BR be managed?, and Is this a duplicate or dependent BR? The participants in the negotiation in this BR also participate in negotiations and activity in the other bug reports that are part of the same bug report network.

Example 1: Bug Report 200

In bug report 200, the preferred way of handling of one aspect of the user interface – the collapsible toolbars – is the subject of a design negotiation. The bug, which appeared as a regression after a normal system build, was that application toolbars, after being collapsed (that is, made invisible) by the user, could no longer be made visible – the interface “tabs” or “handles” were no longer on the interface. The report states in a comment on day 1:

Reporter: The correct fix for the problem would be not allowing collapsing of ‘File Edit...’ menu bar. [Turn 1: What is best design?]

On day 2, commentator_1 asks:

Commentator_1: Why is it suggested that the correct fix would be to disallow collapsing of the “File Edit...” toolbar? When this toolbar behaved correctly, the collapsed toolbar were easily retrieved by clicking on the little tab. What happened to the little tabs? That’s the problem. [Turn 2: What is best design?]

The reporter responds within one hour on day 2:

Reporter: In comparison to <application name>, the “File Edit...” menu toolbar shouldn’t be collapsible. Once can argue as long as we could provide a way to expand the collapsed toolbars then things would be fine. However, this may seem odd when comparing to other apps. [Turn 3: What is best design?]

² Other bug reports in the repository not included in this sample contain hundreds of comments.

As it turns out, the reporter's argument that other applications don't provide collapsible toolbars failed to convince the community and the "File Edit..." toolbar remains collapsible to this day. In Example 1, misalignment about generally proper application behavior occurs and is expressed publicly in the bug report. Two different points of view are expressed in a pointed but not hostile manner in this regression bug that was marked fixed on day 3.

Example 2: Bug Report 196

Bug report 196 is a request for an enhancement, not a report of a bug. Someone asked for the ability to create versions of the system more selectively. That is, the enhancement requestor, Commentator_2, wanted to be able to leave a major, but optional, component of the system out.

The reporter starts the design negotiation by saying at the time the bug report is created (day one):

Reporter: This may be terribly easy: i.e., simply conditionally excluding the [component] meta module in [systemName]/modules/staticmod if the DISABLE_[component] variable is set. **[Turn 1: What is best design?]**

The first commentator replies at 11:04 on day one:

Commentator_1: except that the installer is expecting [component] so we need to use even yet another manifest file for creating packages (i'm not sure if we have to use another set of .jst files in xpininstall, or if the .xpi files are created solely from the directories that the manifest creates). **[Turn 2: What is best design?]**

The reporter responds at 11:45 on day one:

Reporter: Who cares about the installer? This is for people that are building the [systemName] tree in order to embed it. **[Turn 3: What is best design?]**

The module owner (Commentator_1) replies at 16:18 (day one):

Commentator_1: maybe the *installer* you don't care about, but embedders still probably want a functional list of stuff to embed. Are all the embedding clients keeping their own list of what they pull out of dist? **[Turn 4: What is best design?]**

Assignee_1 creates a patch and adds this comment at 18:44 on day one:

Assignee_1: Created an attachment (id=44324) simple fix, just like <reporter> suggested **[Turn 5: What is best design?]**

The bug report is untouched for several days, until day eight when assignee_1 submits another, much more complex patch. The bug report doesn't record anything between day one and day eight, but it's possible that the people working on this problem and/or other community members were communicating using other means (chat, email, face-to-face, etc.).

At 12:13 on day nine, after the second patch is code-reviewed by the reporter, the original requester, commentator_2, writes:

Commentator_2: minor enhancement request....when [systemName]_DLL is set, [systemName].exe should link to [subsystemName].dll rather than statically linking the same libraries.

That will make isolating bugs much easier by having a testbed [systemName].exe loading the same [subsystemName].dll as an embedded client. **[Turn 6: What is best design?]**

Later on day nine, at 12:20, assignee_1 submits a third patch:

Assignee_1: Created an attachment (id=45281) implement <commentator_2>'s suggestion (not really tested yet) **[Turn 7: What is best design?]**

On day ten, at 13:16, the reporter, who seems to have been code-reviewing assignee_1's four patches and, to some extent, guiding her work (note her comment "just like <reporter> suggested"), assigns the bug report to himself. Comments continue to contain a high level of code detail as the ramifications and side effects of the enhancement emerge and are addressed. The reporter submits another seven patches before the problem is resolved on all supported platforms (Windows, Macintosh and Unix/Linux). Commentator_1 takes on the role of code reviewer for the reporter's patches. The bug report is closed on day 29, after 35 comments and 11 patches are posted and two formal code reviews are conducted. Implementing this enhancement did not turn out to be "terribly easy."

