

WIPDash: Work Item and People Dashboard for Software Development Teams

Mikkel R. Jakobsen¹, Roland Fernandez², Mary Czerwinski², Kori Inkpen²,
Olga Kulyk³, and George G. Robertson²

¹ Department of Computer Science, University of Copenhagen, Denmark
mikkelrj@diku.dk

² Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
{rfernand, marycz, kori, ggr}@microsoft.com

³ Human Media Interaction, University of Twente, 7500 AE, Enschede, The Netherlands
okulyk@utwente.nl

Abstract. We present WIPDash, a visualization for software development teams designed to increase group awareness of work items and code base activity. WIPDash was iteratively designed by working with two development teams, using interviews, observations, and focus groups, as well as sketches of the prototype. Based on those observations and feedback, we prototyped WIPDash and deployed it with two software teams for a one week field study. We summarize the lessons learned, and include suggestions for a future version.

Keywords: Information visualization, software development, large display, cooperative work, CSCW, situational awareness, field study.

1 Introduction

Team collaboration and coordination in software development is difficult [14]. First, it may require frequent coordination to plan and review progress of a team. Second, completing a task often involves team collaboration because knowledge is divided between team members who have different roles or own different parts of the system. Team members may work on multiple task items at a time, or belong to more than one team, adding to the challenge of coordination. Thus, team members need to be aware of what others on the team are doing [10,14].

In this paper, we present WIPDash (Work Item and People Dashboard), a visualization of work items in a team's software repository. Our goal is to help software teams be aware of the overall status of a project, and understand ongoing activities related to the team. Initially, we conducted interviews and field observations within a software development organization in order to understand the needs of collocated software teams. We then discussed these results in a series of focus group to iteratively design WIPDash. Finally, we deployed WIPDash with two software teams in an attempt to observe which features and functions the team actually used, and how they used those features. Our findings led us to a number of design lessons, and yet another design iteration, which we introduce at the end of the paper.

The main contributions of this work include (1) detailed findings about how developers maintain team awareness using existing techniques and tools, (2) a novel awareness visualization based on developers' needs, and (3) lessons learned from a deployment with two teams along with a conceptual overview of new design ideas based on that deployment.

2 Supporting Software Team Awareness

Many organizations adopt Agile Software Development methodologies that promote shorter iterations and daily stand-up meetings to improve coordination [15]. Also, collocation of a team in a shared team space may improve productivity [19]. Yet, software teams are still challenged with maintaining awareness of ongoing activity [10], and find collaborative tools that support their development work useful [14].

Software teams typically store information about work items (e.g., tasks and bugs) in software repositories, such as Microsoft Team Foundation Server (TFS), to support team coordination. However, such repository systems are not designed to give an overview of the state of a project or to keep team members aware of the team's current activities. In addition, it is not easy to see changes to work items in a software repository. Developers may not feel that they get a proper return on the time they invest on updating work items and often the status of work items is not up to date.

One way to improve team awareness is to show data from a team's repository on a large display in a shared workspace [e.g., 1,8]. FASTDash [1] showed developers' current activities in a code base. A field study showed that FASTDash increased communication within the team by 200%, and helped participants know who had which files checked out, who was blocked and needed assistance, and helped resolve conflicts with checked out code. Improvements were suggested based on the study, such as using metrics other than file size to allocate screen space, and to add support for people to track work items that are assigned to them. Still, it was not clear which types of information were most useful or how visualizations could be best designed to get awareness information at a glance. O'Reilly et al. [12] visualized checked-in code changes on a multi-monitor display. The authors concluded that the visualization helped to inform developers about progress and overall effort of the team. However, it is not clear how participants used the display. De Souza, Froelich and Dourish [16] have shown that source code could be mined to visualize both social and technical relationships of projects. We were inspired by these findings and focused our visualization on support for work item awareness.

Fitzpatrick et al. [6] described a long-term study of a software team using a tickertape tool where messages from CVS, a revision control system, were displayed. The authors found that the tickertape tool stimulated more focused discussion about source code changes, reduced the number of empty check-in messages, and helped coordinate and negotiate work within the team. The authors mentioned the modest screen real estate requirements of the tickertape tool as an important benefit.

