FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams

Jacob T. Biehl[1] **Mary Czerwinski**[2] **Greg Smith**[2] **George G. Robertson**[2]

[1]Department of Computer Science
University of Illinois
Urbana, IL 61801, USA
jtbiehl@cs.uiuc.edu

[2]Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
{marycz, gregsmi, ggr}@microsoft.com

ABSTRACT
Software developers spend significant time gaining and maintaining awareness of fellow developers' activities. FASTDash is a new interactive visualization that seeks to improve team activity awareness using a spatial representation of the shared code base that highlights team members' current activities. With FASTDash, a developer can quickly determine which team members have source files checked out, which files are being viewed, and what methods and classes are currently being changed. The visualization can be annotated, allowing programmers to supplement activity information with additional status details. It provides immediate awareness of potential conflict situations, such as two programmers editing the same source file. FASTDash was developed through user-centered design, including surveys, team interviews, and *in situ* observation. Results from a field study show that FASTDash improved team awareness, reduced reliance on shared artifacts, and increased project-related communication. Additionally, the team that participated in our field study continues to use FASTDash.

Author Keywords
Collaborative Programming, Awareness, Visualization, Field Study, Large Display.

ACM Classification Keywords
H5.3 Information Interfaces and Presentation (e.g., HCI): Group and Organization Interfaces

INTRODUCTION
As the complexity of software systems continues to increase in parallel with a growing economic dependence on these systems, there is an ever-greater need for software companies to produce reliable, bug-free products. A recent study by the National Institute of Standards and Technology (NIST) estimates that software defects cost the U.S. Economy $59.6 billion annually [2]. While the causes of software defects are wide-ranging, a significant proportion of errors are caused by a poor collective understanding of project status and fellow team member activities within the shared software project [14, 26-29].

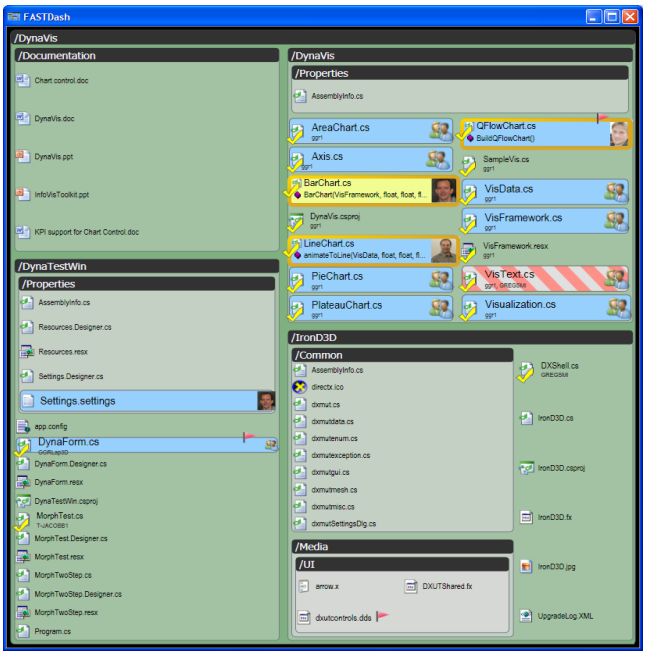

Figure 1: FASTDash showing three programmers working in a shared code base.

Unlike other collaborative jobs where workers can observe each other's activities as they physically move and perform actions within their shared workspace, programmers use electronic tools to operate in a shared virtual environment where other programmers' activities are difficult (or even impossible) to observe directly. This significantly increases the effort for programmers to gain and maintain an awareness of how their shared workspace is changing.

Many development teams are leveraging innovative methodologies such as Agile [31] and eXtreme Programming (XP) [13] to increase awareness and cooperation among programmers. These methodologies increase awareness and team interaction by employing shortened, highly iterative development cycles and co-

CHI 2007, April 28–May 3, 2007, San Jose, California, USA.

locating programmers in the same physical workspace. Yet, from our own interviews and surveys, we found that these programmers still lack adequate tools to support sufficient ongoing awareness of group members' actions. For example, a key piece of awareness information such as who is working with which source files is difficult to determine.

To assist software teams in gaining an ongoing awareness of team activities, we created a new visualization tool, called FASTDash (Fostering Awareness for Software Teams Dashboard, see Figure 1). Designed for project teams of 3-8 programmers, FASTDash enables immediate access to key information for group awareness. Such information includes which code files are changing, who is changing them, and how they are being used. By loading FASTDash on a large display in the shared workspace or on a second personal display (both of which are common in the environments we studied), programmers are provided a common, shared source for team activity awareness.

