Awareness 2.0: Staying Aware of Projects, Developers and Tasks using Dashboards and Feeds

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

ABSTRACT

Software development teams need to maintain awareness of various different aspects ranging from overall project status and process bottlenecks to current tasks and incoming artifacts. Currently, there is a lack of theoretical foundations to guide tool selection and tool design to best support awareness tasks. In this paper, we explore how the combination of highly configurable project, team and contributor dashboards along with individual event feeds is used to accomplish extensive awareness. Our results stem from an empirical study of several large development teams, with a detailed study of a team of 150 developers and additional data from another four project teams. We present how dashboards become pivotal to task prioritization in critical project phases and how they stir competition while feeds are used for short term planning. Our findings indicate that the distinction between high-level and low-level awareness is often unclear and that integrated tooling could improve development practices.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*Integrated environments*

General Terms

Human Factors, Management

Keywords

Collaboration, Dashboards, Feeds, Web 2.0, Awareness

1. INTRODUCTION AND MOTIVATION

The success of software projects largely depends on the effectiveness of communication and coordination within teams [13]. As the complexity of software systems increases, maintaining awareness of the overall status of a project and gaining an understanding of current bottlenecks in the process becomes a challenge. Awareness tools that display information solely based on source code become insufficient as individual developers need to take on managerial tasks. Rather than source code alone, the status of the project arises from an aggregation of data on open and closed development tasks, successful and failed builds, delivered and pending changes, and successful and failed tests as well as evolutionary information.

In recent years, development environments are becoming more advanced with respect to awareness support. In particular, IBM's Jazz software development environment [10] includes feeds and dashboards that aggregate data to improve awareness of high-level and low-level aspects. However, with these advances comes the need to further our understanding of what the most appropriate toolset is [14]. Research on collaborative environments often draws on both Software Engineering and Computer Supported Cooperative Work (CSCW). Most of the related work on awareness tools for software development has focused on low-level code-specific tasks rather than higher levels of abstraction [23]. There is still a lack of understanding of how to achieve high-level awareness of project management issues with low-level awareness of more fine grained activities such as source code changes and development task creation.

In this paper, we report on an empirical study of several development teams and their use of dashboards and feeds in day-to-day development activities. These teams use IBM's Jazz as a development platform. Jazz offers web-based dashboards for projects, teams and individual contributors. The dashboards are highly configurable and offer many different kinds of widgets. Figure 1 shows an example of a dashboard in Jazz. Jazz also provides several feeds to keep developers updated on events such as build results, modifications to tasks or incoming tasks and approvals. While we did not explicitly expect to see awareness as the main theme of this research, it emerged during our study of how and why these tools are used.

With our study, we aim to assist managers and developers in their decisions about using awareness tools in software development projects. We examine the role of dashboards and feeds in supporting development activities, and we investigate what advantages and disadvantages are associated with these tools, as well as how their combination impacts collaborative development. While many research tools aiming to provide awareness of some sort have been implemented, only very few have been evaluated in an industry setting, and there is no comprehensive theory on awareness needs and the interplay of awareness tools in software engineer-

ICSE '10, May 2-8 2010, Cape Town, South Africa

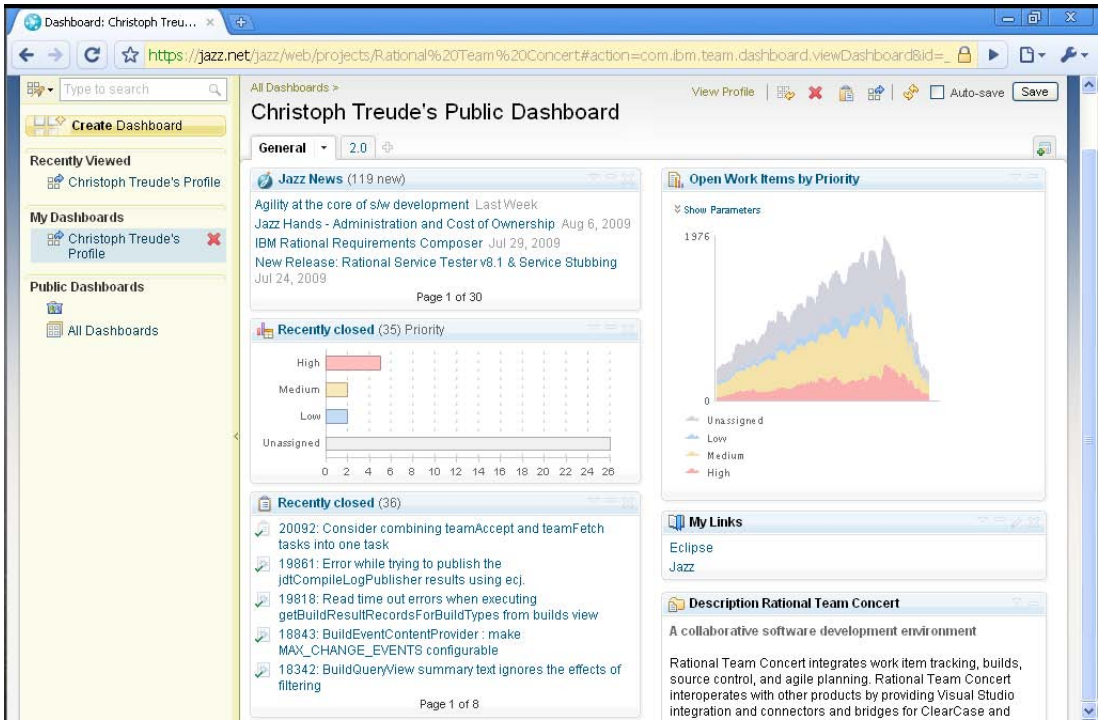

Figure 1: Sample Dashboard in Jazz

ing. Studying the early adoption of dashboards and feeds in Jazz presents a unique opportunity to start to contribute to knowledge on how an integrated development environment (IDE) should provide a comprehensive mechanism for awareness.