Hill and Holland [9] describe the concept of showing the history of a user's interactions with files as part of the representation of the files. Recent research has empirically studied use of interaction history to help software development teams

[3,7]. TeamTracks [3] directs the attention of a programmer to important parts of the source code based on the history of programmers' interactions with the code. Augur [7] combines information about code activity with a line-oriented source code visualization similar to SeeSoft [4]. Froehlich and Dourish [7] present case studies of four developers who used Augur to gain insight into their code and their development activities. Their findings support the idea of combining information about activity and code in one view that is based on spatial organization of the code. However, Augur's potential usefulness as an awareness tool for a collocated team remains unknown.

The research efforts mentioned so far involved representations of source code, check-ins, and the use of code files. In contrast, Ellis et al. [5] aimed at helping large distributed software teams to coordinate their work on change requests by visualizing bugs. They presented SHO, a visualization with bugs shown as circles ordered, colored, and sized by different importance metrics. Participants in an experiment were more successful at completing tasks using SHO than using Bugzilla. Another recent study by Sarma et al. [14] presented a desktop awareness system based on code activity and check-ins. Their Palantír tool addressed mainly artifact changes in order to prevent potential conflicts. The visualizations were useful for exploring databases of bugs to identify areas of concern. However, it is not clear how useful these types of visualizations are for maintaining awareness of work items activity and project progress in collocated software teams, which is the focus of this paper.

3 User-Centered Design

The research presented in this paper follows a user-center design approach. We began by gathering observations and conducting semi-structured interviews with software developers to gain insight into their work practices and needs in collocated team workspaces. Our goal was to understand how collocated teams coordinate their work and to get input on what visualization features and views would be most useful for them. We then sketched an initial design of a visualization to support team awareness, which we presented to a focus group for feedback.

3.1 Interviews and Field Observations

In situ observations were performed with two Agile teams (see Figure 1). The teams were observed for five hours during one week. Observations were carried out at different times of the day and included morning stand-up meetings and iteration planning meetings. One team typically had five to ten members present, while the other team typically had eight to fifteen members present. We also carried out semi-structured interviews with ten individuals from these teams (eight males). Each interview lasted 35-40 minutes and was audio recorded with the participants' permission. Interviewees received a free lunch coupon as gratuity for their participation. Each interviewee had two to fifteen years of experience in software development and ranged in age from 20 to 46. Based on questions from related studies [10,17] and questions motivated by our in situ observations, the interviews focused on the following aspects:



Fig. 1. Shared team rooms for the two software teams we observed.

- experiences working in a collocated team workspace;
- what tools and alerts are currently used to keep track of what other team members are working on and what is missing in existing tools;
- how work items and tasks are currently managed;
- types of meetings and the use of a projector in the team room;
- how progress and project health in general is monitored;
- wishes on what to display on the large shared screen and how to support work flow.

3.2 Interview and Field Observation Results

The interviews and observations showed that the teams work in iterations, which are blocks of time typically one to two weeks long. Daily stand-up meetings in the morning help the team to keep track of who works on what, and they are considered an important time for team bonding. Work items are created in a repository.

The teams used many software tools to coordinate their activities including Team Foundation Server (TFS), email, instant messaging, Live Meeting, and SharePoint. The teams varied in work style, size, and physical workspace arrangement. Both teams adopted a seating arrangement that corresponded to individual roles on the team: developers, testers, writers, and support (including program managers).

In addition to regular stand-up meetings, iteration planning, and bug triages (where resolved bugs are discussed and new bugs are assigned to team members), ad-hoc conversations frequently occurred in the team rooms. Some team members used chat or email, but often team members just shouted out a question or rolled their chairs over to talk to each other. Moreover, other people came in and out of the team spaces.

Although shared team rooms can be noisy and distracting, and offer less privacy than private offices, most team members felt that the team room was more effective for team work. An exception to this is documentation writers (two out of ten interviewees) who said that they preferred to work in a private office or from home; they needed to concentrate and only came to the team room for meetings or when they needed to speak to a team member. Since they were frequently absent from the team room they tended to be less aware of activities that were going on within the team.

Both teams used shared whiteboards and sticky notes on the walls. Team members defined, categorized, and prioritized work items during iteration planning meetings and used sticky notes to represent tasks or work items. The teams also used a projector on the wall to display information for various tasks such as work items during iteration planning, when assigning new tasks, or for code reviews.

The dynamic nature of stand-up meetings requires a quick, glanceable overview of recent activity. Teams currently do not have a suitable tool for displaying important information. TFS gives no overview of unassigned tasks, and does not allow more than one person to be assigned to a task. Also, team members have to make a burn down chart or a task list for each meeting, which is time consuming. It is possible to export charts and work items from TFS, but this often results in a long Excel table with no way of synchronizing changes back with work items in TFS.

3.3 Focus Group Feedback

Based on what we learned from the interviews and observations, we sketched an initial design for a team awareness visualization. This initial design sketch, based on the current iteration of one of the team's work items repository, was presented to a focus group on a large projection screen in a meeting room to help participants imagine how the visualization would look when deployed in their own team room. Eight participants from two Agile development teams took part, including an architect, program managers, developers, lead developers and testers ranging in age from 30-48. The focus group session was video recorded with the teams' permissions.

Participants found that the awareness display presented information in a new perspective they had not seen before and they liked being able to see an overview of the whole project in one view. Team members expressed the need for different filters and view modes, since some work items might be irrelevant for their role.

We derived three key requirements for our design based on this feedback. The awareness visualization should (1) give an overview of iteration progress, with the ability to summarize over the last day, week, month, or version, (2) give details on individual work items and the people these are assigned to, and (3) list current and recent activities, either of people or on work items.

4 Work Item and People Visualization

Based on all of these results, we developed WIPDash, a visualization suitable for a large shared display in a collocated software team space. The visualization was implemented as a Windows application that reads data about work items from TFS. Our intention was that team members could glance at the shared display to see the overall status of the project and the recent changes made (especially from the past 24 hours). We also wanted team members to be able to use WIPDash on their individual machines, where they could switch between different views and filters, and get details on demand.

The WIPDash window consists of two parts (see Figure 2). The left part of the window contains a spatial representation of areas of the project and the work items in

those areas. For instance, the area labeled ‘Docs’ contains items related to project documentation. The right part of the window consists of a list of view modes, a team panel, and drop-down lists of iterations, work item states and types. These lists can be used to filter and highlight work items in the left view.

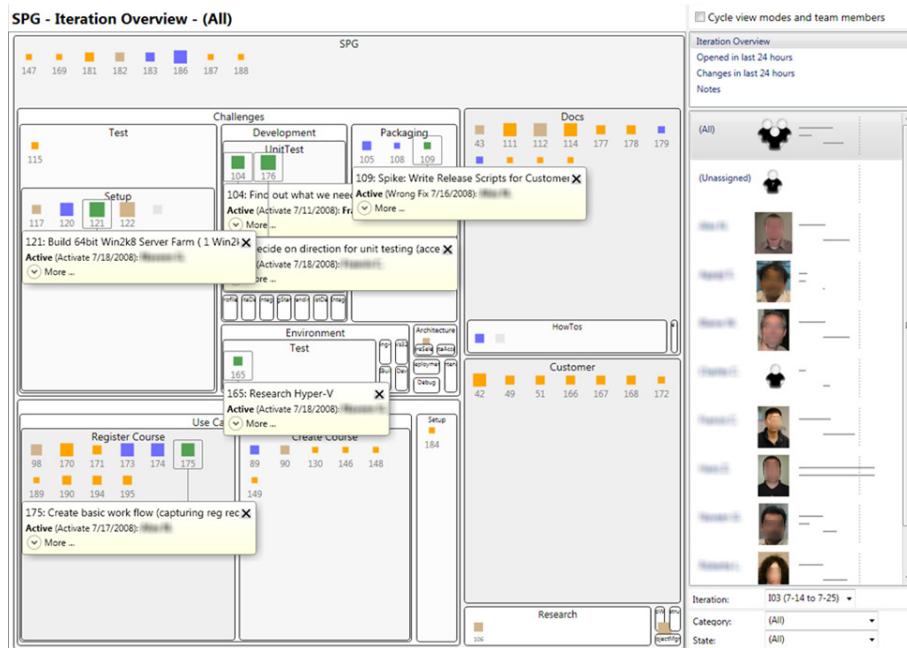


Fig. 2. WIPDash showing the iteration overview for the current iteration of a project.

4.1 Work Item Treemap

WIPDash uses a squarified treemap for laying out project areas as rectangles [2]. The treemap is a scalable approach to spatially organizing hierarchically structured data such as a hierarchy of project areas. Each rectangle is sized proportionally to the number of open work items in the area. A minimum threshold is used to ensure that areas that do not contain any open items are shown in the map. Rectangles are labeled with the name of the project area.

We wanted to preserve the spatial layout of project areas and work items to make it easier for users to remember where areas are located in the visualization. However, treemap algorithms can cause the spatial layout to change considerably when the data changes. Since WIPDash would be shown on both a large display and on individual team members’ displays (which may have different screen dimensions), the layout had to vary across instances of WIPDash. In order to keep the layout consistent for the purpose of the field study, the treemap was fixed and then shared by all instances of WIPDash. This layout could be explicitly updated and the treemap would render again. Since the relative size of an area does not change dynamically to reflect the number of open work items, we color a rectangle darker as more open work items are associated with the area.

Each rectangle in the treemap contains icons that represent the work items associated with that area. The icons in a rectangle are evenly spaced in a grid, placed in the order they were created starting from the top-left corner. Space between icons is reduced to fit all icons within the rectangle, and if there is enough space between icons, the ID number of the item is shown below the icon. The color of an icon indicates the state of the work item (e.g., proposed, active, resolved, or closed) and the shape of an icon indicates the type of the work item (e.g., feature, bug, or task). Icon size represents either priority level or estimated hours of the work item, with larger sized icons representing items of higher priority or higher estimate of work hours, as designated by the user or team.

Moving the mouse cursor over a work item icon shows a tooltip with details about the item. Clicking on an icon shows a popup window with details about the work item. An ‘Add note’ section in the detail window can be expanded to add a sticky note to the work item. A small yellow sticky note symbol is displayed on the work item to indicate that it has a note attached.

4.2 Iteration Filtering and Highlighting

Selecting an iteration in the “iteration list” shows all work items that are assigned to that iteration and highlights them on the treemap. Since teams are usually only interested in closed items for the current iteration, we removed closed items that were not assigned to the selected iteration in order to avoid clutter.

4.3 Team Panel

The team panel (see Figure 3a) contains the names and pictures of the team members. Clicking on a team member shows the icons for all work items assigned to or closed by the selected team member. The team panel contains two options in addition to the team members: (1) “all” which is used to select all items regardless of whom they are assigned to, and (2) “unassigned”, which is used to select all unassigned work items.

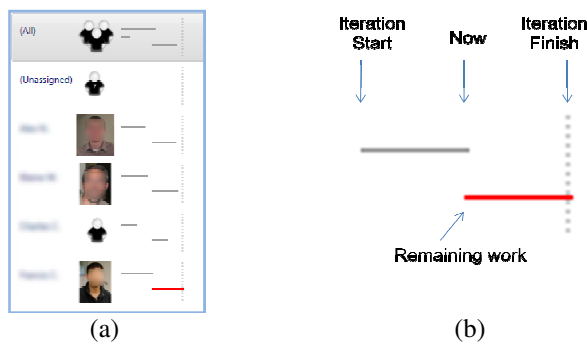


Fig. 3. (a) Team panel showing names and pictures of team members, and (b) a graphical representation of the amount of work each member has assigned, completed and remaining.

For each person, WIPDash shows horizontal lines that represent the total amount of work, the amount of completed work, and the work remaining in the iteration that is assigned to that person (Figure 3b). The x-axis measures work hours and a dotted vertical line represents the end of the iteration, corresponding to the total number of work hours in the iteration. The remaining work line is colored red if the estimated hours of remaining work exceeds the time left in the iteration.

4.4 View Modes

One goal of our design was to make the information on the display glanceable. Thus, to avoid cluttering the display by showing too much information in one view, users can choose between four different view modes.

The *Iteration overview* mode highlights all work items in the current iteration. This view aims to provide an overview of the iteration status. Team members can see how much work has been done and how many work items remain in the iteration. Details are automatically displayed for work items that are currently being working on, including who is working on the item. More details about an item can be shown by clicking on 'More' (see Figure 2).

The *Opened in last 24 hours* view is designed to keep the team aware of incoming tasks and issues. Work items opened within the last 24 hours are highlighted with a yellow border and background. The opacity of the border and background varies to distinguish recently opened items from items that were opened less recently. Also, similar to the iteration overview, details are shown for recently opened items, including when an item was opened and by whom.

The *Changed in last 24 hours* view aims to keep the team aware of recent changes to work items. Similar to the *Opened* view mode, a yellow border and background is shown around icons of work items that changed within the last 24 hours. Again, details are automatically displayed for recently changed work items, including the change made and who made the change. For simplicity, the same color coding is used for both *Opened* and *Changed* view modes in order to draw users' attention to work items with any recent activity.

The *Notes* view calls up the sticky notes for all work items with a note attached. The Notes view allows users to spot work items that need attention, for example if a team member makes a request to pair up on a specific task.

In order to provide continuous awareness and to allow passive use, WIPDash allows cycling through view modes and through team members within each view mode. A person remains selected for ten seconds before the next person on the team panel is selected. The next view mode is selected after cycling through all team panel selections. We were interested to see if this cycling behavior was useful or distracting to the teams we studied. The visualization updates with new or changed work items by querying Team Foundation Server (TFS) once per minute. All of the information used in our visualization came from the teams' data entries in TFS.

5 In Situ Deployment

We deployed WIPDash with two collocated Agile software teams and observed its use for one week. Our aim was to understand the usefulness of the WIPDash and the effect it had on team members' situational awareness and on group processes. Team A had eight members (seven male), with an age range of 22-46. Team B had 16 members (14 male), with an age range of 27-48. Individual roles on both teams included lead developer, developer, tester, test lead, program manager, writer and group manager. WIPDash was installed on a large display in the team rooms and on team members' individual workstations. Data were automatically collected in WIPDash in order to describe how participants interacted with the visualization throughout the study. WIPDash was installed on a Thursday and an orientation session was given the following day. We observed the teams the following week (Monday through Friday). Afterwards, we met with each team for post-usage discussions. Each participant received a \$50 gratuity coupon for their participation. In the Team A room, the awareness visualization ran on a projected wall display on the most accessible wall to the whole team. In the Team B room, the awareness display was installed on a 52-inch plasma touch screen toward the front and right side of the team room. Some members of Team B had their backs to this display.

5.1 Supporting Daily Stand-Up Meetings

Team A used WIPDash daily on their large display during stand-up meetings to coordinate meetings. Specifically, they found the *Iteration overview* useful, both in terms of status and also to jog their memories about work items from the previous day. During stand-up meetings, one of the team members selected each person from the team panel to display his information, and that person then talked about his work. The team considered details about the active items assigned to a person especially useful. Team members said they would have found it beneficial to have team members displayed in a random order during stand-up meetings—just to make it more fun. They also referred to the display to view the status of a remote team member when she called in for the standup meeting. This suggests that WIPDash could be useful for supporting collaboration with distant team members. Some team members commented that they liked to look at the awareness display first thing in the morning to see what team members in Argentina had been doing for the last ten hours. Some team members who were on vacation for most of the time during our study also said that they used the awareness visualization when they got back to get a sense of what the team had been doing for the past week and “*Where are we now?*” in the iteration.

We observed that the available information in WIPDash was not completely sufficient for reviewing what had been worked on during the previous day. Specifically, if a developer had completed his work on a work item and then reassigned the item to somebody else for testing, that item no longer showed in the *Changed* view for that developer. This was discussed during a stand-up meeting, and the team suggested a view where all work items that a person had worked on would be highlighted, even if they had been reassigned. The team further elaborated on an idea of one, concise overview containing all the information they would need for their

standup meetings, including recently resolved items that were reassigned to other members and items that members had worked on yesterday. Finally, the team expressed a wish for extending the shared wall display with an additional projector to show project information like spreadsheets or code next to WIPDash.

5.2 Automatic Cycling

The automatic cycling between views was found to be problematic. WIPDash cycled between all members in the team panel, including people with no items assigned or without any recent activity. Thus, nothing of interest is shown in parts of the cycle. Members of Team A suggested that cycling would make sense if done only between views that contain recent changes. Team B had only one work item assigned per person at a time and therefore cycling through each view for each team member was not useful.

Team members commented that notifications needed to be more assertive when a change or an update happened, such as an audio herald combined with a fisheye notification message about the change on the awareness display. Also, team members wanted to configure which views were displayed on their personal displays and when notifications should appear. An RSS-style feed would probably be a useful option for the personal workstations. We are pursuing that idea in the next iteration.

5.3 Use on Large Display and Individual Displays

We analyzed data logged by WIPDash to see how the two teams used WIPDash on the large display and on their individual machines. In all, we collected log data from ten personal machines, five members on Team B and five members on Team A, in addition to the two machines running the large shared displays. When a user started interacting with WIPDash, the default cycling between views was suspended. The log data showed that Team A used the large display (showing the iteration overview) during their stand-up meeting every morning around 9:30 am. Apart from the stand-up meetings, Team A physically interacted with the large display only twice during the study. In contrast, Team B interacted with the large display on average five times per day. In Team B some of the participants did not have WIPDash installed on their own machines. For this team WIPDash was running on a new touch screen display, which had a lot of appeal. One possible reason why Team A did not interact much with the large display was that it was projected high up on one of the walls, making personal display interaction more reasonable than going to the laptop in the corner of the room that controlled the view in order to interact with the group view.

5.4 Types of Interaction

In all, 596 selections were made in the right panel of WIPDash. Selections were primarily made in the team panel (66%). The view mode panel (17%), and iteration panel (15%) were also used to change the view. *Iteration overview* was the most

frequently selected view mode, selected more than 50% of the time. The *Changes* view and the *Opened* view were each selected between three and eight times, whereas the *Notes* view was only selected once.

Participants from Team A clicked on items to call up details 21 times while participants from Team B brought up additional details 22 times. Only two sticky notes were created in WIPDash, both by a lead developer on Team A. Follow-up discussions related to sticky notes revealed that users would rather use the existing ‘comments’ data structure in TFS to view and add notes to work items in WIPDash.

5.5 Questionnaires

Satisfaction with WIPDash was assessed using a questionnaire administered to team members at the post-usage meeting. The questionnaire contained nine questions from [13] and eight questions to address distraction and awareness, using five-point Likert-scale with lower scores reflecting negative responses. We balanced the valence of our satisfaction questions. For negatively phrased questions (marked with an asterisk in Table 1), we reversed the rating so that higher was always positive.

Ratings on usefulness and satisfaction with the system were mostly neutral to positive (see Table 1) and there were no significant differences between the two teams (paired t-test). Team members said that they had confidence that the information was displayed correctly and on time. From the ratings, it was clear that the notifications of changes were not grabbing attention well enough, and that WIPDash was seen as less reliable than we would have liked. However, the teams were not embarrassed to show their personal work item information and they leaned towards using a future version of WIPDash.

Table 1. User Satisfaction

Question	Average Rating (SD)
1. I have difficulty understanding WIPDash.*	4.10 (0.7)
2. WIPDash is easy to use.	3.63 (0.8)
3. WIPDash is reliable.	2.63 (1.1)
4. I have confidence in the information provided by WIPDash.	3.90 (0.7)
5. I need more training to understand WIPDash.*	4.05 (0.9)
6. WIPDash is informative.	3.32 (0.9)
7. WIPDash is comprehensible.	3.37 (0.8)
8. Overall, I am satisfied with WIPDash.	3.00 (1.1)
9. I would be happy to use WIPDash in the future.	3.11 (1.1)
10. I find WIPDash distracting.*	3.95 (0.9)
11. WIPDash grabs my attention at the right time.	2.84 (0.9)
12. It’s worth giving up the screen space to run WIPDash on my PC.	2.74 (0.9)
13. WIPDash helps me stay aware of information that’s critical.	2.72 (1.0)
14. I like being notified when a work item gets reassigned.	3.22 (1.2)
15. WIPDash’s notifications often distract me.*	3.83 (0.8)
16. Having WIPDash displayed in front of the team is embarrassing.*	4.17 (1.0)
17. I would rather have WIPDash displayed only privately.*	4.18 (1.1)

A questionnaire on situational awareness and group satisfaction was administered to team members before in situ deployment and at the post-usage meeting. The questionnaire included questions from [11,18] and used five-point Likert-scale with lower scores reflecting negative responses. Paired t-tests showed no significant differences between the Before and After conditions. The results could be affected by a particular iteration stage or the day of the week the questionnaire was completed.

6 Discussion, Lessons Learned and Conceptual Redesign

Several factors may have affected the use and adoption of WIPDash. First, size and location of the shared display affected how team members made use of the display by glancing at it or physically interacting with it. During observations of Team A, we saw that team members often looked at the display when entering or leaving the room. This was not the case in the Team B room. One reason may be that the Team A room had a large display, projected high up on a wall, that was visible to everyone in the room. In contrast, Team B's shared display was smaller, located in a corner of the room, and was not directly visible to all team members.

Second, the two teams in our study organized their work differently. Team A assigned work items to team members during an iteration planning meeting, and had daily stand-up meetings to follow up on progress of the team. Using the shared display during stand-up meetings may have influenced Team A's familiarity with the display, and consequently increased their use of the display. In contrast, Team B did not have daily stand-up meetings; instead, they talked with each other about progress and status throughout the day. Also, Team B assigned only one work item to each person and viewed work in terms of releases that span several iterations, not single iterations. This suggests that Team B could make even better use of an overview of activities, but using different time spans.

Third, Team B had many proposed items in their repository, but most items were not scheduled to be worked on. Also, many of the project areas, which were shown in WIPDash because they contained proposed work items, were simply not considered relevant by the team at the time of our observations. Thus, as Team B had many more items to track than did Team A, and since WIPDash showed all work items in 'proposed' state, the display for Team B was more cluttered.

Our study has limitations that should be considered when interpreting the results. Although very different work styles were observed, the two teams were from the same organization. Therefore, our results may not be generalizable to software teams everywhere. Also, we only observed the teams for one week. While we learned much from the initial feedback, it would be interesting to see the long term effects of WIPDash. Another concern is that we do not know how the collocated Agile teams that we have focused on compare with larger, distributed teams. It could be that a focus on collocated, Agile software teams may reveal some issues in team coordination that also apply to distributed software teams. For example, Gutwin et al. [8] suggest a need for awareness of areas of expertise within a distributed open source project team because developers work on all parts of the code. This might relate to the information needs we seek to provide with our visualization (e.g., what people are

working on and have been working on). For example, a glance at WIPDash first thing in the morning to see what team members in Argentina have been doing for the last ten hours could be very useful in terms of setting daily priorities or offering more timely assistance. We intend to extend our focus to distributed team awareness in our future work.

Informed by the insights we have gained from our study of WIPDash, the design of the next version of the dashboard will include the following:

- A more glanceable display;
- sound cues for users to look at the dashboard when information changes;
- on a user's display, it will only show notifications (e.g., 'toaster' type alerts in the system tray) as information changes;
- a wide diversity of project data field definitions and usages;
- support for add-on visualizations developed by a dashboard community;
- data caching to reduce the load on the repositories from dashboard clients.

Conclusion and Future Work

This study on WIPDash suggests benefits from providing awareness of work item status. Earlier work on FASTDash [1] showed benefits from providing a team situation awareness display based on code activity. An interesting perspective for future work is to combine information about work items with information about code activity in one visualization [14]. A potential disadvantage of such an approach is that the display gets too busy. However, linking code activity and the state of work items could give the team a solid shared context if it focused simply on what is or has just recently changed. That is the main goal for our next iteration. We intend to create a simple list of recently changed work items, and the people who are actively related to them. In addition, the new WIPDash will have integrated chat and RSS feeds for notifications to the desktop. The ethnographic study of Souza and Redmiles [17] confirms our observations: (a) a need for awareness of who made the changes; (b) a need to peripherally integrate awareness notifications into an existing work items repository, and to link them to code changes in order to keep work items' status better up to date and thus coordinate work more proactively.

In this paper, we have reported on a human-centered design approach to developing a situational awareness dashboard visualization to help software development teams track people and work items. From observations and interviews of development teams we learned about their current work practice and what might be provided to improve situation awareness. We then developed a dashboard for supporting awareness in teams, called WIPDash (Work Item and People Dashboard). We gathered feedback on an initial design from a focus group, which drove the detailed design and the implementation of WIPDash. Finally, we have studied the use of WIPDash in situ with two development teams, and reflected on the observations and data we gathered. While questions remain to be answered, the results from our study provide initial insights about use of a shared display to support team awareness of work item data in software repositories.

References

1. J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson (2007). FASTDash: a visual dashboard for fostering awareness in software teams. In *Proc. CHI '07*, 1313–1322, ACM.
2. M. Bruls, K. Huizing, and J.J. van Wijk (2000). Squarified Treemaps. In *Proc. TCVG '00*, 33-42. IEEE Press.
3. R. DeLine, M. Czerwinski, and G. Robertson (2005). Easing program comprehension by sharing navigation data. In *Proc. VL/HCC '05*, 241-248. IEEE Computer Society.
4. S. C. Eick, J. L. Steffen, E. E. Sumner Jr. (1992). Seesoft - A Tool for Visualizing Line Oriented Software Statistics. In *IEEE Trans. on Software Engineering*, 18 (11), 957-968.
5. J. B. Ellis, S. Wahid, C. Danis, and W. A. Kellogg (2007). Task and social visualization in software development: evaluation of a prototype. In *Proc. CHI '07*, 577–586, ACM.
6. G. Fitzpatrick, P. Marshall, and A. Phillips (2006). CVS integration with notification and chat: lightweight software team collaboration. In *Proc. CSCW '06*, 49-58, ACM.
7. J. Froehlich and P. Dourish (2004). Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proc. ICSE '04*, 387–396, IEEE.
8. C. Gutwin, R. Penner, and K. Schneider (2004). Group Awareness in Distributed Software Development, *Proc. CSCW '04*, 72-81. ACM.
9. C. W. Hill, J. D. Hollan, D. Wroblewski, and T. McCandless (1992). Edit wear and read wear. In *Proc. CHI '92*, 3-9, ACM, New York, NY, USA.
10. A. J. Ko, R. DeLine, and G. Venolia (2007). Information Needs in Collocated Software Development Teams. In *Proc. ICSE '07*, 344-353. IEEE Computer Society.
11. B. A. Olaniran (1996) A Model of Group Satisfaction in Computer Mediated Communication and Face-to-Face Meetings, *Behaviour and Information Technology*, 15 (1), 24-36. Taylor and Francis.
12. C. O'Reilly, D. Bustard, and P. Morrow (2005). The war room command console: shared visualizations for inclusive team coordination. In *Proc. SoftVis '05*, 57-65, ACM.
13. S. Paul, P. Seetharaman, and K. Ramamurthy (2004) User Satisfaction with System, Decision Process, and Outcome in GDSS Based Meeting: An Experimental Investigation. In *Proc. HICSS '04*, vol. 1. IEEE Press.
14. A. Sarma, D. Redmiles, and A. van der Hoek (2008). Empirical evidence of the benefits of workspace awareness in software configuration management. In *Proc. SIGSOFT '08/FSE-16*, 113-123.
15. K. Schwaber and M. Beedle (2002). *Agile Software Development with Scrum*. Prentice Hall.
16. C. de Souza, J. Froehlich, and P. Dourish (2005). Seeking the source: Software source code as a social and technical artifact. In *Proc. GROUP '05*, 197-206, ACM.
17. C. R. de Souza and D. F. Redmiles (2008). An empirical study of software developers' management of dependencies and changes. In *Proc. ICSE '08*, 241-250, ACM.
18. R. M. Taylor (1989). Situational Awareness Rating Technique (SART) The Development of a Tool for Aircrew System Design. *Proceedings of the Symposium on Situational Awareness in Aerospace Operations*, AGARD-CP-478.
19. S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson (2000). How does radical collocation help a team succeed? In *Proc. CSCW '00*, 339-346. ACM.