How Negotiation Shapes Coordination in Distributed Software Problem Management

Robert J. Sandusky Les Gasser Gabriel Ripoche
Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
{sandusky,gasser,gripoche}@uiuc.edu

ABSTRACT

All software exhibits operational problems as well as opportunities for redesign. How software projects handle problems and ongoing redesign efforts – Software Problem Management or "SWPM" – is not well understood, especially in open distributed communities such as Free/Open Source Software (F/OSS) projects. Little is known about the factors that influence performance or outcomes (e.g. time, cost, quality, sustainability) of software development communities involved in SWPM. Publicly accessible repositories of SWPM data, such as bug report repositories maintained by F/OSS development communities, provide vast stores of data about distributed SWPM practices and knowledge. Individual bug reports (BRs) within these repositories are texts that provide detailed records of community activities including responses to, and analyses of, problems. By examining these records we can develop detailed pictures of how many collaborative, distributed, computer-mediated SWPM processes unfold, creating finer-grained accounts of how coordination actually works, and extending current theories of computer-supported collaboration. This paper reports on several ways in which activity and information interact in SWPM. We have identified a number of "basic social processes" (BSPs) that shape how participants organize and use information, build and maintain community knowledge, and coordinate community activities at a very large scale (e.g., several gigabytes of online information, tens of thousands of reporters, over 50,000 simultaneously open problems). We focus on how one BSP, negotiation, shapes coordination activities in SWPM. Using detailed qualitative methods, we report on the varieties and frequencies of negotiation practices, and analyze how different types of negotiation in different situations influence what knowledge is available to participants, how collective activity gets focused in specific directions, and how the organization of information shapes activity. Overall, this provides a more detailed understanding of key collaborative distributed coordination mechanisms than has been available in the CSCW literature to date.

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 [6]. 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 [11] (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.

In addition to the costs of finding and fixing them, 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 [10] and again, in a separate situation, in 2001 [5]. False warnings generated by safe altitude warning system software led air traffic controllers to drastically reduce the range of the system, from 54 miles to 1 mile. Controllers were then unable to warn the pilot of KAL flight 801 that the plane was about to crash into a mountain. 225 people died in the accident on Guam [5, 12].

Software is ubiquitous: it is in our homes, our automobiles, infrastructure (power, transportation, and communications networks), and medical instruments. 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

[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." [11] (p. ES-3).

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.

Until now, little focused research has been done to understand the socio-technical process of managing software problems. 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 one large, thriving F/OSS development community as our primary data.

Background information on bug reports, SWPM, distributed collective practices, and negotiation 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 implications of the findings and directions for future work are discussed.

1.1 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 and ongoing software redesign. A bug report contains (see Appendix):

- 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 [15]

A bug report does more than hold facts or information. A bug report also has tracking, interpretive, management, and sentimental functions and is constitutive of the community. 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* [16], 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 [8].

The bug report is also a boundary object [17]: it is a single coordination mechanism that can be used and understood by people playing various roles within the community (end-user as reporter; software expert as assignee; quality assurance specialist as QA Contact) while each of those actors may have a different understanding of the bug report and what it represents. For example, the reporter may describe a detected misalignment in non-technical terms while the assignee describes the underlying system behavior in terms of events, messages, classes, and methods.

The bug report comments and the bug activity data also form a representation of a bug report's trajectory – a path, progression, or line of development resembling a physical trajectory – from the time the bug report is created until it is set to a terminal state (e.g., "status = verified"). But the bug report also reveals information about how the work of managing software problems is aligned and coordinated; who participates in that process; how the organization of the community affects and is affected by the information contained in the bug report.

Also crucial in understanding SWPMs 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 in repository tools. (For example, many bug reports may contribute to the understanding or resolution of a single bug [7, 15].) The distributed, collective activity of SWPM is mediated in F/OSS communities by information and communication technologies (ICTs). Collective interpretations and actions (such as judgments on causality, management actions and code creation) are actively constituted, constructed, and actively coordinated via the social media of negotiation, and the traces of negotiation in bug reports serve as surrogates for interpretations, specifications, decisions, rationales etc.