FASTDash was developed after a series of surveys, interviews and team observations, and then evaluated *in situ* with an actual programming team. In this field study, we found that FASTDash increased group awareness and that programmers found value in using the tool. In addition to the visualization tool, the contribution of this work includes: an underlying awareness reporting system which can be used to support a multitude of programming environments, repository systems and future visualization tools; a new observation coding scheme that can be more broadly used for studies of group behavior in shared workspaces; and results showing how a peripheral awareness display affects group programming behavior.

RELATED WORK

In this section, we discuss previous research on group programming practices, review existing tools for promoting awareness within software development activities, and situate our work within the broader arena of group awareness research.

Studies of Group Programming Practices

Many studies have been performed to better understand the work practices of software development teams. In their field study, Curtis et al. [14] investigated how design decisions were made, represented, communicated and updated throughout the development process. They found that teams face significant communication and coordination breakdowns attributable to the absence of appropriate communication tools that supported their work practices.

Similarly, Kraut and Streeter's [26] analysis showed that programmers lack effective informal communication tools, especially those that promote a team-wide view of a project's organization. Wu et al.'s [37] work also supports programmers' preference for communication tools that are lightweight. Gutwin et al.'s [22] interviews with distributed programmers found that explicit communication methods, like email and chat, were only effective when programmers made a significant effort to stay committed to those tools.

Singer [33] conducted a series of interviews with programmers across a variety of corporate and government software teams. Through these interviews, she found that programmers rarely use formal documentation, as it is too often outdated and inconsistent. The code base itself was found to be the de facto source programmers used to gain and maintain an understanding of program functionality.

More broadly, several studies have investigated the overall breakdown of a programmer's time. These studies report that maintaining an awareness of a project's status consumes a significant proportion of a programmer's time [27-29]; one study reports programmers spend as much as 40% of their work day communicating about code [27].

Our work builds on the findings and lessons of these previous studies. FASTDash specifically seeks to enhance awareness of other programmers' actions within the team's shared code base – their primary information source for project status. The tool is also lightweight, automatically collecting a broad range of activity information and providing this information in a single view. Programmers who have used FASTDash indicated that the tool improved their awareness of other developer's actions.

Awareness Tools for Software Development Activities

Several tools have been created to enhance an individual's awareness and understanding of large software code bases. SeeSoft [18] and its variants [24, 25, 32] provide a line-based view of the code. Each line of code is mapped to a thin row in a column which represents a code file; rows are colored to represent a particular attribute of the code line. For example, lines that have recently changed appear red. CodeThumbnails [16] provides miniaturized views of code files that can be arranged spatially. Highlights are shown on the sections of files that the programmer has open in the code editor. Programmers can click on the thumbnails to navigate within the code base.

TeamTracks [15] provides information about how fellow developers have navigated a code base in the past. This enables programmers to build on the actions of others, and to better understand and navigate the code themselves.

While these tools focus on improving understanding and navigation of a large code base (e.g., what files contain which classes and methods, or what sections of code have changed in the past), FASTDash focuses on providing a real-time awareness of *fellow team members'* activities within the shared code base (e.g., who has which file open, or what files are currently being edited).

Augur [19] provides a line-oriented view like SeeSoft, but combines activity information as one of the visualization attributes. Augur provides a per-line level view of activity, good for learning specific changes within a single or small set of code files. However, this line-oriented view has difficulty accommodating large code bases. In contrast, FASTDash's file-oriented view allows code bases of well over a hundred source files to be seen in a single view.

Similarly, Palantír [30] and Jazz [12] provide visualizations of programmers' actions within a source control repository (e.g., who is checking in a file). FASTDash includes source repository activity and more. With FASTDash, one can see fellow programmers' *ongoing* local file activities that occur outside of a source repository. For example, FASTDash allows developers to maintain an awareness of which files are opened (but not necessarily checked out) by a fellow programmers, which method or class a programmer is currently editing, or if a programmer is debugging code.

SeeSys [7] visualizes the historical evolution of changes for an entire code base. Using a space-filling representation of the code base (similar to a Treemap), the visualization highlights changes using animations of different code snapshots over time. In contrast, FASTDash is designed to allow programmers to quickly determine and maintain an awareness of the activities *currently* taking place within the code base and to aid programmers in coordinating their ongoing activities within a shared code base.