Most of the design negotiation took place early in this bug report's trajectory and was concerned with the overall ramifications of making this change on different kinds of system users and situations of usage. Four individuals participated in the negotiation making 7 contributions during the first 9 days the bug report was open. The reporter clearly was advocating for making the change while commentator_1 presented challenges or objections. The final contribution to the design negotiation was a request for a further enhancement, made by commentator_2 between assignee_1's second and third patches. This seems to have been accepted without any response, as assignee_1's third patch incorporates commentator_2's request.

While the level of effort expended to resolve the issues raised in this bug report was not anticipated, the negotiations did bring the enhancement, its implications, and options for design into the open and make them available for comment in an open and constructive way [6]. The bug report also provided a common space where the requester (commentator_2), the advocate / implementer (reporter), the first assignee (assignee_1) and the skeptic (commentator_1) could negotiate both the scope and merits of the request as well as manage the work of implementing it. The assignees and code reviewers could report status. Other commentators could make suggestions and provide encouragement.

Turns 2 through 4 in example 2 show an example of the interrelationship between the design of the system and the effects of design decisions on different types of users. "Installers" are those who simply want to install this system and run it. "Embedders" are people who are interested in taking this system, or selected components of it, and include as a subsystem it in other systems. This example shows, through this instance of negotiation, that managing bug reports software changes is not purely a technical issue.

Example 3: Bug Report 147

This bug report contains instances of negotiation on four separate issues. The instances of negotiation run concurrently: they all overlap during the life cycle of this bug report.

The reporter begins negotiation on three of the issues in the description area of the bug report (day 1):

Reporter: Behavior should be consistent and subject to control by event handlers.... Expected Results: Forms with a single input are

submitted, even if a keypress event handler attempts to prevent it with `stopPropagation()` and `preventDefault()`. [Issue 1, turn 1: What is best design?]

Reporter: I'm guessing at the correct component. `<Component_1>` or `<Component_2>` are other obvious possibilities, but this seemed like the best fit. [Issue 2, turn 1: Who's responsible for fixing this?]

Reporter: If it's decided that letting Enter submit is desirable, this is probably two bugs: one to get Enter behaving consistently, one to get the event handler's efforts respected. [Issue 3, turn 1: How should the BR be managed?]

On day 4, a commentator begins the fourth negotiation instance on the issue Is this a duplicate or dependent BR?

Commentator_1: See also bgu `<X>`, "Enter in text input submits form iff there is exactly one text input". [Issue 4, turn 1: Is this a duplicate or dependent BR?]

The reporter adds to two of the instances of negotiation on day 5. This marks the termination of negotiation on the issue "What is best design?" in this bug report.

Reporter: If `<commentator from bug X>`'s spec is implemented in such a way that Enter keypress events on the input in a single-input form can be caught, my concerns would be completely addressed. (I don't actually much care what the default behavior is, as long as I can catch the event and override that behavior.) [Issue 1, turn 2: What is best design?]

Reporter: Similar but not identical. `<commentator from bug X>`'s proposed spec would address at least part of my problem, since he proposes activating whatever submit control is appropriate given his rules, and my `<form>` has no submit control. Presumably, in the absence of such a control, the form would not be submitted. [Issue 4, turn 2: Is this a duplicate or dependent BR?]

At this point, the bug report records no activity for almost two months. On day 61, a second commentator makes an implicit contribution to the negotiation on who's responsible for fixing this by changing the component field to a new value (changing it from "event handling" to "user interface design"). Changing the component field automatically changes the assignee and QAContact fields, an implicit re-attribution of responsibility by re-assignment of the bug report.

Commentator_2: Moving to UI/design.... [Issue 2, turn 2: Who's responsible for fixing this?]

On day 63, the remaining three instances of negotiation are concluded. The issue of "Who's responsible for fixing this?" is firmly addressed by the new assignee, who turns it back to the original assignee and concludes the negotiation on this issue with this comment and by resetting the component field to the value chosen by the reporter on day 1:

Assignee_2: This ain't a UI design problem, it's an Event Handling problem (or possibly an `<Component_2>` problem). [Issue 2, turn 3: Who's responsible for fixing this?]