Finally, empirical observation reveals that bug reports often represent entities other than bugs. For example, bug reports may represent requests for new features, bug report networks [15], collections of bugs or reports ("meta-bugs"), invalid problems (that is, issues that the community decides are not going to be worked on), duplicate reports of the same phenomenon, and even social gatherings and parties.

1.2 Software Problem Management

SWPM is one constituent part of the iterative use of either traditional or F/OSS development processes. Normatively, SWPM is a process comprising steps shown in Table 1 [1, 2, 3, 16]. This paper is focused on activity that usually occurs during step 3 (Bug Report Triage and Assignment) and step 4 (Expert Analysis, Fix Development, Testing and Deployment) of the normative SWPM process described in Table 1.

Table 1. SWPM Process

Normative Software Problem Management Process
1. Problem Identification Someone using the software notices an anomaly or mistake; identification can occur during normal use or during software testing; actors identifying problems may be testers [1, 2] or customers / users [2].
2. Problem Reporting Someone creates a bug report – a representation of the bug – to enable the bug and follow-up activities to be tracked through a bureaucratic process; actors creating the bug report may be testers [1] or intermediary such as a help desk on behalf of the customer or user [2].
3. Bug Report Triage and Assignment Someone takes action to begin the process of identifying and fixing the bug. An attempt to recreate the problem is made. The priority of the bug is assessed (triage); the proper person or organization to work on the bug is identified (assignment). In Carstensen, Sørensen & Tuikka's study [1], priority and assignment are handled by a group of software designers whose primary role is the management of software problems. In Crowston's study [2], set in a larger organization, problem recreation and prioritization is done by "marketing engineers" and assignment by "programming managers."
4. Expert Analysis, Fix Development, Testing and Deployment The assignee investigates the bug, determines its cause, evaluates repair options, consults other experts, coordinates the work of multiple experts, etc. The assignee modifies the software to resolve the bug and tests the modification. The modification, or fix, is deployed if testing is successful. This is performed by software designers [1] or software engineers [2].
5. Fix Verification Someone other than the assignee, playing a quality assurance role, verifies that the bug has been corrected. This is performed by the platform master (the person responsible for the software build containing the fix) [1] or the integration team [2].
6. Problem Closure Someone marks the bug report "closed:" the process is complete. In Carstensen, Sørensen & Tuikka [1], this is the responsibility of the central file manager; in Crowston [2], this is the software engineer's responsibility.

Beyond being a workflow, we see SWPM as a sensemaking process [21]: a 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 framework, is dynamically created by the community, and used to marshal, organize, and interpret information, as well as shape and coordinate subsequent sensemaking activity.

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 [22] (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. We compared the frequency of "product" values in the sample to the frequency of the same characteristic in the population and concluded, based on that characteristic, that the sample is representative of the population. 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 [20]. Concepts, phenomena, and relationships between phenomena were inductively derived and refined as they emerged from the bug reports during data analysis using grounded theory [19]. 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. 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. As interesting constructs and relationships emerged from the data, additional bug reports were sometimes selected from the population, a part of the grounded theory approach known as theoretical sampling.

3. NEGOTIATION IN SOFTWARE PROBLEM MANAGEMENT

The F/OSS community studied here is engaged in "distributed collective practice" (DCP) [7]. 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 [18] is one process frequently employed by this DCP to resolve problems of uncertainty and complexity, coordinate activity (e.g., fix design, problem triage), and support sensemaking, using the bug report as its locus.

3.1 Related Research

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] talk in general terms about the role of negotiation in software problem management, 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 [16], 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 each bug report: each bug report presents its own unique circumstances, contexts, and sensemaking trajectory.

Egger and Wagner [4], in one of the most detailed examinations of negotiation in the CSCW literature to date, analyze negotiation of a scare 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' [18] theory of negotiation as a way 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 [9] 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.

3.2 Processes and Impacts of Negotiation

Negotiation is a core concept that has emerged from this analysis. Strauss [18] 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 play in SWPM within the F/OSS community. At the higher, structural level [18], 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 [16]. The normative SWPM process (Table 1), based upon practice in traditional software development organizations, is not questioned or subjected to negotiation. 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. As Crowston [2] points out, the coordination mechanism used to support similar processes can vary from organization to organization. Or, 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 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 [13] in contrast to traditional, bureaucratic software development organizations. While examination of these factors is beyond the scope of this paper, it is useful be 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.