Group Awareness

Group awareness, the understanding of who you are working with, what is being worked on, and how your actions affect others, is essential to effective collaboration [17]. A common method for gaining and maintaining this awareness is through the use of shared artifacts [34].

Several tools have been created to convey status of shared artifacts. Tools like Community Bar [36] and others [3, 6, 9] allow collaborators to share views of artifacts currently open on their screens with other collaborators. Tools like SEAPort [10] extend the capabilities to gain artifact awareness when working within a physical workspace composed of multiple shared and personal devices. Systems such as CoWord [38] and Network Text Editor [23] enable users to collaboratively work inside a shared document.

FASTDash is different from these existing tools in two ways. First, it provides a holistic view of the entire shared workspace. Second, it does not replicate a collaborator's local workspace to remote users, but rather extracts and visualizes just the key individual activity information useful in an overview of workspace activity.

While designed for software development activities, in principle FASTDash is applicable to any work environment where many shared artifacts exist within the workspace, where those artifacts are usually accessed by multiple users, and where there is a need for collaborators to maintain awareness of how those artifacts are being used.

STUDY OF EXISTING PRACTICES

Following a user-centered design process, we started by conducting a survey and interviews with programmers to learn more about their awareness needs.

Motivation and Methodology

Building off the work of [20, 27, 37] and others, we sought specifically to understand the types of information used by programmers to establish group awareness, the methods used to gain and exchange this information, how often this information was consulted, how often it changed, and how those methods differed across different programming methodologies (e.g., traditional, Agile, SCRUM) employed at Microsoft Corporation, a large software company.

Our survey was distributed electronically using the corporate network. The survey contained over 30 questions about programmers' existing practices. Questions inquired as to how often programmers collaborate with fellow team members, what types of information they use to acquire awareness, and which existing tools are used for acquiring awareness. Most of the questions were free form response, while some were agreement/disagreement rating questions. 90 employees responded. Respondents' responsibilities included creating code, testing code, and/or project management. Respondents ranged across product teams, project sizes, and experience with different development methodologies. Respondents received no remuneration.

Thirteen of the survey respondents, comprised of both Agile and traditional programmers, were selected for one hour follow-up interviews. During these interviews we asked them to demonstrate or explain how they attain and maintain awareness of fellow programmers' actions. These interviews provided a contextual depth to complement the breadth of our survey results. Interviewees were provided a free dinner coupon for their participation. In addition, since Agile programmers reported that they often used large, projected displays in their environment, we chose to investigate this group further to understand how a tool like FASTDash might be integrated into their current practices.

Results

From our surveys and interviews, we found the following:

- *Programmers use a key set of information items to form an awareness of others' activities.* Overwhelmingly, we found that work items (e.g., which tasks are assigned to which people), bug descriptions, and the status of the shared code base (which files are checked out and being edited) to be the most common information items sought.
- *Programmers use many sources to acquire their key set of information items*. Programmers used both informal tools (e.g., whiteboards, sticky notes) and formal tools (e.g., bug databases, design documents). Supporting Singer [33], we also found source code and source code repositories to be a frequently used information source.
- *While most information items change on a weekly or monthly basis, the most often used are those that change on a daily, hourly or minute-by-minute basis.* We found that the key information items used to gain awareness are also the items that change frequently. Our interviews found that programmers would keep the tools for acquiring these key information items open, often configured to be visible on their screens.
- *Little differences in the methods used to gain awareness were seen across programming methodologies.* We