To gain an understanding of the role that dashboards and feeds play in awareness, we gathered data through the inspection of project repositories, by conducting interviews with developers and project managers, and through a web-based survey. Our main contribution is the identification of the different ways in which dashboards and feeds as provided by Jazz support awareness in software development and how the interplay of these tools provides awareness on different levels. We also suggest tool enhancements based on the results of our study.

The remainder of this paper is structured as follows. In Section 2, we discuss related work on awareness in collaborative software development and lightweight tool support. We introduce dashboards and feeds in Section 3. Our research questions and methodology are presented in Section 4. Section 5 comprises the main part of this research and describes how dashboards and feeds are used by software developers. Subsequently, Section 6 discusses our findings and we identify limitations of our work in Section 7. Our work is concluded in Section 8.

2. BACKGROUND AND RELATED WORK

Work related to our research can be divided into two areas: research on awareness in collaborative software development and research on lightweight tools to support social development processes. Our study can be seen as the intersection of these areas: exploring lightweight tool support with the emergent theme of awareness.

2.1 Awareness in Software Development

Maintaining awareness of each other's activities, the overall project status and current tasks is challenging as it is not limited to simple metrics based on source code or the fulfillment of a single plan item. Dourish and Bellotti define awareness as *"an understanding of the activities of others, which provides a context for your own activity"* [4]. In complex software development processes, awareness includes being aware of technical and social aspects of the development [3] as well as current and upcoming articulation work [16]. Depending on the context of the task at hand, the required granularity of this awareness can vary significantly.

Many researchers have recognized awareness as an essential part of collaborative software development and collaborative work in general. In their studies on groupware, Gutwin *et al.* found a need for workspace awareness to sustain effective team cognition and detailed awareness of one another [11]. Awareness can be maintained in an active or in a passive manner. While active mechanisms require the explicit generation of awareness data such as directed messages, passive awareness exploits information already available in a shared workspace [4]. Even if achieved passively, maintaining awareness of other's activities and the project status is time-consuming [20] but it is indispensable in the construction of mental models of a software project [15].

Tool implementations that support awareness have traditionally focused on source code and information available from source code management systems. An early example is Seesoft [6], a software visualization that maps each line of source code to a thin row and uses colours to indicate changes. Augur [9] adds software development activities to a Seesoft style visualization, thus allowing for the exploration of relationships between artifacts and activities.

FastDASH [2] uses a representation of a shared code base to highlight current activities aggregated at the level of files and methods. The workspace awareness tool Palantír [21] follows a similar approach by providing insight into workspaces of other developers, in particular artifact changes.

These tools allow for insights regarding the current activities in a project. However, in large projects, they often fail to provide an overview of the overall status of a project. Researchers have recognized this challenge and have started to develop tools that allow for higher level insights. Social Health Overview [7] works at the level of change requests and mines the history of development artifacts to reveal social and historical patterns in the development process. CruiseControl[1] achieves high-level insights through the use of a dashboard that focuses on builds rather than source code. The World View in Palantír also addresses *"awareness in the large"* [22]. It provides a comprehensive view of the team dynamics of a project, in particular regarding the geographical location of developers.

Most of these tools lack evaluation in industry settings. A notable exception is WIPDash [12], a large screen visualization for small co-located teams designed to increase awareness of tasks and source code activity. WIPDash was evaluated with two small teams during a one week field study. In contrast, the teams in our study have been using the Jazz awareness tooling for at least one year. Studying the use of dashboards and feeds in several large projects using Jazz provides a unique opportunity to examine the role awareness tools play in software development. As Jazz is the first environment to tightly integrate these awareness tools into the IDE, it also allows us to study the interplay of awareness tools rather than to examine isolated features.

2.2 Lightweight Tool Support

Research on lightweight tools to support social development processes is not yet far advanced. After the success of Web 2.0 [18] and its lightweight collaboration and communication mechanisms such as wikis, blogs, tags and feeds, software developers and researchers have started to ask how these mechanisms can be transferred into software development. For the social aspects of team-based development practices, lightweight collaboration and an *"architecture of participation"* [19] are promising.

Annotating artifacts is one of the mechanisms associated with Web 2.0. In our previous work, we explored how task annotations [24] and tagging [25] support informal individual, team and community processes in software development, including life cycle management, task assignment and task categorization. Websites such as delicious[2] and facebook[3] have found counterparts in software development. For example, Dogear [17] introduces the idea of social bookmarking for large enterprises and Codebook [1] uses the approach of facebook for software development.

Here, we explore two further Web 2.0 mechanisms and their use in software development: dashboards and feeds.

3. DASHBOARDS AND FEEDS

In this section, we introduce the functionality of dashboards and feeds as they are implemented in IBM's Jazz.

[1] `http://cruisecontrol.sourceforge.net/`
[2] `http://delicious.com/`
[3] `http://www.facebook.com/`

3.1 Dashboards

Dashboards are information resources that support distributed cognition; they are crucial to many business intelligence applications [5]. Dashboards in the Jazz IDE are displayed and configured using the web interface. They are intended to provide information at a glance and to allow easy navigation to more complete information. By default, each project and each team within a project have their own dashboard; and an individual dashboard is created for each developer when they first open their web interface. Figure 1 shows an example dashboard. A dashboard consists of several **viewlets**. Viewlets are rectangular widgets displaying information about some aspect of a project. Each viewlet is an instance of a **viewlet type**. The actual content shown in a viewlet depends on the viewlet type, e.g., visual representations of the current workload or a list of members on a team, as well as the way the particular instance has been configured. Developers can add viewlets to their dashboards and configure the viewlets using different parameters. While the list of available viewlet types is constantly expanding, Table 2 shows common viewlets at the time of our study with a short description. Viewlets can be organized into different tabs within a single dashboard.

By default, dashboards only display general purpose viewlets containing information about developers and teams, and links to general feeds. In addition to individual customizations of dashboards, project managers or component leads can customize the default settings. It is intended that the development manager of a project is in charge of updating the project dashboard, the component leads are responsible for the team dashboards, and individual developers change their own dashboards.

3.2 Feeds

For awareness on the basis of events, Jazz provides feeds. Feeds can either be displayed in the client application or as a viewlet as part of a web-based dashboard. The most common way to view feeds is through the *Team Central* view in the client application. Figure 2 shows an example. *Team Central* is not accessible through the web interface.

Team Central is organized into multiple sections that are updated continually with the latest events. By default, *Team Central* displays a bar chart of current work for the signed-in developer by priority. The event log in the middle of the view's default configuration shows feeds. These are configured to include build events for all teams that the signed-in

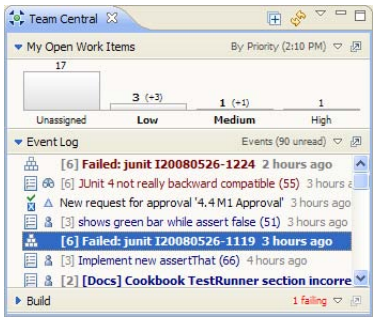

Figure 2: The *Team Central* View in Jazz

developer is part of, work item[4] changes that are pertinent to the signed-in developer, and changes to teams. Developers can add or remove feeds and filter events to personalize their event log. In addition, incoming events are displayed as small popup windows in the client application.

Unlike dashboards, feeds and *Team Central* do not offer functionality for sharing sections, views or events.

4. RESEARCH METHODOLOGY

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

4.1 Research Questions

Our research questions focus on how and why awareness tools are used, as well as on the impact of awareness tools on software development practices:

1. What is the role of dashboards in supporting individual and collaborative software engineering activities?
   (a) How are dashboards adopted and adapted?
   (b) Why are dashboards used and which roles do they support?
   (c) Which individual and collaborative processes do dashboards support?
   (d) How does the use of dashboards evolve over the life cycle of a project?
2. What is the role of feeds in supporting individual and collaborative software engineering activities?
   (a) How are feeds adopted and adapted?
   (b) Why are feeds used and which roles do they support?
   (c) How does the use of feeds evolve over the life cycle of a project?
3. What is the impact of dashboards and feeds on development practices?
4. What are potential tool enhancements?

4.2 Methodology

In the following paragraphs, we outline the setting of our research as well as the data collection methods we used.

4.2.1 Research setting

Our study took place with several development teams from IBM. We conducted a detailed study using archival data, semi-structured interviews and a web-based survey with one team, while the other teams were only invited to participate in the survey. The team of our detailed case study develops the Jazz platform and consists of approximately 150 contributors in about 30 functional teams, with some teams acting as sub-teams of larger teams, and many contributors assigned to multiple teams. The team members are located at 15 locations worldwide, primarily in North America and Europe. The developers have been using Jazz for more than three years and follow the "Eclipse Way" development process [10]. This process, developed by the Eclipse Development team, is an agile, iteration-based process with a focus on consistent, on-time delivery of software through continuous integration, testing, and incremental planning.

In addition, the survey was sent out to 1,082 Jazz users that are not part of the Jazz development team. They have been using Jazz for about one year and mostly develop business intelligence and financial applications. Not all of these additional developers follow agile development processes. They work on four interrelated projects.

We focused our analysis on development teams using Jazz because it is the first development environment that tightly integrates dashboards and feeds. Thus, it allows for early insights into the role that these tools can play in collaborative software development.

4.2.2 Data collection

Our methodology followed a mixed method approach, collecting both quantitative and qualitative data to allow for triangulation. To gather quantitative data on the configuration of dashboards, we accessed the repositories of the detailed case study team and extracted all dashboards along with the viewlets used and their configurations. We analyzed 311 dashboards containing a total of 2,975 viewlets. At the time of our extraction in June 2009, the team was working on the 2.0 release of their product. For the detailed case, we collected qualitative data through a series of interviews and a web-based survey. We conducted a total of nine interviews with the development manager, the project administrator, one component lead and six developers from five different component teams.

The web-based survey was given to both the developers in our detailed case and the developers in the other teams. It allowed us to reach significantly more participants than through interviews. We received 21 responses from the developers in our detailed case (response rate 14%) and 98 responses from other developers (response rate 9%). 76 of our 119 respondents identified themselves as contributors, 28 were component leads, nine were development managers and six were project administrators.

The questions asked in our survey and interviews were similar. Both the survey and the interview scripts contained about 50 questions on participants' use of feeds and dashboards. Questions inquired about the frequency of use as well as the reasons for using these tools. We investigated which viewlet types were mainly used in dashboards and which sections were mainly used in *Team Central*, and we inquired about the reasons behind those usage patterns. We asked about tool enhancement requests and inquired about the impact of dashboards and feeds. We also sought information about the participants' roles and the teams' development practices. About half of the survey questions required a free form response, while the remaining ones were yes/no or multiple choice questions[5]. Respondents were allowed to skip questions. In contrast to the survey, the interviews allowed for clarification questions.

In addition, one of our researchers spent five months onsite frequently having informal discussions with developers regarding their use of dashboards and feeds while also facilitating member checking of our findings. The findings gained from our data collection were mirrored in his observations.

[4]In other development environments, work items are known as bug reports, modification requests or change requests.

[5]The survey questions are available online at `http://tinyurl.com/awareness20`.

5. FINDINGS

This section presents the study findings organized by our research questions.

5.1 The role of dashboards

In a first step to answer our first research question, we report the number of dashboards and viewlets to establish the content of a typical dashboard. Then, to add to the small body of empirical research on awareness, we report the reasons for dashboard use and the processes they support.

5.1.1 Adoption and adaption of dashboards

36 of the 74 participants that answered the corresponding question on our survey indicated that they use dashboards, and 38 indicated they do not use them at all. How dashboards are used varies significantly depending on the role of the user. While both the project administrator and the development manager in our detailed case study answered that they look at dashboards at least once a day, only 26 respondents were able to recall the last time they looked at a dashboard. Out of these 26 dashboard users, two had used a dashboard within the last 10 minutes before filling out our survey, and another four had used dashboards within the last hour. Seven participants indicated that they had used a dashboard within the last 24 hours, and 13 respondents looked at a dashboard within the last week.

Of the 311 dashboards in our detailed case, two were project level dashboards, 72 belonged to teams, and the remaining 237 belonged to individual contributors. While 121 out of 168 contributors owned only one personal dashboard, there were 47 cases in which contributors created more than one dashboard. That was particularly the case for developers that were part of several teams.

Table 1: Number of viewlets per dashboard

# viewlets	# dashboards
> 100	1
51-100	3
21-50	19
11-20	38
7-10	66
6	73
5	101
< 5	10

The number of viewlets per dashboard is shown in Table 1. The majority of dashboards contain five or six viewlets, which is the number of viewlets included in the default configurations (five for contributors and six for teams).

Table 2 identifies the most used viewlet types, partitioned by the scope of project, team and contributor. Viewlets either show information on project members (e.g., *About Me* and *Team Members*), on artifacts (e.g., *Work Item Statistics* and *Builds*) or on events (e.g., *Feeds*), or they display static content (e.g., *Bookmarks* and *Description*). Overall, the most used viewlet types are *Feeds* and *Work Item Statistics*. *Work Item Statistics* display a graphical representation of the results of one work item query, visualized by one parameter. Data can be presented as bar chart, pie chart, tag cloud or table. On the project level, *Feeds* and *Reports* are the most common viewlets, while *Work Items* and *Work Item Statistics* account for most viewlets on the team level. Figure 1 shows examples of the six most used viewlet types.

By default, a contributor dashboard displays instructions on how to customize it, an *About Me* viewlet and a *Feed* for a Jazz newsgroup along with two *Bookmarks* viewlets. A default team dashboard shows customization instructions, an *HTML* team description, a *Feed* for team events, the *Team Members*, and *Plans* and *Builds* for the team. These defaults can be adjusted per project and per team, but we found that managers and component leads rarely used that opportunity. Dashboards are individually customized instead. It is impossible to report the default viewlets among the common viewlets as some viewlets may be part of the default configuration, but have been individually customized by developers, e.g., by subscribing to a different feed.

5.1.2 Reasons and roles for dashboard use

The reasons for dashboard use can be categorized into project awareness and individual convenience.

Gaining a high-level **overview of the project status** was named 14 times in our survey and in six out of the nine interviews we conducted: *"That's what dashboards are for: to give you this sort of overall high-level view of things – it is a faster way than running separate work item queries to see which items are [...] in need of attention."* This high-level overview was found to be particularly useful when looking at releases. In fact, we found that project and team dashboards were often organized by releases, with a separate tab for each release: *"It's really looking at the release at a glance. All the plan items that we have committed to, the open vs. closed, some of our burndown, [...] and statistics broken down by team."* The ability to produce high-level overviews is especially important for project managers: *"The development manager, that's the perfect person to be using the dashboards, 'cause that's what you want, you want to be reporting things all the time, based on a lot of data."*

In addition to the awareness of the project status, dashboards help with awareness of other developers and teams and their current focus: *"I then found it also helped to give a* **peripheral awareness** *of how the other teams were doing, which I would not have had if I just ran the query for the [my team] items."*

Dashboards also help with the **identification of bottlenecks** in the development process: *"There were so many things going on late in the game in [release], that I was relying on that [dashboard] in some respects, too. In the timing meetings, someone would say, 'hey, there's still these items showing up on the dashboard for [a team], is somebody looking at them?' [...] As you're getting towards the end, you're paying more close attention to the last few remaining items as you wind down – and having high-level awareness of what those are and what state they're in is good."* Dashboards are used for *"finding out about work items that need to be fixed or addressed at the end of a development phase."*

For project managers, dashboards provide the opportunity to **compare teams** against each other and make differences between teams visible: *"One of the reasons I use them a lot is [...] for showing the teams against one another, 'cause it sort of encourages a bit of a competitive thing. Because when you show defects for example, no one wants to be the team that's at the top of the list."*

For individual convenience, some viewlets allow for **navigation** to the underlying queries and work items by clicking

Table 2: Number of viewlets per viewlet type

type	project	team	individual	sum	description
Feeds	7 (24%)	77 (9%)	545 (26%)	629 (21%)	internal or external feed
Work Item Statistics	3 (10%)	120 (15%)	298 (14%)	421 (14%)	chart of work item query result
Work Items	0 (0%)	149 (18%)	168 (8%)	317 (11%)	result of work item query
Reports	6 (21%)	111 (14%)	178 (8%)	295 (10%)	predefined reports
Bookmarks	2 (7%)	18 (2%)	241 (11%)	261 (9%)	customizable list of bookmarks
HTML	0 (0%)	56 (7%)	188 (9%)	244 (8%)	snippet of HTML mark-up
Work Item Queries	1 (3%)	10 (1%)	176 (8%)	187 (6%)	links to executable queries
About Me	0 (0%)	3 (0%)	168 (8%)	171 (6%)	information about a contributor
Builds	2 (7%)	73 (9%)	64 (3%)	139 (5%)	notifications from the build engine
Team Members	0 (0%)	70 (9%)	28 (1%)	98 (3%)	list of contributors with roles
Plans	1 (3%)	47 (6%)	32 (2%)	80 (3%)	progress of plan for an iteration
Description	1 (3%)	65 (8%)	7 (0%)	73 (2%)	description of a project or team area
Other	6 (21%)	20 (2%)	34 (2%)	60 (2%)	e.g., server status, list of sub-teams
Sum	29 (100%)	819 (100%)	2127 (100%)	2975 (100%)	

on the data. 19 out of the 35 participants in our survey who answered the corresponding question indicated that they use dashboards for navigation: *"I just use that as a shortcut, so I don't have to go and remember the [work item] number."*

Sometimes, work items are specifically flagged using the tagging feature for work items to increase their visibility and to make sure that they show up on a dashboard: *"There's things that we specifically track. For example, when we were doing the [release] cycle and we wanted to track [a specific issue], we put a* **tracking** *tag on it, so that it would show up here – it just raised the visibility of it."*

Three out of our nine interviewees use dashboards as a work item **inbox**, in part because of the compact presentation: *"I waste less time navigating through work item queries for things I need to do because they are all on one page."*

We asked our participants if the information displayed in dashboards was otherwise available. All participants agreed that it was, but added that it was much easier to access it through the dashboards: *"Yes, but just not as easily. Because every one of these is based on a query, and all you are really doing is just categorizing things, showing them with a particular view. It's just that you can't get that instantaneous summary of the information, especially if you've got thousands and thousands, that's difficult."*

While project and team awareness is tentatively more important for project managers and component leads and individual convenience is the main use case for developers, it is hard to separate these roles – partly because individuals often do not clearly belong to one group or the other. Out of the 76 respondents that identified themselves as contributors in our survey, only 25 stated that they spend 100% of their time on development activities. 27 of the 76 contributors indicated that they spend 50% or less of their time on development activities. At the same time, 13 of these contributors spend 50% or more of their time on project management activities.

5.1.3 Dashboards support individual and collaborative processes

In Jazz, there are dashboards for three different scopes: project, team and contributor. These dashboards are entirely separate. We found that the dashboards that are used regularly are almost exclusively project and team dashboards. Personal dashboards are usually left in their default configuration, which shows the roles of the developer along with web links and newsgroups. With the current configuration, default dashboards have practically no value beyond indicating team membership.

An interesting case where dashboards support an individual process is the use of feeds viewlets by one developer, considering that feeds are also available in the client application: *"Having that viewlet has saved me a couple of times where I've been wanting to look at the work item inbox from home, and I didn't have the Eclipse client, so my usual path [...] wasn't available to me. So I used the feed viewlet."*

Developers and managers customize dashboards for their own benefit but also do so to communicate important insights to their teams: *"It's not just stuff for me. I often put dashboards up, for things that are not actually things that I care so much about, but they're things that I know the teams will care about – or that I want them to care about."*

As dashboards are web pages, it is easy to share them and to send links to dashboards to other project members. This opportunity is used frequently by managers and component leads, using tools such as instant messaging or mailing lists, posting links to dashboard pages on the project wiki, or using dashboards to follow along during planning calls. When communicating the current status of a project to executives, dashboards are also used: *"Often we go to some executive meeting, and they would say – for example, there might be 400 bugs outstanding, and they'd say, 'what insurance can you give us all this stuff is going to be fixed?' and I could say, 'look, in this 3 week cycle we fixed 2500 bugs, so really fixing 400 is not a big deal.' "*

5.1.4 Dashboards evolve over project life cycle

As mentioned before, project and team dashboards often contain release specific information. A prominent example is given by this quote: *"During the [release] cycle, we had this middle column, which is called the mustfix column, which is basically all the defects that have to be fixed for the release. And everything else is, well it's important, but it's not as important as what's in here. [...] And people were constantly looking at those to see who has stuff that needs to be fixed."*

This dependence on releases implies that dashboards have to be changed when a product version has been released: *"So*

now, as we're starting all this new work, this dashboard will probably change a fair bit." Jazz does not offer any means to automate that process – *"cleaning up"* dashboards is a manual task. However, 25 out of the 36 developers that use dashboards indicated that their use is constant across different project phases. The few exceptions mentioned the "endgame", i.e. the last cycle of the project before a release.

5.2 The role of feeds

This section looks at a finer granular awareness tool: feeds. Since the data on the use of feeds by developers is not stored on a central server, our findings below are based on the results from our interviews and our survey.

5.2.1 Adoption and adaption of feeds

Feeds can be accessed through the *Team Central* view, through dashboards and through a couple of other mechanisms in the client application such as e-mail notifications. 52 out of 86 participants indicated they made use of the *Team Central* view (cf. Figure 2) and only 10 out of 78 participants that answered this question indicated that they use feeds outside of *Team Central*. When asked what was the last time they looked at *Team Central*, nine looked at it in the last 10 minutes, seven in the last hour, nine in the last 24 hours, 15 in the last week and 12 did not remember when they last accessed *Team Central*.

The feed that most developers look at is the *My Work Item Changes* feed, an event feed of changes to work items that the signed-in developer owns, created, modified, or is subscribed to. Only two developers took advantage of the opportunity to create their own custom feeds, e.g., based on individual work item queries.

5.2.2 Reasons and roles for feed use

Feeds are primarily used to "**track work** *at a small scale"*. The majority of participants who answered the corresponding question in our survey indicated that they use feeds *"to see what work items are updated and new ones coming in quickly and easily."* Then, the work items that catch interest are expanded. Feeds are similar to e-mail notifications in other systems. In fact, a reason to use feeds is because *"it allows [you] to turn off all work item related e-mail."*

The event log in *Team Central* is largely seen as a **personal inbox** that is primarily used to answer the question *"What should I do next?"* and thus helps developers **plan** their day. It also helps to quickly **get information** such as the due date of a particular feature. The other sections of *Team Central*, in particular the team load section, are used frequently to get information on the overall status of a team.

Team Central and feeds are also used by new developers on a team to get an overview of what the team is working on and to **understand common work practices**.

We found no difference in the use of feeds between different development roles. While feeds support collaborative work, how they are used is not shared with other team members.

5.2.3 Feeds evolve over project life cycle

The extent to which developers use *Team Central* is mostly constant across different project phases. Out of 52 answers, only 14 participants in our survey indicated that their use of *Team Central* is not constant over time. The exceptions were developers who are fairly new to developing software in Jazz and thus were unsure how the view works.

5.3 The impact of dashboards and feeds

Awareness tools increase the transparency in collaborative software development. As mentioned in Section 5.1.2, competition between teams is one of the reasons dashboards are used by managers and component leads. They are aware of the **peer pressure** effect that arises from this competition, and they see it as one of the benefits of using dashboards: *"That's partly why we use them, they definitely do [create peer pressure]. I mean, it increases the exposure and the visibility a great deal. [...] An executive is probably unlikely to be going in writing a query or browsing around in the work items but it's easy for them just to go here and just right away see."* Competition is usually seen as a good thing: *"The need to look like you are making progress is useful."* Any peer pressure is based on data that already exists: *"I mean we're showing data that's already there, it's just making it visible at a glance."*

Developers and managers are aware that dashboards can only be as good as the data that they display, i.e. peer pressure should not occur because of outdated or incorrect data: *"The problem is work items are only as good as the data that's in the work items and the way they're being filed, so obviously, if all of this is going to be useful, we need to ensure that the data that's in the open work items is useful, too. [...] If you try to manage purely by numbers, then that's a big mistake. You've got to always be applying common sense."* In some cases, correcting steps are taken so that the dashboards and feeds are **not misleading**: *"Sometimes we have to put those caveats on things. For example, there would be something and would show some team with a really long bar. Sometimes there are reasons for that. Just because one team has a lot more defects than another that doesn't necessarily mean that the quality of that component is any worse."*

It is worth noting that when we asked developers in our survey if they would alter their work practices because of information displayed in dashboards, 26 out of 32 participants who responded to that question answered no. However, we noted that some developers look at dashboards to compare team activities: *"Just knowing that you're not the component that's on the critical path for number of bugs remaining."*

Whether awareness tools are **distracting** or not largely depends on their use of push vs. pull mechanisms. None of our respondents found dashboards distracting, in particular because they are not part of the client application but are displayed in a web browser instead: *"You've got to go there to look at them. Of course you could choose not to look at them. I guess one thing that can happen, is just in terms of the amount of noise or information on them – sometimes people go there and they're going looking for one particular thing, and they're seeing a thousand other things going on."* Feeds were sometimes considered to be distracting. Six out of 40 respondents found that they receive irrelevant event notifications. On the other hand, some developers consciously *"try to limit unimportant changes and combine them into one change"* to reduce the number of events.

5.4 Potential tool enhancements

5.4.1 Enhancements for dashboards

One of our findings is that developers are often unaware of the full functionality of dashboards. In these cases, our study sometimes helped to raise awareness of features. Two of our interviewees changed their dashboard configuration

during the interviews because they learned about the available functionality. While it is difficult to make developers work through the complete list of features, the utility of dashboards could be made more apparent by adding **advanced default dashboards**. The same applies on a more fine grained level to different settings for particular viewlets. We found that developers were often unaware of some of the available settings, e.g., visualization and feed configuration settings. Again, default viewlets could help communicate the full utility of the dashboard functionality.

Developers often miss support for dashboards in the **client applications**. An integration would enable them to open links to artifacts in the client instead of the web browser.

In addition, more **inbox functionality** would improve the usefulness of interactions with dashboards. Developers who use dashboards to check events requested that events would disappear from the viewlets once read. Also, more information such as upcoming builds, milestones or feature freezes and pending approvals should be easier to access.

In dashboards with more than one tab, tabs often get missed because they are not obvious to the user. In this case, it would be helpful to show an overview of the **available tabs** rather than the first page by default or to offer filtering mechanisms such as only displaying tabs pertinent to a certain release.

There were also requests to **improve the visualizations** in dashboards. For example, to compare open vs. closed work items per team, two viewlets need to be created: one for open work items and one for closed work items. Having a stacked chart showing both would save screen real estate and make comparisons easier.

With the heavy reliance of many project and team dashboards on the release cycle, it should also be possible to let viewlets or whole tabs **expire** when a new version of a product has been shipped. This would help to avoid having outdated information on dashboards.

It is impossible to unite data from **different projects** into one dashboard right now. For developers working on more than one project at the same time, a common occurrence, this is a major shortcoming.

5.4.2 Enhancements for feeds

Most of the enhancement suggestions for feeds and *Team Central* involve adding additional information and perspectives to the tooling. A **burndown** view that shows a graphical representation of remaining work vs. time was requested several times, along with schedules, general announcements, and a section focusing on quality control.

For work item feeds, it would be useful to identify **relationships between tasks** that are affected by events. When a sub-task is completed, the tool should show which parent task it belongs to. It would also be useful to offer more **filter options** for feeds. In particular, it should be possible to distinguish between items that require action, such as failed builds, and items that do not require any action, such as successful builds. Alternatively, the number of events can be limited by **grouping** events that belong together. This is already implemented for some cases, but can further be improved.

For some developers, the reason not to look at feeds in their *Team Central* view is the **time interval** for the auto reloads of the event log section. Enabling e-mail notifications usually displays events faster than the event log.

Since the layout of *Team Central* and the **configuration** of specific feeds are not stored on a central server, they are lost whenever a new version of the client application is installed or whenever a developer switches clients. For many developers, this is a reason not to configure their views.

Switching between *Team Central* and dashboards is tedious and could be avoided by having an analog to *Team Central* in the **web interface**.

6. DISCUSSION

This section discusses our findings and identifies future directions for tool designers.

Reliance on informal tool features for critical phases.

One of our least expected findings is the reliance of developers and managers on dashboards in the most critical project phase: the "endgame" right before a release. The project teams in our study used dashboards to stay aware of bottlenecks in the process and work items that needed to be closed before the shipping date. In fact, the defects to be fixed before a release were identified using a combination of informal mechanisms: work items were tagged with "mustfix" to indicate their importance, and then dashboards and feeds were used to track the state of those work items. In one of the dashboards, a "mustfix" column was implemented that only showed viewlets based on the "mustfix" tag, and feeds were used to see state changes of the "mustfix" items in real-time. This reliance on informal tools demonstrates both the versatility and the reliability of informal tools in software development. Informal tools can be used to organize artifacts in a lightweight manner and to support tasks that do not occur every day.

Relationship between dashboards and feeds.

There are three main differences between dashboards and feeds: **granularity**, **privacy** and **configurability**. Even though it is possible to make dashboards private, most developers share dashboards with the whole team. Project and team dashboards are shared by default with the entire team. This is in contrast to feeds in the client application: *"In Team Central it's a bit too introverted, because it's your own view of things, and you can't share that with people."*

The suggestions for tool enhancements indicate that the relationship between dashboards and feeds is not clear-cut. It was suggested to introduce dashboard functionality into *Team Central*, in particular burndown views and grouping of events. At the same time, developers requested more inbox functionality for dashboards, such as viewlets for incoming work items and work item changes.

Implementing these suggestions would result in *Team Central* being very similar to dashboards and vice versa. There are two different conclusions possible: first, a clear distinction between dashboards and *Team Central*. As one of our interviewees put it, *"there seems to be a split between managers and non-managers"* with regard to their tool use. This border could be sustained, and developers could be required to use dashboards as soon as they are looking for information outside their own work items. Alternatively, dashboards and *Team Central* could consciously be aligned to have similar functionality and configurations. This could go as far as changes to sections in *Team Central* being reflected in dashboards and vice versa. In the current setting we also

found that individual dashboards do not have obvious uses, unlike team and project dashboards. Individual dashboards often remain unchanged, display outdated information such as links to feeds with expired passwords and are rarely used.

The current distinction between overall project status in dashboards and pertinent work item changes in the client application is insufficient as developers and managers need both: awareness in the large and in the small, but to different extents. An example are the "mustfix" work items: developers not looking at the project dashboard regularly had to rely on being informed by somebody of the "mustfix" tag in order to adjust their priorities. Our results lead us to believe that integrating web-based dashboards and *Team Central* functionality would solve some enhancement requests and provide a clearer conceptual model of awareness.

Dashboards and feeds are ephemeral resources in a software development environment. The secret of their success in the projects that we observed may lie in the fact that often team memberships and roles turned out to be ephemeral as well. When developers need to take on tasks of project managers and vice versa, a strict distinction between high-level awareness and low-level awareness is rendered ineffective.

The paradox of choice.

As with many highly configurable user interfaces, dashboard users suffer from the paradox of choice. Most developers are unaware of all the options available to them. This leads to unsatisfactory configurations and to developers avoiding dashboards due to not understanding their benefits. Instead of adding more viewlet types and properties, removing the least used viewlet types and offering well-designed default configurations would be a better solution.

Towards IDEs as socially translucent systems.

In their work on *"socially translucent systems"*, Erickson and Kellogg suggest that systems that support social processes have three characteristics: visibility, awareness and accountability [8]. IDEs are also systems that support social processes in software development; and thus need to be designed with these characteristics in mind. These issues were clearly evident in our findings on how dashboards and feeds provide different levels of awareness in software development. Many enhancement requests contradict the original intent of the individual features. When combined, the enhancement requests point to the need for one integrated system rather than a set of isolated features to improve awareness and visibility.

Research on awareness in collaborative software engineering lacks a theoretical background. One of the most important themes that came out of our work is the need to design studies that look beyond one individual tool and focus on the interplay of processes, artifacts and developers. We need to evaluate how awareness tools and features are used across different dimensions of collaboration, and we need to formulate new theories and add to existing theories from the domain of Computer Supported Cooperative Work (CSCW) to be able to assist managers and developers in software projects.

7. LIMITATIONS

As with any chosen research methodology, there are limitations with our choice of research methods.

For usage data on both dashboards and feeds, we had to rely on answers from interviews and from our survey. Jazz does not log data on the use of these artifacts. However, a strength of our mixed method approach is that we are able to triangulate findings obtained through different methods. The large number of respondents in our survey also provides a good basis for our conclusions.

Some of the individuals who participated in our survey skipped some of the questions while filling out the survey. Consequently we reported throughout the paper the number of responses to the questions discussed. Another limitation lies in the low response rates to our survey. However, as it is the nature of a large software 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 achieve better response rates.

When the teams started on their projects, dashboards were still being improved and new viewlet types were added. This might have influenced the specific usage. Because our detailed case study was conducted with the developers of Jazz, their willingness to adopt dashboards and feeds might be above-average. However, we confirmed our findings with several non-Jazz teams. While the team in our detailed case used an agile development method, not all of the additional teams were agile. We found no differences between users across the different teams and development methods.

Our conclusions are limited to how dashboards and feeds are implemented in the Jazz environment. However, Jazz is the first development environment supporting configurable dashboards for every developer and giving a prominent role to feeds, thus providing a first chance to study how such tool features are used. As more projects and development environments integrate dashboards and feeds, additional studies need to be conducted to gain further insights into the use of these awareness tools in software development.

8. CONCLUSIONS AND FUTURE WORK

The main contributions of this paper are the identification of different uses for dashboards and feeds in collaborative software development, the discussion of their impact and their interplay, as well as concrete suggestions for tool enhancements and integration.