The focus of community members directly involved in SWPM is on the resolution of particular software problems, and even more accurately, on the disposition of individual bug reports (remember that bug reports do not always represent software problems). New bug reports are created at a peak rate of about 140 per day (based upon the population sampled). The community seemingly distinguishes boundaries between what is and is not negotiable in order to successfully ingest and process this volume of bug reports. The coordination mechanism – the bug report and the associated protocol – are considered fixed parts of the structural context. The issues represented by individual bug reports are considered to be fluid and subject to negotiation.

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.

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)

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%

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 (number 40), one negotiator begins his comment with "What's wrong with you people?" In terms of duration and number of participants, we have found examples in our sample of 12 contributions from 5 people and 17 contributions from 4 people.

Community members negotiate about bug priority; bug resolution scheduling; code reviews; whether a bug report represents a bug, an intended feature or an enhancement; whether the bug's cause has been determined; how a bug report should be managed by the community; whether the bug's cause has been identified; whether the bug is fixed yet or not; whether a problem exists or not; what the scope of the bug is; what type of bug is it; who is responsible for fixing the bug.

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 common. 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.

The definition and selection of a correct design to fix a problem is the second most frequently negotiated issue, occurring in about one-third of the bug reports in this sample. These negotiations 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, below). However, some bug reports are explicit requests for enhancements (example 2), some of which may be larger in scale. Negotiation about design is often necessary because the ideas for fixing problems must be developed and expressed. There are at least two situations that call for explicit expression of design ideas: 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, usually the person assigned to fix the bug or the person reporting the bug, wants to express the approach she or he believes best addresses the bug in order to allow other community members to comment on the proposal.

Table 3. Topic frequency

Topic 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 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%

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 still consider these as examples of negotiation. 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 also, 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 4 different issues: What is best design?, Who's responsible for fixing this?, How should the BR be managed?, and Is this a duplicate 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 de-
sign?]
```

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 ex-
pand the collapsed toolbars then things
would be fine. However, this may seem odd
when comparing to other apps. [Turn 3:
What is best design?]
```

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 [compo-
nent] meta module in [system-
Name]/modules/staticmod if the DIS-
ABLE_[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 pack-
ages (i'm not sure if we have to use another
set of .jst files in xpinstall, or if the
.xpi files are created solely from the direc-
tories 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 [system-
Name] 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 proba-
bly 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, commetator_2, writes:

```
Commentator_2: minor enhancment re-
quest....when [systemName]_DLL is set,
[systemName].exe should link to [sub-
systemName].dll rather than statically
linking the same libraries.

That will make isolating bugs much eas-
ier by having a testbed [system-
Name].exe) loading the same [subsystem-
Name].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.

Design negotiation occurred here, but it seems to have failed in some senses. The design negotiation failed to reveal the scope of the effort either in lines of code, elapsed time or degree of effort. It did, however, bring the enhancement into the open and make it available for comment, which occurred in an open and constructive way [8]. 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 topics in the description area of the bug report (day 1):