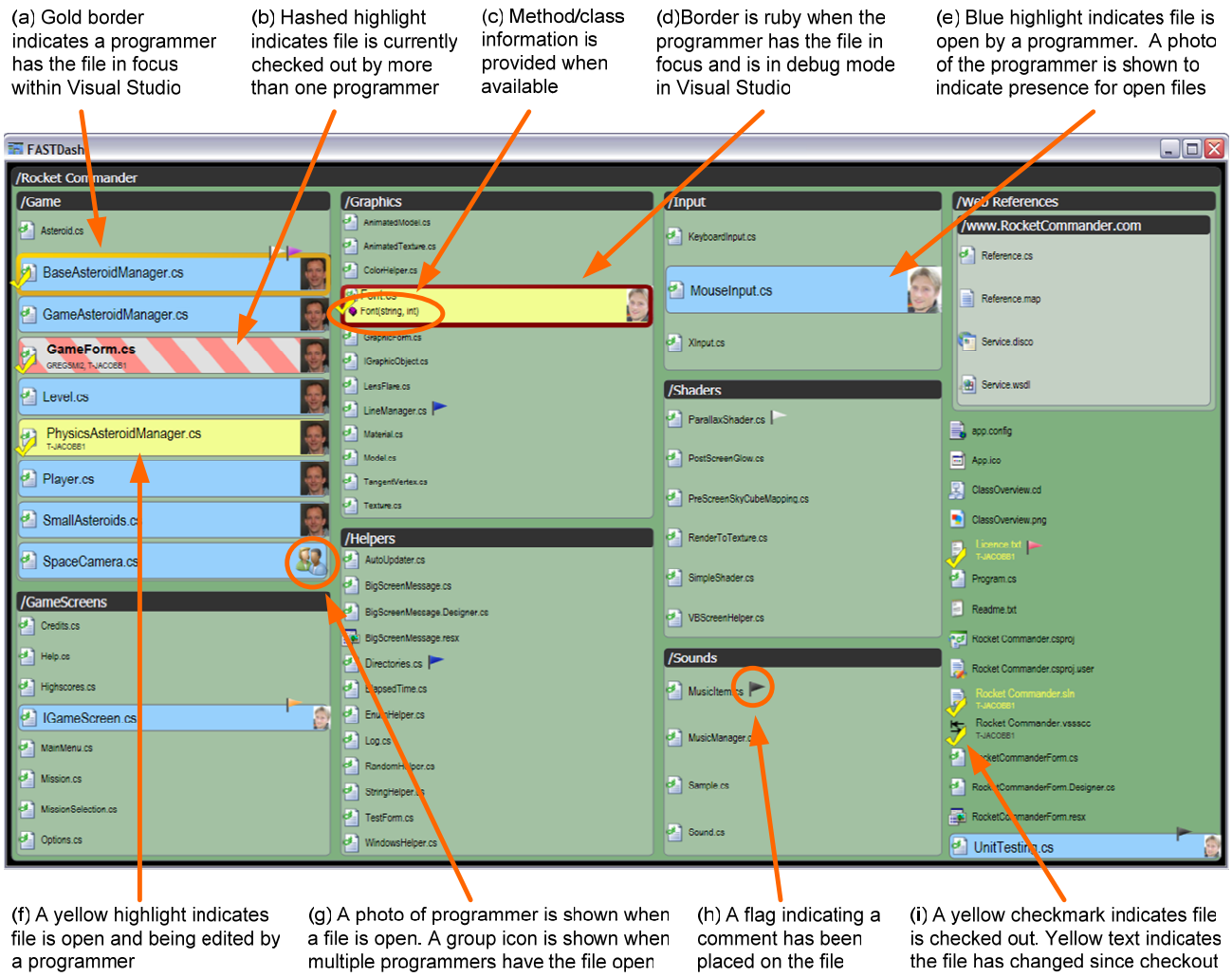

Figure 2: The FASTDash visualization runtime showing the active file and source code repository activities for a project with two active programmers. Call-outs explain individual features.

found that the teams following the Agile methodologies communicated verbally more often, and communicated more quickly and frequently when the default information gathering method failed or provided insufficient or conflicting information. Otherwise we found their methods and use of tools for gathering key information items to be consistent with the other programming methodologies.

- *Programmers saw great potential for large shared displays to be used for awareness, but their existing tools were not effective for this use.* Though we found the use of large shared displays to be sporadic, we did discover that when used, the information displayed and the communication that resulted were crucial to the ongoing, collaborative activity of the team. We also discovered that sharing information on the large display was cumbersome and tedious, since only one programmer's information could be viewed at a time.

Design Implications

Based on our interviews, we determined that programmers could significantly benefit from a tool that enabled quick acquisition and maintenance of workspace awareness. We specifically targeted source code activity as the principal information item that our tool would provide. While other information items were found to be important, we chose source code activity for three important reasons: 1) it was the information item that changed most frequently; 2) it was what programmers spent the most effort acquiring; and 3) it was the universal referent around which most of the *other* information items (work items, bug status, feature requests, and more) were focused.

FASTDASH

The dashboard is implemented in three components: the front-end visualization runtime, a plug-in for Visual Studio (a commonly used integrated development environment), and a SQL database. FASTDash also uses a source control management (SCM) system for source file data. The current implementation supports two existing SCM systems

(an internal system used at Microsoft and Team Foundation Server [4]), but the design of the plug-in allows it to be easily extended to support alternative repository systems (e.g., CVS [1] or SVN [5]).