Later on day 63, the assignee concludes the remaining two negotiations in this bug report in a single, complex comment:

Assignee_1: The fact that you can't cancel the submission with the enter key is a dupe of bug `<Y>`. A discussion of the second part about whether or not we should submit forms with multiple text fields on Enter is covered in bug `<X>`. *** This bug has been marked as a duplicate of `<Y>` *** [Issue 3, turn 2: How should the BR be managed?] and [Issue 4, turn 3: Is this a duplicate or dependent BR?]

Example 3 clearly illustrates the complex and contested relationships between a bug (the observed, experienced system phenomenon) and bug reports (the socially constructed representations of bugs and other types of things, like bug report networks, parties, etc.). In this case, the reporter did experience misalignment between the system's behavior and expectations. However, this bug report didn't get "fixed:" it got marked as a duplicate of bug report Y. In addition, as a result of the negotiation regarding how the bug report should be managed, the community decided that another portion of the reporter's issue would be tracked by a third bug report (bug report X). The reporter added a comment reiterating the points he made in this bug report (see comments marked "issue 1, turn 1" and "issue 1, turn2," above) to bug report X one day after this bug report was marked resolved: duplicate. The reporter also registered an interest in bug report Y by adding himself the "carbon copy" list to ensure he received e-mail updates on the progress on and changes to bug report Y. The reporter was still invested in seeing that the problem he had reported was actually resolved. He was also willing to abide by the community's decision regarding how the bug report he had created should be managed (issue 3). Assignee_2 was active in the eventual resolution of bug report X, but did not play a specific role (e.g., assignee, QAContact). Assignee_2 was never recorded as a participant in bug report Y. Assignee_1, on the other hand, was very active throughout the history of bug report Y, acting as the initial assignee, yet never participated in bug report X.

4. DISCUSSION: NEGOTIATION, INFORMATION AND ACTIVITY

The distributed, collective activity of SWPM is mediated in F/OSS communities by information and communication technologies. This community's bug report repository forms the coordination mechanism – the bug report and the associated protocol – for supporting SWPM. Collective interpretations and actions (such as judgments on causality, management actions and code creation) are actively constituted, constructed, coordinated and documented in the bug reports held within the repository. Bug reports also reveal who participates in the SWPM process and how the information contained in the bug report affects individuals' workloads.

The instances of negotiation found in bug reports help us trace the genesis and evolution of interpretations, specifications, decisions, rationales, etc. Instances of negotiation are ubiquitous: 61% of the bug reports in our sample contain negotiation on at least one issue and 27% contain evidence of negotiations on more than one issue. We estimate that on the order of 110,000 of the 182,000 bug reports in the repository contain instances of negotiation and nearly 50,000 of those contain instances of negotiation on more than one issue. Decisions affecting an individual bug report's trajectory through the SWPM process are frequently taken after negotiation has occurred on relevant issues.

The SWPM process is itself one constituent part of the larger software development process. The SWPM process can be decomposed into interrelated, nested sub-processes such as the four phases of the normative SWPM process presented in Table 1 (e.g., triage), basic social processes such as negotiation, and sub-processes such as collective and individual debugging.

Beyond interpretation as a workflow or an engineering process, we see SWPM as a *sensemaking* process [26]: a contingent social response by the members of the development community as they attempt to “structure the unknown” (p.4) in response to an interruption to the normal state of affairs, a situation in which predictions have broken down (p.5). Sensemaking is an appropriate analytic lens for SWPM because it addresses “how the text is constructed as well as how it is read. Sensemaking is about authoring as well as reading” (p.7). Each bug report is a unique text that, while based upon a common artifact and supporting protocol, is dynamically created by the community, and used to marshal, organize, and interpret information, as well as shape and coordinate subsequent sensemaking activity. When SWPM is conceived of as a sensemaking process, finer-grained processes such as individual and collective debugging (see example 2’s interactions between the reporter and assignee_1) can also be interpreted as sensemaking endeavors instead of deterministic workflows.

A bug, by its nature, represents misalignment within the community. A user has noticed some behavior in the system that is at odds with established standards or expectations. These expectations can be set formally by the community (e.g., conformance to a standard) or be the expectations of the user (e.g., when I select an interface element that looks like X, I expect behavior Y). Given problematic phenomena experienced by a user, tester, or developer, a bug report may be created to represent those phenomena. While some of the symptoms, conditions, and evidence may be known and entered into the bug report, other questions may remain about the causes of the phenomena; the boundaries of the suspect system behavior; where the issues raised by a bug report fit within the constellation of the community’s values; what approach should be taken by the community in response to the bug report; what code changes, if any, should be generated to re-establish alignment. Negotiation is a basic social processes employed by the community to “make sense” of problems and move toward re-alignment within the community regarding issues of behavior, values, design, causes, and code, and thus to problem resolution.

In order to re-align system behavior with expectations, the community must come to agreement about (1) whether the phenomenon reported by the user represents a software problem, (2) whether the reported phenomenon has already been recorded in the bug report repository, (3) the desired behavior of the system when the problem is resolved, (4) how the community should allocate resources to manage the bug report and resolve the problem it represents, (5) the best technical design for the resolution to the problem, (6) the priority of the problem relative to all other outstanding problems, (7) when the fix should be scheduled for implementation, and (8) whether the submitted fix resolves the problem. Negotiation is frequently employed by the community as part of the distributed collective process of data interpretation, construction of explanations and code, and the coordination of activity.

Instances of negotiation in bug reports make many aspects of SWPM visible. In some bug reports, no evidence of explicit negotiation exists: in these cases, what needs to be done is somehow obvious to the community members (e.g., this bug report is obviously a duplicate of another bug report). In other situations, the negotiations can be long and involve many

community members (see examples 2 and 3, above). When, in a design negotiation, more than one point of view is expressed, it may be that no single point of view prevails, but that more than one approach contributes to the implemented changes. The design of a system is not only in what is expressly included, but also in what is excluded, in a positive ground / negative ground relationship. Design negotiations also give us information about the community’s values when participants include the rationale for their points of view, as in example 2 (above), where providing improved support for a particular subset of the system’s users was negotiated. Recovery of design rationale, especially in F/OSS development communities that produce little or no formal documentation, motivates continued investigation into the activities and information represented in bug reports.

Negotiation is employed by this community as a means to organize information and allocate resources in support of the larger goal of managing and resolving software problems and thereby improving the quality of the software. In the SWPM process employed here, with a peak bug report creation rate of about 140 per day, information management activities are arguably as important as problem solving activities because effective resource allocation (e.g., valuable programmer time) depends upon effective information management. Information management activities are generally antecedent to the actual correction of the code and thus to the resolution of the problem.

Example 3 illustrates how negotiation is employed in information management activities through negotiation of three different issues: (1) Who’s responsible for fixing this?, (2) How should the BR be managed?, and (3) Is this a duplicate or dependent BR?, which all bear upon the management of information. Communal agreement on the issue of Who’s responsible for fixing this? has the effect of *classifying* the bug report as related to a particular system component. Once a bug report is classified (the determination of the answer to an information management question), a particular assignee is “allocated” to design and code the resolution to the problem. The determination of this classification is one marker of the transition from the “triage” to the “expert analysis” phase of the SWPM process. As example 3 shows, the determination of Who’s responsible? is contingent upon the agreement of the other community members engaged with this bug report. The initial component classification was “event handling,” which was changed to “user interface design” and back to “event handling.” Note that two other components were mentioned during the negotiation as possible correct component classifications.

The second information management issue negotiated (How should the BR be managed?) shows a typical community response to complex bug reports, factoring complex phenomena into multiple, separate bug reports. This instance of negotiation is closely related to the third information management issue negotiated (Is this a duplicate or dependent BR?) in this bug report. The community determines that the phenomena reported in are already represented in two previously existing bug reports. This bug report is closed as a duplicate of one of those pre-existing bug reports. Ultimately, the community is able to ignore this bug report and take it off the list of items requiring attention. Removing a bug report from the list of active bug reports represents tangible progress for the community simply by

reducing the number of bug reports requiring the community's attention.

All three examples given above include instances of negotiation of the issue What is the best design? This is the second most frequently negotiated issue, occurring in about one-third of the bug reports in this sample. Negotiations on design are usually focused on relatively bounded aspects of the system because they occur in the context of trying to solve a particular software problem (see example 1, above). However, some bug reports are explicit requests for enhancements (example 2), some of which may be larger in scope. Negotiation about design may be necessary because the ideas for fixing problems must be developed and expressed. In some cases, there may be multiple ways to bring the behavior of the system back into alignment with expectations. In other cases, a community member, often the assignee or the reporter, wants to express the approach she or he prefers in order to allow other community members to comment on the proposal.