```
Reporter: Behavior should be consistent and
subject to control by event handlers…. Ex-
pected 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 compo-
nent. <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 re-
spected. [Issue 3, turn 1: How should the BR
be managed?]
```

On day 4, a commentator begins the fourth negotiation instance on the topic Is this a duplicate 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 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. <commen-
tator from bug X>'s proposed spec would ad-
dress 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 ab-
sence of such a control, the form would not
be submitted. [Issue 4, turn 2: Is this a du-
plicate 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 mul-
tiple 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 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

4.1 Implications

Negotiation is a basic social process commonly used in a variety of contexts by this community to support SWPM (Tables 2 and 3). 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. Because our sample size is representative of the population (as discussed in section 2 above), 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. Negotiations can occur regarding whether a bug report represents an actual bug or a request for system behavior that is not, should not or will not be provided ("Is there a problem?"). Negotiations often arise as the community tries to understand whether the reported bug is actually fixed or not ("Is this problem fixed?"), which can relate back to differing expectations about what a correct, complete or sufficient fix is ("What is best design?"). There are several other issues that are subject to negotiation, including bug report management ("How should the bug report be managed?"), fix scheduling ("When should fix be scheduled?"), duplicate bug report identification ("Is this a duplicate bug report?") and who is responsible for fixing a bug ("Who's responsible for fixing this?").

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). In order to re-align system behavior with expectations, the community must come to agreement about (1) the actual desired behavior of the system and (2) what actions are necessary to effect re-alignment. In some bug reports, evidence of explicit negotiation doesn't exist: in these cases, what needs to be done somehow seems 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.

This work informs and extends existing theories of coordination by examining the nature of SWPM at a different, finer grained level of analysis, the individual bug report. Coordination theory is primarily to be used to analyze processes and identify ways to improve them [2]. Crowston analyzes how dependencies between activities are managed in the software change process, focusing on a higher, process-level view. However, Crowston presents no information about how activity and information are used to manage dependencies arising within individual cases of software change.

Schmidt and Simone [16] also focus on the process level by arguing that a combination of artifact and protocol together provide a coordination mechanism. The bug repository is the central coordination mechanism in this F/OSS development community. The repository supports SWPM by enacting an artifact (the bug report) and a protocol (standard, but flexible, sequences of activities). Schmidt and Simone's theory does not, however, predict or explain why negotiation is so prevalent in the field of work, within individual bug reports.

In Strauss' terms [18], social order is negotiated order. Strauss uses the term structural context to describe "that 'within which' the negotiations take place, in the largest sense" [18] (p. 98). What we see and are analyzing here, however, is negotiation applied 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 negotiation of both information and social order within each of these sub-communities. Negotiation at the bug report level is emergent, dynamic, and organic in the service of managing both bug reports (which may or may not represent problems) and the software problems that some bug reports represent. In contrast, the bug repository – this community's coordination mechanism – changes far more slowly and within the parameters of the normative SWPM process described in Table 1. Crowston's and Schmidt and Simone's theories can be applied to the analysis of structural context (e.g., coordination mechanisms), like a SWPM system, but they cannot be applied to the study of individual instances of negotiation.

We also conceive of SWPM as a sensemaking process [21]. 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. We see negotiation occurring in a majority of the bug reports in our sample. Negotiation is one of the basic social processes employed by the community to move toward consensus regarding issues of behavior, values, design, causes, and code, and thus to problem resolution.

Instances of negotiation in bug reports make many aspects of SWPM visible. 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). 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.

4.2 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, we are planning to test for the effects of the presence or absence of instances of negotiation on SWPM management outcomes. Two measures of outcome are (1) whether the bug report ever moves into a valid resolved state and (2) the elapsed time from the creation of the bug report to the time the bug report is resolved. Specifically, we can test for (1) the effect of the presence or absence of negotiation; (2) the effect of negotiation of specific issues; (3) the effect of the length and complexity of negotiation; (4) the effect of negotiation of combinations of issues.

We are also investigating the extent, nature, and scope of networks of bug reports [15]. We have not yet looked systematically at how negotiation is employed within bug report networks, although we have seen evidence (as in example 3, above) that individuals do make contributions to multiple bug reports within a bug report network.

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)?

We are also eager to investigate the generalizability of the results presented here by examining bug reports and SWPM in other settings, such as traditional software development organizations or in other F/OSS communities with different characteristics. We would also like to examine problem management practices in settings other than software development. Additional methods, like participant / observation and interviewing, could be applied and allow access to information not accessible using our present method.

The results of this qualitative investigation may also be used to support development of computational tools for supporting SWPM work. One example is a computationally based tool that can be used to recover information about design discussions and decisions from bug reports, based upon linguistic markers identified by qualitative examination of a large sample of bug reports [14]. Another tool example is a system to detect patterns of negotiation that are likely predictors of bug reports with long durations.

5. ACKNOWLEDGMENTS

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.

6. 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] Digital Library Production Service. *A Bug's Life Cycle*. Available: http://bugzilla.umdl.umich.edu/bug_status.html; accessed [2004, March 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] Gage, D., McCormick, J. (2004). Case 109 Panama's cancer institute. *Baseline*, March 2004, 32-ff. Available http://www.eweek.com/article2/0,1759,1543652,00.asp?kc=EWNWS030804DTX1K0000599; accessed [2004, March 10].

[6] Gasser, L. (2003). Software quality assurance and organizational processes. Project Memo UIUC-2003-09, 18 March 2003.

[7] Gasser, L., & Ripoche, 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).

[8] Gerson, E., & Star, S. L. (1986). Analyzing due process in the workplace. *ACM Transactions on Information Systems*, 4(3), 257-270.

[9] Larsson, A. (2003). Making sense of collaboration: the challenge of thinking together in global design teams. *GROUP '03. Proceedings of the 2003 International ACM SIGGROUP Conference on Supporting Group Work* (Sanibel Island, Florida, USA, November 9-12, 2003). New York: ACM Press, 153-160.

[10] Leveson, N. & Turner, C.S. (1993). An investigation of the Therac-25 accidents. *IEEE Computer*, 26(7), 18-41.

[11] 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.

[12] National Transportation Safety Board. (2000). *Aircraft accident report: Controlled flight into terrain, Korean Air flight 801*. Washington, D.C.: NTSB. Available: http://www.ntsb/gov/bpulictn/2000/AAR0001.pdf; accessed [2004, March 11].

[13] Raymond, E. (1999). *The cathedral and the bazaar: musings on Linux and open source by an accidental revolutionary*. Sebastapol, CA: O'Reilly.

[14] 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).

[15] Sandusky, R.J., Gasser, L., & Ripoche, G. (submitted for publication). Bug report networks: varieties, strategies, and impacts in a F/OSS development community. Submitted to *MSR 2004: International Workshop on Mining Software Repositories, ICSE 2004, IEEE International Conference on Software Engineering* (Edinburgh, Scotland, UK, May 25, 2004).

[16] 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.

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

[18] Strauss, A. (1978). *Negotiations: varieties, contexts, processes, and social order*. San Francisco: Jossey-Bass.

[19] Strauss, A., & Corbin, J. (1990). *Basics of qualitative research: grounded theory procedures and techniques*. Newbury Park, CA: Sage.

[20] Weber, R. P. (1990). *Basic content analysis*. (2nd ed.). Newbury Park, CA: Sage.

[21] Weick, K. E. (1995). *Sensemaking in organizations*. Thousand Oaks, CA: Sage.

[22] Powell, R.R. (1991). *Basic research methods for librarians*. (2nd ed.). Norwood, NJ: Ablex.

7. APPENDIX: SAMPLE BUG REPORT

This sample bug report---a very simple one drawn from the repository investigated in this project---is included to provide a sense of the kinds and varieties of information that a bug report typically contains. In this case, the reporter and the assignee are the same individual. Identities of individuals and some system-related names are disguised (usually framed by angle brackets: thus Microsoft Windows becomes <Commercial Operating System>).

```
Bug 140 - Paste is enabled for readonly textfields

Bug#: 140                        Product: <product_name>            Version: Trunk
Platform: All                    OS/Version: All                    Status: VERIFIED
Severity: normal                 Priority: --                       Resolution: FIXED
Assigned To: <Reporter/Assignee>   Reported By: <Reporter/Assignee>   QA Contact: <QAContact_1>
Component: Editor: Core          Target Milestone: ---                    URL:
Summary: Paste is enabled for readonly textfields
Keywords:
Status Whiteboard:
Opened: 2001-01-24

Description:

Paste is enabled in the context menu for readonly textfields (but it still
doesn't do anything).  Patch coming; <CC_1>, can you <perform second level code review>?

------- Additional Comment #1 From <Reporter/Assignee> 2001-01-24 21:22 -------
Created an attachment (id=23433)
[patch] disable paste if not modifiable

------- Additional Comment #2 From <CC_1> 2001-01-25 10:51 -------
<second level code review>=<CC_1>

------- Additional Comment #3 From <Commentator_A> 2001-01-25 12:34 -------
<first level code review>=<Commentator_A>

------- Additional Comment #4 From <Reporter/Assignee> 2001-01-25 13:36 -------
checked in.

        Status changed from New to Resolved
        QAContact changed from QAContact_1,QAContact_2 to QAContact_1
        Resolution changed from NULL to Fixed
```

```
------- Additional Comment #5 From <QAContact_1> 2001-02-07 13:43 -------
verified in 2/6 build.

       Status changed from Resolved to Verified
```