Visualization Runtime

The visualization runtime provides programmers with several pieces of key information to enhance awareness of group programming activities (Figure 2). At the foundation is a spatial representation of the project's code base, implemented using a modified version of the Squarified Treemap algorithm [11]. Within the representation, folders are visualized as nested rectangles; the size of each rectangle is based on the number of files contained in the folder. Files are represented in the visualization by a textual label of the file's name and icon. Representing files and folders in this way allows the layout to remain spatially stable as items are added and removed from the code base.

Informed by our study of existing practices, we selected two types of development activities to overlay on the spatial representation: active file actions and source repository actions. *Active file actions* are based on the programmer's activity within his or her current Visual Studio session(s). File action information is useful because it helps convey the presence and interest of a programmer within areas of the shared code base. Active file information includes (with reference to Figure 2):

- Which files are open and by whom (***e*** and ***g***),
- Which files are currently being edited (***f***),
- Which file a programmer has in focus (***a*** and ***d***),
- What development modality (edit or debug) the programmer is in (***a*** and ***d***),
- If available and appropriate, which method or class the programmer is currently editing/viewing/debugging (***c***).

Source repository actions are based on the programmer's activity within the project's source repository system. Repository actions are useful because they allow fellow programmers to understand how the code base is currently changing. Source repository information includes:

- Which files are checked out and to whom (***f*** and ***i***),
- Which checked out files are different from the current version in the repository (***i***),
- Where there are potential checkout conflicts; e.g. where two programmers have the same file checked out (***b***).

In addition to the development activities, the visualization also allows programmers to attach comments to specific files. Markers for existing comments are shown as flags near the file's name (***h***). Programmers can view or edit a comment by clicking on its flag (Figure 3). New comments can be added by right-clicking on a particular file and selecting the Add Comment option.

The visualization runtime queries the SCM system at startup to build the source file hierarchy, and subsequently

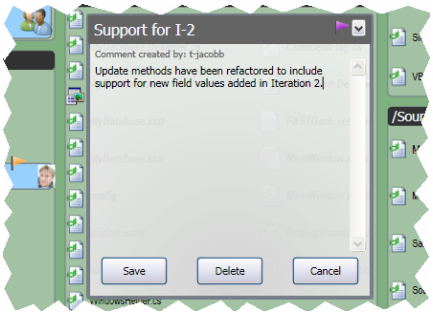

Figure 3: A comment box as it would appear when a programmer is viewing, updating or adding a comment.

begins polling the central SQL database for team activity. As an independent executable, the visualization runtime can easily be placed on a shared screen in a group's workspace (e.g., a hallway display or projected display in a common work area), on an individual programmer's workstation, or on a small peripheral display.

Visual Studio (VS) Plug-in and SQL Database

The Visual Studio plug-in is responsible for providing activity data to the central SQL server for use by the visualization runtime. It uses the standard VS plug-in API to monitor which files are open, the current development modality (edit or debug), which file the programmer is currently working in (file in focus), and which method or class he or she is working on. Each running instance of the plug-in maintains a connection to the central SQL database and reports its active files to this server, identifying itself by machine name, user, and process ID. Plug-ins also periodically poll the SCM system for file checkout status and use hash signatures of the local and repository files to determine if a checked out file has been changed.

Multiple instances of the plug-in for a given workstation cooperate so that only one takes responsibility for reporting these source repository actions to the central server at any given time. In order to keep all of the active file and source repository state consistent and current, the server maintains a list of reporting processes and periodically cleans up after processes that have unexpectedly stopped reporting. All of the plug-in's polling and server communication is done on a background thread to avoid interfering with the programmer's user interface experience.

FIELD STUDY

To better understand FASTDash's impact on users' awareness of fellow programmers' activities, we performed an observational study of a software team's collaborative behavior before and after the introduction of our tool.

An observational approach was chosen because we felt it better addressed our goal of gaining an overall understanding of how FASTDash affects group programming behavior. We felt that allowing programmers to work on their own projects, in their own workspace over a period of several days would provide more useful and representative data compared to a controlled lab study. Further, our *in situ* approach allowed us to understand how

the use of our tool might fit into the real-world methodologies and practices used by programmers.

Observation Coding Scheme

Many coding schemes exist for the capture and categorization of user behavior, and our investigation led us to consider three widely used schemes. Interaction Process Analysis (IPA) [8] enables experimenters to capture and categorize both task-oriented and social-emotional leadership roles within a group. Gutwin et al.'s mechanics of collaboration [21] codes the basic mechanisms of teamwork. Olson et al.'s work on analyzing collaboration [28] provides a reliable set of categories within which to classify activities that take place during design meetings.

However, the existing schemes did not cover the complete set of activities that we found to be essential to our study and that would be effective for our workspace and group configuration. Further, many schemes lacked classifications that could be easily quantified and compared across different conditions. As a result, we leveraged the earlier work to develop a new coding scheme (summarized in Table 1) that was a synthesis of the most relevant parts of existing schemes and our basic requirements.

Our coding scheme is divided into five main categories: communication, shared display use, shared artifact use, collaboration type, and collaboration configuration. *Communication* is broken down into eight categories, each representative of a different contextual class of communication. For example, an advice/instructional communication event would be coded when a programmer was heard providing instruction or advice on how to complete a task; agreement was coded when a programmer made a verbal agreement with the communication or action of another programmer.

Because our initial study of existing practices showed that inter-group communication rarely occurred outside of the verbal communication channel, we designed our coding scheme to focus on capturing verbal exchanges between co-located group members. For non-verbal communication behaviors (e.g., email or instant messaging), the collaboration type and configuration used were noted.

Shared display behaviors include categories for describing how a large projected or wall mounted public display is used, and what types of information are displayed on it. *Shared physical artifact use* refers to whiteboards, sticky notes, and other physical artifacts that are used as shared communication tools. When new artifacts were created, we classified these as *create* events. When an existing artifact was modified or updated, it was classified as a *modify/update* event. We coded any and all types of shared artifact use, including (but not limited to) design diagrams, code segments, to-do lists, and requirements.

Collaboration type and group configuration coding allowed us to determine the types of collaborations that were taking place within the group space, and how the developers configured themselves (i.e., whether or not the programmers could see all of the displays in which work was being performed) and their devices and tools for performing the shared tasks (i.e., what types of devices were used, and how they were controlled).

Category	Classification
Communication	Advice/instructional
	Agreement
	Collaboration request
	Disagreement
	Information/bottleneck
	Orientation/understand
	Status
	Other
Shared display use	Load information
	Connect device
	Transfer control
	Visual scan
Shared physical artifact use	Create
	Modify/update
	Deictic reference
Collaboration type	Co-located with shared visual workspace
	Co-located without shared visual workspace
	Distributed
Collaboration configuration	Multiple personal devices
	Single device, single control
	Single device, shared control
	Shared display only
	Shared and personal devices used

Table 1: Coding category and classifications used in our observation study.

While this coding scheme was created to satisfy our particular study requirements, it can be re-used in subsequent field studies that seek to learn more about how co-located users collaborate within a technology rich workspace, such as in a multiple display environment.

Methodology

Below we discuss the experimental methodology that we followed to evaluate the effectiveness of FASTDash and its impact on group programming behavior.

Subjects

We selected one product group composed of 6 experienced programmers from our previous survey respondents who volunteered to be observed. Individual roles included a program manager, a product manager, a documentation and usability expert, and 3 developers. The group had an average of 12.3 years of experience (SD=7) with 2.6 years on average tenure at Microsoft. Their average age was 35 (SD=6.6).

As a team that followed Agile development methodologies [31], they were an ideal group for a rapid field study. The group completes a full iteration on their code every week, which allowed us to observe and test our visualization within an entire development cycle. Team members were co-located in the same workspace (see figure 4), allowing us to code behaviors more easily and with higher accuracy.

Figure 4: Work environment for development team observed during the study. Note the location of FASTDash visualization runtime running on the team's large shared display.

Design

Our study was a pre/post observation design; we carried out two observations before the introduction of FASTDash and then two observations after it had been introduced. Before performing the actual observations, two researchers performed 170 minutes of pilot observation to allow for refinement and eventual validation of our coding scheme. After coding the pilot observation, the two coders discussed the coding results and iteratively adjusted the coding scheme until agreement was reached.

Through this pilot process, we found that the large number of dimensions within the coding scheme required that we use two independent coders to efficiently collect and transcribe the data. One coder was responsible for coding the Communication events while a second coded the events from the remaining categories. Coders periodically sampled each other's data to check for consistency with the agreed upon coding scheme. