How Tagging helps bridge the Gap between Social and Technical Aspects in Software Development

Christoph Treude, Margaret-Anne Storey
Dept. of Computer Science, University of Victoria
ctreude@uvic.ca, mstorey@uvic.ca

Abstract

Empirical research on collaborative software development practices indicates that technical and social aspects of software development are often intertwined. The processes followed are tacit and constantly evolving, thus not all of them are amenable to formal tool support. In this paper, we explore how "tagging", a lightweight social computing mechanism, is used to bridge the gap between technical and social aspects of managing work items. We present the results from an empirical study on how tagging has been adopted and adapted over the past two years of a large project with 175 developers. Our research shows that the tagging mechanism was eagerly adopted by the team, and that it has become a significant part of many informal processes. Our findings indicate that lightweight informal tool support, prevalent in the social computing domain, may play an important role in improving team-based software development practices.

1. Introduction and Motivation

Software development processes are among the most complicated tasks performed by humans. In a typical software development process, developers perform several different activities: they use numerous tools to develop software artifacts ranging from source code and models to documentation and test scenarios, they use other tools to manage and coordinate their development work, and they spend a lot of time communicating with other members on their teams. Most tools used by software developers in their daily work are tailored towards individual developers and hardly support team work. However, software is rarely developed by individuals and the success of software projects largely depends on the effectiveness of communication and coordination within teams [22].

In recent years, the focus of tools for software developers has shifted towards team-aware tools that support communication and cooperation in one way or another. Among those tools, there are comprehensive development environments, such as IBM's Jazz [11], and tools that only focus on certain aspects, such as groupware (e.g. INCOME/STAR [26]). As these tools are brought into the mainstream, the tension of balancing support for formal engineering practices with the social informal aspects of a team becomes obvious. Indeed, a key finding from the Computer Supported Cooperative Work (CSCW) research community is that tools that ignore emergent work practices and social aspects of a tool's use frequently fail (for examples, see [16]). Thus, a challenge for the software engineering tool community is to develop tools that support both aspects.

Balancing formal and informal user needs is particularly important for task management in a socio-technical system. Tasks are important cogs in the development process machine that need to be carefully aligned with one another, both in what they achieve and in their timing. Since tasks crosscut both technical and social aspects of the development process, how they are managed will have a significant impact on the success of a project.

Software development environments typically have explicit tool support for managing tasks. For example, Jazz has tool support for managing "work items", where a work item is a generalized notion of a development task (see Fig. 1). Work items are assigned to developers, are classified using predefined categories, and may be associated with other work items. Jazz work items also have informal tool support to address social aspects. Specifically, Jazz supports a discussion thread and a lightweight "tagging" mechanism. Using this latter feature, developers can freely associate user defined keywords with work items.

In our earlier research, we have been investigating tool support for task management, paying particular attention to the social aspects of task management [32, 33, 34]. Here, we report the results from an empirical study on the practice of tagging work items within Jazz. In our case study, we examine how a software development team uses tags for work items, both for individual use and for collaboration. We gathered data through the inspection of the project repositories and by conducting interviews with developers

ICSE'09, May 16–24, 2009, Vancouver, Canada
978-1-4244-3452-7/09/$25.00 © 2009 IEEE

on the team. Our main contribution is the identification of different ways in which tags support informal processes in software development and thus fill the gap between social and technical aspects of artifacts. Furthermore, we examine how tagging was adapted by software developers to suit their needs and we identify potential tool enhancements.

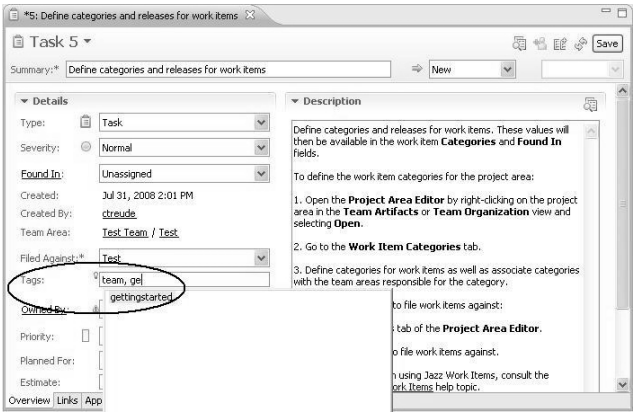

Figure 1. Work items in Jazz

The remainder of the paper is structured as follows. In Section 2, we discuss related work on informal processes in software development and provide background on tagging. Our research questions and methodology are presented in Section 3. Section 4 comprises the main part of this research and describes how tagging of work items has been adopted, how it supports informal processes, and how it differs from tagging of web content. Subsequently, Section 5 discusses the implications of our findings on tool design and the limitations of our study. Our work is concluded in Section 6.

2. Background and Related Work

Work related to our research can be divided into two main areas: research on the social aspects of software development, and research on tagging and its adoption in software engineering. Our work can be interpreted as the intersection of these two areas: using tags to support social aspects in software development.

2.1. Social Aspects in Software Development

Software development is recognized to be one of the most challenging management tasks performed by humans [22]. The larger systems get and the more complicated the compositions of the developing teams are, the more obstacles there are on the way to the release of a software system. Since most software systems are developed by teams, effective coordination and communication are crucial to the success of software projects.

There are at least three strands of research that have considered the impact of social aspects in software development: global software development, open source development and knowledge management. Researchers of these topics recognize that software development processes are more than writing source code, and that "articulation work" [24] must be supported in a software engineering project. According to Gerson and Star [13]: *"Articulation consists of all tasks needed to coordinate a particular task, including scheduling sub tasks, recovering from errors, and assembling resources."* Other examples of articulation work include discussions about design decisions, assigning bug fixing tasks to developers and deciding on interfaces.

Challenges that arise in global software development include dealing with strategic and cultural issues [20], longer development times when coordinating with remote colleagues [19], dealing with communication breakdowns such as unclear dependencies, circular dependencies and schedule changes [5], and managing plan failures [30]. In open source projects, the motivation that keeps a project going is not only of a technical nature, but may also be organizational or ritual [25]. In distributed projects, managing implicit knowledge [23], maintaining awareness [17] and leveraging expertise [10] can also impact the success of a project. These many challenges that arise in team-based software development can be addressed by better awareness tools and processes, improved communication practices, implicit and explicit knowledge management, as well as support for articulation work.

A key result that has an implication when designing improved tools or processes, is that technical artifacts are often intertwined and overloaded with social artifacts during a development project. For example, de Souza *et al.* [7] claim that source code is both a social and a technical artifact and that dependencies do not only exist between artifacts but also between developers. In a previous study on source code annotations, Storey *et al.* [34] report on how annotations are used to document both technical and articulation activities. Grinter [15] also describes how configuration management tools are sometimes co-opted for articulation work, despite the fact that they have significant shortcomings in supporting articulation work. She notes insufficient support for individual developers and teams, and reports challenges from a lack of representation of the work itself leading to inappropriate built-in assumptions about the work flow.

Several researchers have studied how teams use issue tracking systems[1] for managing articulation work. Many of these studies focus on mining and analyzing quantitative data to reveal information about the evolution of the system [21] or to predict future behaviours [2, 27]. Ellis *et al.* [9] report results from an interview of how developers use Bugzilla, a popular bug tracking system. The motiva-

[1] Such systems are also referred to as defect or bug tracking systems.

tion for their study was the design of a visualization tool for tasks. One of their key findings was that Bugzilla played a key role in managing the project. Sandusky *et al.* conducted a qualitative analysis of an open source bug repository to describe how negotiation plays a role in coordination activities [31]. Bettenburg *et al.* also report a study to evaluate the effectiveness of bug reports [3].

Although researchers have considered how bug repositories and issue tracking systems are used for coordinating work, researchers have not thus far considered how tagging can be used to support informal activities by a team coordinating tasks. De Souza *et al.* [8] conducted an ethnography with a software development team and found that tools often create a distinction between private and public aspects of development. To close this gap, several informal practices are adopted in order to manage interdependencies between both perspectives. Similarly through this paper, we wish to consider how tagging is used to bridge the gap between the technical and social aspects of work item management. But first we review related research on tagging, specifically how tagging is currently used in software development.

2.2. Tagging and Software Development

The concept of tagging, as it is currently used, comes from the social computing domain. Social computing technologies, sometimes referred to as Web 2.0, are seeing rapid adoption by emergent communities on the web. Key examples include Facebook (facebook.com), YouTube (youtube.com) as well as community based recommender systems such as CiteULike (citeulike.org), TripAdvisor (tripadvisor.com) and flickr (flickr.com). Tagging is used by many of these systems and is often referred to as social bookmarking. The success of tags is closely related to their bottom-up nature: tags do not have to be pre-defined, every user can choose their own tags, and the number of tags per item is arbitrary. Based on these characteristics, tags are used to classify items in an informal way, and they stand in contrast to formal top-down classification mechanisms.

Golder *et al.* [14] and Hammond *et al.* [18] provide overviews of tagging systems and classify the main reasons for user tagging. A more detailed study was done with the photo sharing website flickr [1]. A common finding across these studies is that users tag to provide information on an artifact (e.g. what an artifact is or to refine a category) and for organizing artifacts. Part of the success of tagging comes from allowing users to define their own vocabulary [12]. Information retrieval is also enhanced by community tagging.

The introduction of tags into software development raises the question of how the informality of tagging affects the process of developing software and how a typical software development process can take advantage of the characteristics of tags. Tagging is not a new concept to software engineering. Tags have been used for decades for annotating check-in and branching events in software version control systems, as well as for documenting bugs in bug tracking systems. Also, Brothers' ICICLE was an early exploration of tag-like structures with a limited, controlled vocabulary during code inspection [4]. Many early uses of the word tagging in software engineering systems relied on a preexisting controlled vocabulary, thus this earlier form of tagging is not consistent with the social computing notion of tagging today. Due to these inconsistencies on the term tagging, we define a tag as follows: *A tag is a freely-chosen keyword or term that is associated with or assigned to a piece of information. In the context of software development, tags are used to describe resources such as source files or test cases in order to support the process of finding these resources.*

Tagging, as defined here, has not been extensively researched in a software engineering context. Some systems support social bookmarking, for example Code Snippets (bigbold.com/snippets) and ByteMycode (bytemycode.com). They support social tagging of source code, but require the user to post code fragments on public servers before tags can be applied. A recent tool that intersects social tagging with software development is described by Storey *et al.* [32]. Their tool TagSEA (Tags for Software Engineering Activities) is a collaborative tool to support asynchronous software development and uses the ideas of social tagging to support coordination and communication. A case study [33] showed that TagSEA provides the user-defined navigation structures that are lacking in traditional task annotations. Opposite to this bottom-up approach, the tools Concern Graphs [28] and ConcernMapper [29] enable developers to associate parts of source code with high level concerns.

Apart from these studies, there is little research on how the lightweight mechanism of tagging can play a role in supporting informal activities in software development. The research described in this paper examines the current use of tags for task management in a software development project with the aim to identify potential tool enhancements. For a more general discussion of lightweight tool support for work activities, we refer to work by Churchill and Bly [6].

3. Research Questions and Methodology

This section identifies our research questions, the setting of the study, and our data collection methods.

3.1. Research Questions

The research questions below encompass the main themes of our research: the adoption of tags by software

developers, the role of tagging in team-based software development, and the adaptation of the tagging feature.

1. How is the social tagging mechanism adopted by developers for annotating work items?
   (a) How does the frequency of tag use vary over the lifetime of the project?
   (b) How many different tags are used, and which tags are used more frequently?
   (c) How many users tag and how does this number of users vary over time?
2. What role does the tagging feature play in the work practices of individuals and team developers?
   (a) Why do developers tag work items?
   (b) How do developers use tags for work items?
   (c) Which role do tags play in collaboration?
3. How does work item tagging compare to social tagging on the Internet?
   (a) What are the differences between work items and web content?
   (b) How are tags adapted to meet the needs of software developers?

3.2. Methodology

In the following paragraphs, we outline the setting of our research as well as the two data collection methods we used: inspection of archival data available in repositories and semi-structured interviews with software developers.

3.2.1. Research setting. Our study took place with a professional development team from IBM. The team consists of approximately 175 contributors and about 30 functional teams, with some teams acting as sub-teams of larger teams and some contributors assigned to multiple teams. The team members are located at 15 locations worldwide, primarily in North America and Europe. The developers of the team use IBM's Jazz to develop software and they follow the "Eclipse Way" development process [11]. This process, developed by the Eclipse Development Team, is an agile, iteration-based process with a focus on consistent, on-time delivery of quality software through continuous integration, testing, milestones, and incremental planning.

Developers using Jazz organize their work around so-called work items which can be interpreted as development tasks. A typical work item consists of a unique number, summary, description, state, work item type, severity and priority, the component it was filed against, the version it was found in, the creator, and several other details that are optional. As can be seen in Figure 1, there is an optional tag field in which developers can insert an arbitrary number of tags per work item. The Jazz content assistant suggests tags with a common prefix that have been used before. If a developer adds a tag that has not been used before, a popup window appears and asks if this tag should be added to the vocabulary.

3.2.2. Data collection. Our methodology follows a mixed method approach, collecting both quantitative and qualitative data. In order to gather quantitative data on the use of tags in the project, we accessed the repositories of the development team and extracted all relevant information. The amounts of data extracted for the time period from May 2006 to April 2008 are shown in Table 1. The data covers the complete development process up to the 1.0 release of the team's product including six milestone and three beta releases. The team were working on the 1.1 release at the time of our data collection.

Table 1. Data extracted from repository

data object	amount
Work items	37,590
Number of tags applied to work items	13,902
Number of tags removed from work items	1,200

Qualitative data was collected through a series of interviews with developers. All interviews were semi-structured allowing for follow-up questions and clarifications. Most of the questions were aimed at understanding the details of why and how developers use tags. In total, four interviews were conducted. Three participants worked on the same component, one participant worked on a different one. The interviews were conducted in person at an IBM location and lasted about 30 minutes each. Three participants had been members of the team for about two years and thus experienced the introduction of the tagging feature two years ago. The other participant joined the team about 10 months ago.

The quantitative nature of our repository analysis and the qualitative nature of the interviews provided insights for all our previously posed research questions. In particular, the data inspection was used to help answer question 1. This data was also used during the design of the subsequent interviews. The interview data were used to answer the remaining research questions.

4 Findings

This section presents the findings from our case study. The research questions presented in the previous section are addressed in turn by the following subsections.

4.1. Adoption of Tagging

To answer our first research question, we performed an analysis of tag usage over time, looking at both the number of tags that are applied to work items and the number of individuals tagging work items. We note that both of these metrics increased substantially over the two year period of our study. The details are given below.

4.1.1. Frequency of tag use. Figure 2 shows how the number of tags added per day evolves over time from May 2006 to April 2008. The grey line depicts the actual numbers per day, the black line gives the value averaged over the last 30 days at any point of time. Apart from the initial import in June 2006, the number of tag uses per day is increasing over time. The peaks in the grey line mark extensive use of lifecycle related tags around beta releases of the product.

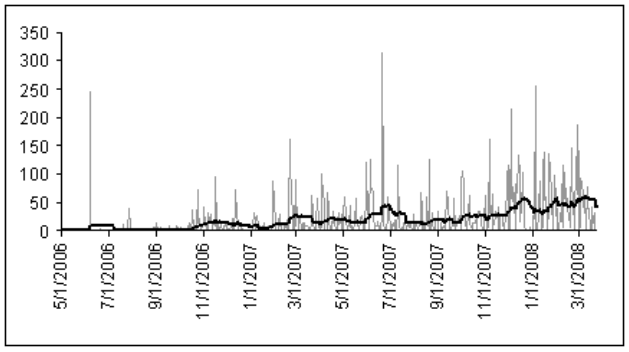

Figure 2. Tag uses per day

4.1.2. Most frequently used tags. About one fourth of all work items have been tagged at least once. The distribution of tags is shown in Figure 3. 682 different keywords have been used. Table 2 shows the ten most frequently used tags.

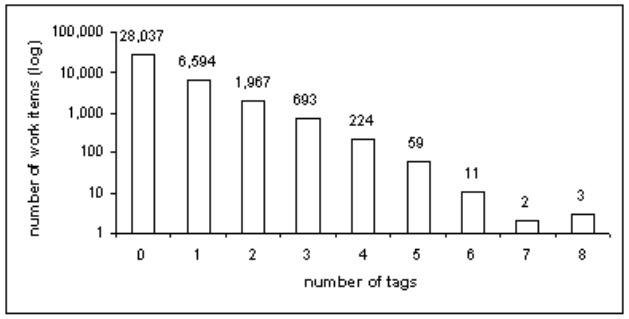

Figure 3. Distribution of tags (log scale)

4.1.3. Number of tag users. Similarly to the number of tag uses, the number of tag users also increases over time.

Table 2. Most used tags

tag	# uses
polish	724
testing	495
ux	483
svt	396
errorhandling	313
beta2candidate	306
usability	301
editor	286
performance	257
m6candidate	236

As shown in Figure 4, there are peaks of up to more than 30 different individuals applying tags on the same day and the only major discontinuities in distinct users per day occur around the Christmas holidays.

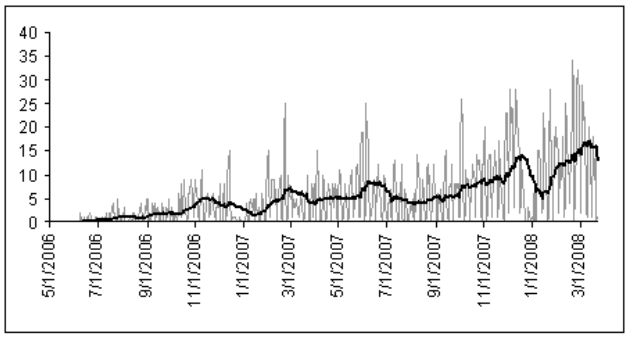

Figure 4. Distinct tag users per day

Out of 178 contributors who owned work items in the last two years, 112 applied at least one tag to a work item. In addition, the project has a web portal that allows clients to submit new work items. There were 45 individuals from outside the company applying tags through this web portal. However, the main tag users were team members from inside IBM. The top 25 most prolific taggers applied between 180 and up to more than 2,500 tags, using about 100 different keywords.

These statistics indicate that tags were used continuously after their initial introduction and that software developers found them helpful enough to keep using them over a period of two years. More details on tag usage in support of informal processes and collaboration are given below.

4.2. Tagging supports Informal Processes

In this section, we discuss the findings on our second research question: why software developers tag work items

and what role tags play in the work practices of individuals and teams.

4.2.1. Reasons for tag use. The predominant reason for the use of work item tags is categorization. As one of our participants put it: *"Mainly as a kind of* **categorization.** *[...] Tags are useful for identifying cross-cutting concerns like performance or accessibility or scalability or responsiveness, things like that, or testing."*

While the Jazz interface already provides an opportunity to categorize work items (*Filed Against* field in Figure 1), tags are more flexible. The category tree can be altered, but this would change the available categories for the whole team and is still only one-dimensional and thus not appropriate for cross-cutting concerns. One of our participants identified this disadvantage of the top-down classification: *"The problem is its very administrative side feeling, which is fine, except it's not as flexible to just* ad hoc *make things."*

Tags are also seen as a way to organize work items: *"[I use tags] because I feel like [work items] should be* **organized.** *I feel like they're there and so I should use them. [...] I don't know if they do organize work items, but it makes me feel like I'm doing something when I associate a tag with it. In theory, I'd like to believe that tags draw work items that have a similar area together. That's my hope."*

The main use case for tags is **finding** work items later: *"I use [tags] to categorize things basically, so I can have queries and find things. I'm afraid of losing work items if they aren't [tagged]."*

4.2.2. Individual tag use. Even though tags are seen as being helpful, especially due to their *ad hoc* and cross-cutting characteristics, the use of tags to search for work items is limited and not always part of the daily routine. However, the following quotes give exemplary scenarios in which tags have been used to find work items:

"During the polish phase of [a release], we had two weeks to polish. It's like, what polish things do we do? So we had tags like polish and usability, and I use that to kind of guide what work items we could work on."

"I used them the other day, trying to search for [...] a list of bugs against [a related product]. Thinking that, you know, I was probably a good boy and had tagged any [product] related issues with the [product] tag, I did a search for [product] tags and that actually found very few. [Laughs] 'Cause I hadn't been a very good boy and tagging my Jazz work items with the [product] tag."

It should be noted that tagging behaviour differs from developer to developer, based on their individual preferences. In addition, we found that planning-related tags are more likely to be created by team leads.

Another frequent use case emerges from design decisions regarding the structure of work items in Jazz. For example, work items do not have an *Operating System* field. Thus, tags like *linux* or *windows* are used to mark work items that are specific to an operating system.

Most tags are created soon after the corresponding work items: About half of the tags (50.2%) were created on the same day as the work item was created; and 75.6% of all tags were created at most 30 days after the work item.

4.2.3. Tags and collaboration. Tags play an important role in collaboration. In almost half of all the cases of applying a tag to a work item, the developer who applied the tag is not the one who created the work item, and in only less than one third of all cases, the developer applying the tag owns the work item. In other words, most tags are created for work items owned by somebody else.

Often, developers tag work items that are relevant to the work in their sub team: *"Most of the tags I mentioned before, like workspaceeditor, aspects, are tags that I would expect other people in the team to assign if they find issues in that area."*

In that context, tags are applied to help other developers on the same team with the management of their work and to make them aware of particular incoming work items. If not meant for other developers directly, tags are usually at least helpful for other developers indirectly. In our interviews, we did not find a need for private tags: *"I haven't really had much need to introduce sort of personal tags, usually between a combination of the tags to describe the area as well as the cross-cutting tags like performance etc. I can usually get by just with those."*

However, tags are rarely used for direct communication. Usually, the developer who applies a tag does not expect other developers to take actions based on a new tag: *"It's more for querying later on – I don't expect people to act on the tags that I put in."*

In terms of the tag vocabulary, the number of keywords that are applied by just one developer is 329, which is almost half of all keywords. The remaining 353 keywords have been applied by at least two different developers. Table 3 shows the ten most shared tags.

The fact that 353 keywords have been applied by more than one developer raises the question of how this vocabulary is formed and if there are any naming conventions among the developers to ensure a consistent vocabulary. The basic rules for choosing keywords for tags are captured by the following quote: *"I make sure I don't use too many acronyms, so that people can understand what the tag means. Apart from that, I try to make it somewhat descriptive, so [...] I wouldn't use perf, instead use performance."*

Official naming conventions are very rare and if they occur, they are usually mentioned in one of the team's internal wiki entries. Most of the conventions are implicit and happen through the work item feeds that are built into Jazz:

Table 3. Most frequently shared tags

tag	distinct users
performance	38
errorhandling	36
beta2candidate	35
usability	33
polish	32
ux	29
m5candidate	25
beta1candidate	23
adoption	21
review	19

"You just sort of see it happening in the work item's feeds." Using these feeds, developers see which tags are applied by their team members and are thus aware of the introduction of new keywords. Before introducing new keywords, most developers make sure that a similar keyword does not exist yet or refrain from adding new keywords to the vocabulary completely. The implicit dissemination of new keywords through the work item feeds results in a mostly consistent vocabulary that includes only a small number of synonymous keywords.

4.3. Work Item Tagging vs. Tagging on the Internet

While tagging originated in the social computing domain, it is being adapted to suit the daily needs of software developers for work items. This section addresses our third research question by highlighting the differences between tagging of work items and tagging of web content, and by describing the different kinds of tags that have emerged in the software engineering domain.

4.3.1. Differences. We noted the following differences between work items and web content in the social computing domain, e.g. on flickr:

- The properties of work items change significantly within hours or even minutes whereas pictures uploaded on flickr do not change after their initial upload.
- Work items have a firm expiry time as when the work item is closed it is usually no longer tagged. We found that 92% of all tags were assigned while the corresponding work item was still open. Moreover, we noted that some tags that were associated with specific milestones were no longer used once the relevant milestone passed by (see Section 4.3.2).

These differences have implications for the way developers tag work items and what can be inferred from tags. For one, idiosyncratic keywords can be used for tags that only have a short life time. There is no explicit need to understand those keywords after the period of interest. Since tags are not required for work items, the use of the tag field is a lot more informal and flexible than the use of other, more rigid properties of work items. This enables developers to use tags for their own benefits without having to worry about tool-related consequences.

The ephemerality of work items and process cycles leads to differences in how tags for work items are used and how they should be interpreted. In our research, we were easily able to infer specific themes of tagging that are unique to software development work item management. These themes are described in the following section.

4.3.2. Development-specific tags. Both our quantitative and our qualitative analysis revealed that different types of tags have evolved over the duration of the project. These different types can be distinguished by their intentions, their use, their users, their time of use, and to a certain degree their naming conventions.

Based on our analysis, we discovered the existence of **lifecycle related tags**. These tags are used extensively only for a specific period of time as they are related to a milestone in the development process and usually have the name of this release in their name, e.g. *beta2candidate*. Figure 5 shows the time lines of all tags that have been applied at least 100 times during the two year period. The time lines show the first and last days that a particular tag was applied to work items. The tags in the figure are ordered by the length of their time line. The lifecycle related tags such as *rc2candidate* appear at the top of the figure as they have a relatively short time line which is bound to the specific milestone, in this case release candidate 2. Compared to the other kinds of tags, lifecycle related tags are transient.

The vocabulary that emerged from two years of tagging comprises 682 different tags. As can be seen in Figure 6, the rate at which new keywords are added to the vocabulary remains fairly constant over the two year period and averages approximately one new keyword per day. This is another indicator that new tags are frequently needed depending on the lifecycle of the software system, and that not all tags are introduced early on.

As mentioned before, the tag field is also used to refine the work item categories that Jazz already provides. Compared to the categories, **component-specific tags** can be introduced without any effort or official conventions: *"He could've made another heading for each of [the sub categories]. But, for some reason I guess, the tagging was probably more open. I guess when you go and modify something like the spec, something like that; it feels very administrative, where this tagging is supposed to be more fluid."* Component-specific tags are usually used to catego-

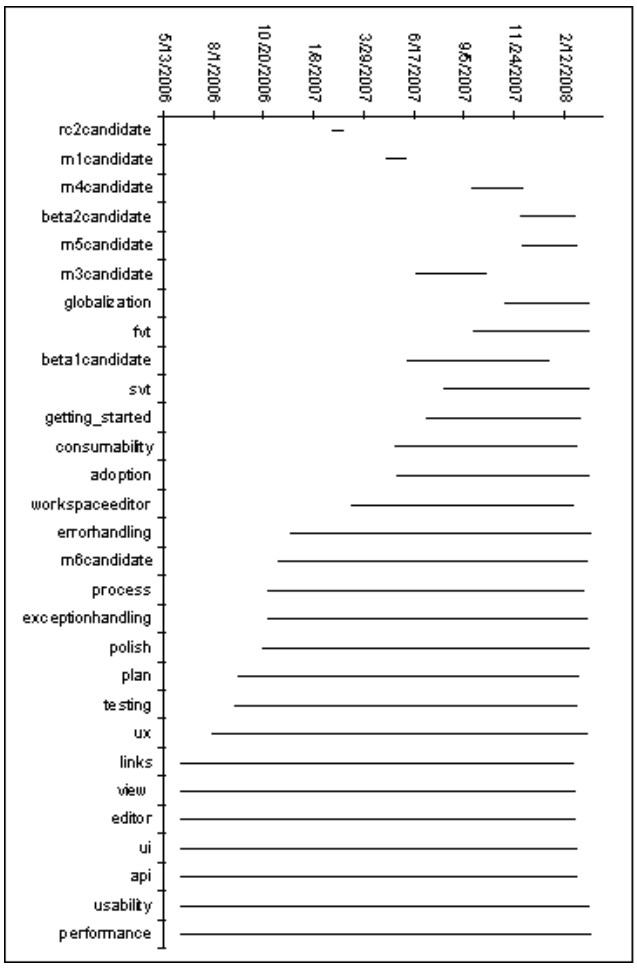

Figure 5. Time line lengths of most used tags

rize work items and their use depends largely on the presence of other categorization mechanisms.

Unlike component-specific tags, **cross-cutting tags** capture aspects of work items that cross cut the hierarchy of categories for work items: *"[Cross-cutting tags] are orthogonal to categories, they are – that's the beauty of tags, that they are cross-cutting. It's not about grouping, when we have grouping then things can only be in one."*

These cross-cutting tags frequently describe non-functional requirements such as performance, accessibility, scalability, or responsiveness, and thus such tags facilitate searching of cross-cutting concerns.

The last group of tags are **idiosyncratic tags**. These tags are neither related to a milestone, nor are they used to organize work items according to components or cross-cutting concerns. They are used for various reasons and are usually only used by very few developers. However, they support certain individual and collaborative processes. The following examples reveal a broad range of idiosyncratic tags:

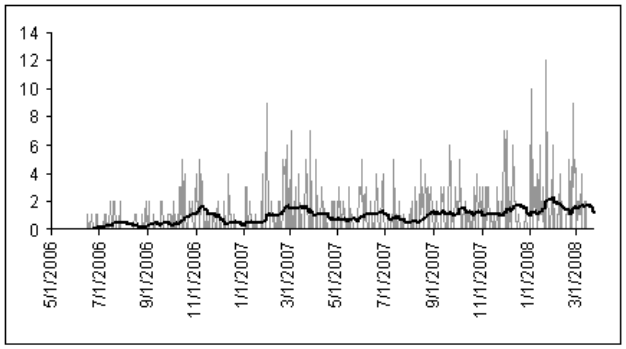

Figure 6. Addition of new tags to vocabulary

- **quickie**. This tag helps prioritizing a developer's task list: *"I've also been tagging things as quickie for things that should be fast and easy to fix."*
- **adoption**. This tag supports communication in between teams: *"Adoption means that there's a work item that's in our bucket or another bucket for which there's a change set attached that someone in another team has to adopt."*
- **included_in_faq**. Tags also help document the process for certain tasks: *"I went through a bunch of things that were tagged with faqable or faq or something like that, so then when I was done in order to see what I've done, I tagged it with included_in_faq."*
- **buildstatus**. Tags are used to flag work items: *"Buildstatus is flagging work items that I've created while I'm [release engineer] that have something to do with the current status of the build. So if it's broken and I'm complaining, I flag it with buildstatus."*

All of these examples describe processes for which formal tool support could be developed. However, since the processes do not occur often, tool support would add overhead to the system. For these kinds of processes, tags are frequently used as a way of informally adding meta information to work items.

Table 4 shows how the tags are distributed over the different types. The classification was done based on the tag names and verified through member checking during the interviews. Most of the distinct keywords used for tagging are component-specific. In contrast, there are comparably fewer distinct keywords for cross-cutting tags, but those keywords are used frequently to tag work items.

5. Discussion

One of the goals of our research is to contribute to the development of tool support for collaborative software devel-

Table 4. Classification of tags

type	tag keywords	tag instances
component-specific	413 (61%)	4,907 (35%)
lifecycle related	105 (15%)	2,505 (18%)
cross-cutting	88 (13%)	5,105 (37%)
idiosyncratic	76 (11%)	1,385 (10%)
sum	682	13,902

opment, especially with regard to tagging. In this section, we discuss how tool support for tagging could be broadened for other social and technical artifacts, and how such tool support could be improved. We also discuss the limitations of our research.

5.1. Implications on Tool Design

While tags have already been adopted by the software developers in our study, there are still areas in which tool support for tags can be improved. However, the eagerness with which tags have been adopted and the experiences of our interviewees suggest that the lightweight nature of tags has to remain intact: *"Tags are interesting in that, I think, part of their value is that they are loosely defined. I think if you start trying to give them stronger semantics, then they start to have a different flavour, more like categories."*

Therefore, enhancements of tool support should recognize the current benefits of tags and the main theme of any changes to tool support should be to help developers use tags. We suggest the following tool enhancements:

- Using the same lightweight approach as with tags for work items, a tag property could be added to other kinds of artifacts, especially source files, test cases, and requirement documents. Tags can also be implemented on a fine-grained level, e.g. for methods and fields. This would enable tagging across different types of content and thus would further support collaborative organization of artifacts. Similar ideas have been successfully tried and tested in TagSEA [32].

- Display tag authors along with the tags. During our interviews, we showed our interviewees a list of tags that were used on their work items but that they did not apply themselves. We discovered that our participants used the tag authors that we revealed to them to understand the tags. Adding the author information of tags is not obtrusive as the information is collected anyway and could just be displayed on mouse-over.

- Apart from author information, the only meta data property that should be added to tags is an optional description. For tags that do not have an obvious meaning such as *adoption* or *buildstatus*, a short description would increase the usefulness as there are keywords in the vocabulary that may be unfamiliar to some developers. When a new keyword is introduced to the vocabulary, a dialog could ask for an optional description instead of just notifying developers that they are about to introduce a new keyword.

- Current tools do not offer any management for tags on work items. Useful functions would be changing all tags with a particular keyword, e.g. to fix spelling mistakes. For synonymous keywords such as *doc* and *documentation*, folding would be beneficial.

- To increase understanding of how tags are used and which tags are suitable for work item search, a reporting mechanism for tags should be added. Information about tags, their authors, the corresponding work items, and the time of the tag creation is available in the system, but this information is not used yet by the work item tooling. In fact, most developers are unaware of the specific characteristics of their tagging behaviour, as well as the tagging behaviour of their colleagues.

- Once the tag vocabulary is analyzed, tags for incoming work items could be suggested. Strong candidates for suggestions are tags that have extensively been used in the near past such as lifecycle related tags and tags that have been applied to work items in the same category.

- The work item search interface in Jazz does not search tags by default yet. Tags should be included here to increase their value.

When tags were initially introduced in Jazz, several additional features were suggested by developers. However, over the time of more than two years now, developers adapted to tags the way they were implemented initially: *"In the beginning, I thought that we should do a lot, have private tags, have more meta data with them. In the hindsight, their simplicity is kind of interesting."*

Being used more and more, tags have developed into a valued property of work items that has its strength in its informality and its flexibility.

5.2. Limitations

As with any chosen research methodology, there are limitations with our choice of research methods. The first limitation of our study lies in the small number of interviewees. However, as it is the nature of a large software development project that developers do not have much time to spare and since for ethical reasons, we did not want the developers to feel coerced to participate, we were unable to recruit more participants. On the other hand, one of our researchers

spent two months on site frequently having informal discussions with developers regarding their use of tags and the answers in the interviews were mirrored in his observations. Also, the interviewees had different backgrounds, from team leads to relatively new members of the team.

When the team started on their project, the tagging feature for work items had not been introduced yet. This might have influenced the specific tagging behaviour. Also, given that the team was one of the first adopters of Jazz, their willingness to adopt new technologies might be above-average. However, we believe that our conclusions regarding how tagging supports informal processes also apply for less innovative teams.

With regard to component-specific tags, it should be noted that the work item category tree was significantly altered over the duration of the project. Since the use of component-specific tags for sub categories largely depends on which categories are already available, the strength of our conclusions regarding these tags is limited.

IBM's Jazz is still new and it is the first software development environment supporting tags for development tasks. Thus, we were only able to get data from one software project. As more projects adopt Jazz or other development environments adopt tagging, additional studies should be conducted to gain further insights into the use of tags in software development.

6. Conclusions and Future Work

The main contributions of this paper are the identification of the various ways in which tagging supports informal processes in software development as well as concrete suggestions for tool improvements.

While there are many formal processes in place for technical artifacts, managing social artifacts is only supported by informal processes if there is any process at all. Tool support is a lot easier to develop for formal processes, so that informal processes are usually carried out via communication mechanisms. In order to understand software development as a whole and in order to provide appropriate tool support, we have to understand both the technical and the social aspects of software development. Tags are one way to look at the informal side of software development in a team setting. Through understanding how developers use tags in their daily work, we can extend our knowledge on informal aspects of software development and furthermore understand how a social computing technology, such as tagging, is adapted by software developers.

Our research has shown how the social computing mechanism of tagging has been adopted and adapted by a large software development team. Not only is tagging used to support informal processes within the team, it has also been adapted to the specific needs of software developers. With lifecycle related tags, tags for sub categories and cross-cutting concerns, and idiosyncratic tags for processes that require meta data but are not formalized, different kinds of tags have emerged over the duration of a software project. The main advantages of using tags in software development are their flexibility and their lightweight, bottom-up nature. While fields such as operating system, milestone, level of effort, or cross-cutting concerns could be part of fixed schemata, this would add overhead. Tags add the same functionality without implying administrative changes.

With the shift to team-based software development and the corresponding increasing importance of articulation work, informal processes and communication mechanisms, social computing mechanisms such as tagging may play an important role beyond work items. They may be used to organize, manage, and categorize software artifacts in general in an informal and collaborative way. Future work lies in the examination of the benefits of social computing mechanisms in other areas of software development.

Collaborative tagging implies an underlying social structure. We are currently exploring which social networks emerge in software development between authors of work items, owners of work items, and tag authors. This will increase our understanding of team dynamics in software development and may ultimately result in better collaborative software development tool support.

7. Acknowledgements

We wish to thank the team that granted us access to their repositories and conducted interviews with us. This research is supported by a fellowship from IBM. We also appreciate comments from Lars Grammel, Thomas Maier, Nick Matthijssen, Peter C. Rigby, Adrian Schröter, and Nancy Songtaweesin that helped improve the paper.

References

[1] M. Ames and M. Naaman. Why we tag: motivations for annotation in mobile and online media. In *CHI '07: Proc. of the SIGCHI Conf. on Human factors in computing systems*, pages 971–980, New York, NY, USA, 2007. ACM.

[2] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE '06: Proc. of the 28th Intl. Conf. on Software Engineering*, pages 361–370, New York, NY, USA, 2006. ACM.

[3] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *SIGSOFT '08/FSE-16: Proc. of the 16th ACM SIGSOFT Intl. Symposium on Foundations of Software Engineering*, pages 308–318, New York, NY, USA, 2008. ACM.

[4] L. Brothers, V. Sembugamoorthy, and M. Muller. ICICLE: groupware for code inspection. In *CSCW '90: Proc. of the*

1990 ACM Conf. on Computer-supported cooperative work, pages 169–181, New York, NY, USA, 1990. ACM.

[5] M. Cataldo, M. Bass, J. D. Herbsleb, and L. Bass. On coordination mechanisms in global software development. In *ICGSE '07: Proc. of the Intl. Conf. on Global Software Engineering*, pages 71–80, Washington, DC, USA, 2007. IEEE Computer Society.

[6] E. F. Churchill and S. Bly. It's all in the words: supporting work activities with lightweight tools. In *GROUP '99: Proc. of the Intl. ACM SIGGROUP Conf. on Supporting group work*, pages 40–49, New York, NY, USA, 1999. ACM.

[7] C. de Souza, J. Froehlich, and P. Dourish. Seeking the source: Software source code as a social and technical artifact. In *GROUP '05: Proc. of the 2005 Intl. ACM SIGGROUP Conf. on Supporting group work*, pages 197–206, New York, NY, USA, 2005. ACM.

[8] C. R. B. de Souza, D. Redmiles, and P. Dourish. "Breaking the code", moving between private and public work in collaborative software development. In *GROUP '03: Proc. of the 2003 Intl. ACM SIGGROUP Conf. on Supporting group work*, pages 105–114, New York, NY, USA, 2003. ACM.

[9] J. B. Ellis, S. Wahid, C. Danis, and W. A. Kellogg. Task and social visualization in software development: evaluation of a prototype. In *CHI '07: Proc. of the SIGCHI Conf. on Human factors in computing systems*, pages 577–586, New York, NY, USA, 2007. ACM.

[10] S. Faraj and L. Sproull. Coordinating expertise in software development teams. *Management Science*, 46(12):1554–1568, 2000.

[11] R. Frost. Jazz and the Eclipse way of collaboration. *IEEE Software*, 24(6):114–117, 2007.

[12] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987.

[13] E. M. Gerson and S. L. Star. Analyzing due process in the workplace. *ACM Transactions on Information Systems*, 4(3):257–270, 1986.

[14] S. Golder and B. A. Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006.

[15] R. E. Grinter. Supporting articulation work using software configuration management systems. *Computer Supported Cooperative Work*, 5(4):447–465, 1996.

[16] J. Grudin. Groupware and social dynamics: eight challenges for developers. *Commun. ACM*, 37(1):92–105, 1994.

[17] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *CSCW '04: Proc. of the 2004 ACM Conf. on Computer supported cooperative work*, pages 72–81, New York, NY, USA, 2004. ACM.

[18] T. Hammond, T. Hannay, B. Lund, and J. Scott. Social bookmarking tools (I): A general review. *D-Lib*, 11(4), 2005.

[19] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter. An empirical study of global software development: Distance and speed. In *ICSE '01: Proc. of the 23rd Intl. Conf. on Software Engineering*, pages 81–90, Washington, DC, USA, 2001. IEEE Computer Society.

[20] J. D. Herbsleb and D. Moitra. Guest editors' introduction: Global software development. *IEEE Software*, 18(2):16–20, 2001.

[21] H. Kagdi, J. I. Maletic, and B. Sharif. Mining software repositories for traceability links. In *ICPC '07: Proc. of the 15th IEEE Intl. Conf. on Program Comprehension*, pages 145–154, Washington, DC, USA, 2007. IEEE Computer Society.

[22] R. E. Kraut and L. A. Streeter. Coordination in software development. *Commun. ACM*, 38(3):69–81, 1995.

[23] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining mental models: A study of developer work habits. In *ICSE '06: Proc. of the 28th Intl. Conf. on Software Engineering*, pages 492–501, New York, NY, USA, 2006. ACM.

[24] P. Mi and W. Scacchi. Modeling articulation work in software engineering processes. *Proc. of the First Intl. Conf. on the Software Process*, pages 188–201, 21-26 Oct 1991.

[25] E. Monteiro, T. Osterlie, K. Rolland, and E. Royrvik. Keeping it going: The everyday practices of open source software. Norwegian University of Science and Technology (NTNU), 2004.

[26] A. Oberweis, T. Wendel, and W. Stucky. Teamwork coordination in a distributed software development environment. In *GI Jahrestagung*, pages 423–429, 1994.

[27] T. J. Ostrand and E. J. Weyuker. A tool for mining defect-tracking systems to predict fault-prone files. In *Proc. of Intl. Workshop on Mining Software Repositories*, 2004.

[28] M. P. Robillard and G. C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *ICSE '02: Proc. of the 24th Intl. Conf. on Software Engineering*, pages 406–416, New York, NY, USA, 2002. ACM.

[29] M. P. Robillard and F. Weigand-Warr. Concernmapper: simple view-based separation of scattered concerns. In *eclipse '05: Proc. of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 65–69, New York, NY, USA, 2005. ACM.

[30] K. Rönkkö, Y. Dittrich, and D. Randall. When plans do not work out: How plans are used in software development projects. *Computer Supported Cooperative Work*, 14(5):433–468, 2005.

[31] R. J. Sandusky and L. Gasser. Negotiation and the coordination of information and activity in distributed software problem management. In *GROUP '05: Proc. of the 2005 Intl. ACM SIGGROUP Conf. on Supporting group work*, pages 187–196, New York, NY, USA, 2005. ACM.

[32] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby. Shared waypoints and social tagging to support collaboration in software development. In *CSCW '06: Proc. of the 2006 20th anniversary Conf. on Computer supported cooperative work*, pages 195–198, New York, NY, USA, 2006. ACM.

[33] M.-A. Storey, L.-T. Cheng, J. Singer, M. Muller, D. Myers, and J. Ryall. How programmers can turn comments into waypoints for code navigation. *ICSM 2007: Proc. of the 2007 IEEE Intl. Conf. on Software Maintenance*, pages 265–274, 2-5 Oct. 2007.

[34] M.-A. Storey, J. Ryall, R. I. Bull, D. Myers, and J. Singer. Todo or to bug: Exploring how task annotations play a role in the work practices of software developers. In *ICSE '08: Proc. of the 30th Intl. Conf. on Software Engineering*, Washington, DC, USA, 2008. IEEE Computer Society.