Awareness in a software development project is composed of many different aspects: developers need to stay aware of the overall status of their projects and critical deadlines, they need to understand current priorities and bottlenecks in the process, they need to know of dependencies between components and teams, and they need to be informed of changes to tasks they are working on in a real-time manner. Previous efforts have often neglected high-level awareness and focused on source code or source code management systems instead. While the awareness of source code is crucial to software development, it is insufficient when development is a team effort. Developers need to be able to see current plans, upcoming deadlines, schedules of other team members, and critical components.

Our research has shown how several teams of developers using IBM's Jazz use a combination of feeds and dashboards to maintain awareness of various development aspects: dashboards are mainly used to keep track of the overall project status, to provide peripheral awareness of the work of other teams and to stir competition between teams. During critical project phases, dashboards also evolve into the central

entity where priorities of tasks are organized and shared. Therefore, dashboards depend largely on the project's life cycle. Feeds are used to track work at a small scale and to plan short term development activities.

As developers need to understand issues that span several teams and components, the line between requirements for dashboards and feeds becomes blurred and tool enhancement requests add up to a considerable overlap of both tools. Future work will have to be conducted to identify the ideal toolset for awareness in collaborative software development. Our findings suggest that such a toolset brings awareness of projects, teams and tasks together and no longer separates between awareness in the large and awareness in the small.

9. ACKNOWLEDGEMENTS

We wish to thank the team that granted us access to their repositories and conducted interviews with us as well as the participants of our survey. This research is supported by a fellowship from IBM and the second author is a visiting faculty member at IBM. We appreciate comments from Jean-Michel Lemieux, Tricia Pelz, Adrian Schröter, Nancy Songtaweesin and Annie Ying that improved the paper.

10. REFERENCES

[1] A. Begel and R. DeLine. Codebook: Social networking over code. In *ICSE Companion '09: 31st Intl. Conf. on Software Engineering - Companion Volume*, pages 263–266, Washington, DC, 2009. IEEE.

[2] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: A visual dashboard for fostering awareness in software teams. In *CHI '07: Proc. of the Conf. on Human factors in computing systems*, pages 1313–1322, New York, 2007. ACM.

[3] 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 Intl. Conf. on Supporting group work*, pages 105–114, New York, 2003. ACM.

[4] P. Dourish and V. Bellotti. Awareness and coordination in shared workspaces. In *CSCW '92: Proc. of the Conf. on Computer supported cooperative work*, pages 107–114, New York, 1992. ACM.

[5] W. Eckerson. *Performance dashboards: measuring, monitoring, and managing your business*. Wiley, 2005.

[6] S. G. Eick, J. L. Steffen, and E. E. Sumner, Jr. Seesoft - A tool for visualizing line oriented software statistics. *IEEE Trans. on Software Engineering*, 18(11):957–968, 1992.

[7] 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 Conf. on Human factors in computing systems*, pages 577–586, New York, 2007. ACM.

[8] T. Erickson and W. A. Kellogg. Social translucence: An approach to designing systems that support social processes. *ACM Trans. on Computer-Human Interaction*, 7(1):59–83, 2000.

[9] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *ICSE '04: Proc. of the 26th Intl. Conf. on Software Engineering*, pages 387–396, Washington, DC, 2004. IEEE.

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

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

[12] M. R. Jakobsen, R. Fernandez, M. Czerwinski, K. Inkpen, O. Kulyk, and G. G. Robertson. WIPDash: Work item and people dashboard for software development teams. In *INTERACT '09: Proc. of the 12th IFIP TC 13 Intl. Conf. on Human-Computer Interaction*, pages 791–804, Berlin, Heidelberg, 2009. Springer-Verlag.

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

[14] F. Lanubile. Collaboration in distributed software development. *Software Engineering: Intl. Summer Schools, ISSSE 2006-2008, Salerno, Italy, Revised Tutorial Lectures*, pages 174–193, 2009.

[15] 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, 2006. ACM.

[16] P. Mi and W. Scacchi. Modeling articulation work in software engineering processes. In *Proc. of the 1st Intl. Conf. on the Software Process*, pages 188–201, 1991.

[17] D. R. Millen, J. Feinberg, and B. Kerr. Dogear: Social bookmarking in the enterprise. In *CHI '06: Proc. of the Conf. on Human Factors in computing systems*, pages 111–120, New York, 2006. ACM.

[18] S. Murugesan. Understanding Web 2.0. *IT Professional*, 9(4):34–41, 2007.

[19] T. O'Reilly. What is Web 2.0: Design patterns and business models for the next generation of software, 2005. http://oreilly.com/web2/archive/what-is-web-20.html.

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

[21] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: raising awareness among configuration management workspaces. In *ICSE '03: Proc. of the 25th Intl. Conf. on Software Engineering*, pages 444–454, Washington, DC, 2003. IEEE.

[22] A. Sarma and A. van der Hoek. Towards awareness in the large. In *ICGSE '06: Proc. of the Intl. Conf. on Global Software Engineering*, pages 127–131, 2006.

[23] B. Sengupta, S. Chandra, and V. Sinha. A research agenda for distributed software development. In *ICSE '06: Proc. of the 28th Intl. Conf. on Software Engineering*, pages 731–740, New York, 2006. ACM.

[24] M.-A. Storey, J. Ryall, J. Singer, D. Myers, L.-T. Cheng, and M. Muller. How software developers use tagging to support reminding and refinding. *IEEE Trans. on Software Engineering*, 35(4):470–483, 2009.

[25] C. Treude and M.-A. Storey. How tagging helps bridge the gap between social and technical aspects in software development. In *ICSE '09: Proc. of the 31st Intl. Conf. on Software Engineering*, pages 12–22, Washington, DC, 2009. IEEE.