The choice of a design approach bears directly upon resource allocation: more programming time is needed to implement more complex designs. Example 2 shows an example of the emergent, informal and continuous design process typical of F/OSS software development projects. What was first perceived to be a straightforward change was found to be complex and time consuming: an iterative, trial-and-error process of coding, testing and verifying, and reflection upon the result of the change.

In Strauss' terms [22], social order is negotiated order. Strauss uses the term structural context to describe "that 'within which' the negotiations take place, in the largest sense" [22] (p. 98). The coordination mechanism – the bug report and the associated protocol – is considered a fixed part of the structural context. The issues represented by individual bug reports are considered to be fluid, non-structural and subject to negotiation. This paper examines negotiation at a micro level, within individual bug reports. Strauss uses the term *negotiation context* to describe the "properties entering very directly as conditions into the course of the negotiation itself" (p. 99). Bug reports, as dynamic artifacts, have different (and potentially unique) groups of people associated with them: the various reporters, assignees, commentators, and so on. Negotiation in bug reports represents management of information, resources, and thus the social order within each of these groups. Negotiation at the bug report level is emergent, dynamic, and organic in the service of both managing bug reports (which may or may not represent problems) and making sense of the software problems that some bug reports represent. In contrast, the bug report repository – this community's coordination mechanism – changes far more slowly and within the parameters of the normative SWPM process described in Table 1. While existing theories of coordination [2, 20] can be applied to the analysis of structural context (e.g., coordination mechanisms), like a SWPM system, they cannot be applied to the study of individual instances of negotiation that occur within individual artifacts.

5. FUTURE WORK

A number of questions have been raised by our work on negotiation in F/OSS SWPM. One set of questions relates to one of the larger goals of our project, determining factors that lead to better or worse performance by communities involved in SWPM.

Having identified numerous instances of negotiation in a sample of bug reports, can we predict SWPM outcomes based upon data regarding instances of negotiation? Other questions about the relationships between negotiation and the roles of community members remain to be investigated. Who (that is, which role(s)) typically negotiates (reporter; assignee; commentator; QA contact)? Are there patterns regarding peoples' roles and typically negotiated issues (e.g., QA Contacts often negotiate about whether the problem is fixed or not, but rarely negotiate design alternatives)? How do peoples' roles and the issues negotiated vary depending upon the bug report's current SWPM phase?

The generalizability of the results presented here can be evaluated by examining bug reports and SWPM in other settings, such as traditional software development organizations or in other F/OSS communities with different characteristics. The research reported here relied exclusively upon reading the sampled bug reports and therefore excludes information that may be available in configuration management tools (e.g., CVS), mailing lists, chat archives, etc. Use of additional methods, like participant / observation and interviewing, could be applied in future qualitative examinations of SWPM allowing analysis of information not accessible using our present method.

Results from this qualitative investigation are also being applied to the development of computational tools for investigating distributed collective practices [17]. One example is the use of semantic analysis of linguistic interaction to automatically support the study of distributed collectives [18].

6. ACKNOWLEDGMENTS

The authors thank Gabriel Ripoché for his contributions to this project as well as the constructive comments from the anonymous reviewers. This material is based upon work supported by the National Science Foundation ITR (Digital Society and Technologies) program under Grant No. 0205346. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

- [1] Carstensen, P. H., Sørensen, C., Tuikka, T. (1995). Let's talk about bugs! *Scandinavian Journal of Information Systems*, 7(1), 33-54.
- [2] Crowston, K. (1997). A coordination theory approach to organizational process design. *Organization Science*, 8(2), 157-175.
- [3] Mozilla.org. (2005). *Life Cycle of a Bug*. Available: <http://www.mozilla.org/bugs/> [2005, May 16].
- [4] Egger, E., & Wagner, I. (1992). Time-management: a case for CSCW. *CSCW '92. Proceedings of the Conference on Computer-Supported Cooperative Work (Toronto, Canada, October 31-November 4, 1992)*. New York: ACM Press, 249-256.
- [5] Gasser, L., & Ripoché, G. (2003). Distributed collective practices and free/open-source software problem management: perspectives and methods. *2003 Conference on Cooperation, Innovation & Technologie (CITE'03)*

(Université de Technologie de Troyes, France, December 3-4, 2003).

- [6] Gerson, E., & Star, S. L. (1986). Analyzing due process in the workplace. *ACM Transactions on Information Systems*, 4(3), 257-270.
- [7] Grinter, R.E. (1995). Using a configuration management tool to coordinate software development. *Proceedings of conference on organizational computing systems, 1995, Milpitas, CA*, 168-177.
- [8] Herbsleb, J.D. & Grinter, R.E. (1999). Splitting the organization and integrating the code: Conway's law revisited. *International Conference on Software Engineering, 1999, Los Angeles, CA*, 85-95.
- [9] Herbsleb, J.D., Mockus, A., Finholt, T.A., & Grinter, R.E. (2000). Distance, dependencies, and delay in a global collaboration. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (Philadelphia, PA, December 1-6, 2000)*. New York: ACM Press, 319-328.
- [10] Humphreys, W. (2004). *The Software Quality Profile*. Available: <http://www.sei.cmu.edu/publications/articles/quality-profile/index.html>; accessed [2004, September 12].
- [11] Larsson, A. (2003). Making sense of collaboration: the challenge of thinking together in global design teams. *GROUP '03. Proceedings of the 2003 International ACM SIG-GROUP Conference on Supporting Group Work (Sanibel Island, Florida, USA, November 9-12, 2003)*. New York: ACM Press, 153-160.
- [12] Leveson, N. & Turner, C.S. (1993). An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7), 18-41.
- [13] National Institute of Standards & Technology. (2002). *The economic impacts of inadequate infrastructure for software testing: final report. May 2002. Planning report 02-3*. Gaithersburg, MD: NIST.
- [14] National Transportation Safety Board. (2000). *Aircraft accident report: Controlled flight into terrain, Korean Air flight 801*. Washington, D.C.: NTSB. Available: <http://www.nts.gov/bpulicn/2000/AAR0001.pdf>; accessed [2004, March 11].
- [15] Powell, R.R. (1991). *Basic research methods for librarians*. (2nd ed.). Norwood, NJ: Ablex.
- [16] Raymond, E. (1999). *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. Sebastapol, CA: O'Reilly.
- [17] Ripoche, G. & Gasser, L. (2003). Scalable automatic extraction of process models for understanding F/OSS bug repair. *2003 Conference on Software & Systems Engineering and their Applications (ICSSEA'03) (CNAM, Paris, France, December 2-4, 2003)*.
- [18] Ripoche, G. & Sansonnet, J-P. (forthcoming). Experiences in automating the analysis of linguistic interactions for the study of distributed collectives. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*.
- [19] Sandusky, R.J., Gasser, L., & Ripoche, G. (2004). Bug report networks: varieties, strategies, and impacts in a F/OSS development community. *MSR 2004: International Workshop on Mining Software Repositories, ICSE 2004, IEEE International Conference on Software Engineering (Edinburgh, Scotland, UK, May 25, 2004)*.
- [20] Schmidt, K. & Simone, C. (1996). Coordination mechanisms: towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 5(2-3), 155-200.
- [21] De Souza, C.R.B., Redmiles, D., & Dourish, P. (2003). "Breaking the code", moving between private and public work in collaborative software development. *GROUP '03. Proceedings of the 2003 International ACM SIG-GROUP Conference on Supporting Group Work (Sanibel Island, Florida, USA, November 9-12, 2003)*. New York: ACM Press, 105-114.
- [22] Strauss, A. (1978). *Negotiations: varieties, contexts, processes, and social order*. San Francisco: Jossey-Bass.
- [23] Strauss, A., & Corbin, J. (1990). *Basics of qualitative research: grounded theory procedures and techniques*. Newbury Park, CA: Sage.
- [24] Yamauchi, Y., Yokozawa, M., Shinohara, T., & Ishida, T. (2000). Collaboration with lean media: how open-source software succeeds. *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work (Philadelphia, PA, December 1-6, 2000)*. New York: ACM Press, 329-338.
- [25] Weber, R. P. (1990). *Basic content analysis*. (2nd ed.). Newbury Park, CA: Sage.
- [26] Weick, K. E. (1995). *Sensemaking in organizations*. Thousand Oaks, CA: Sage.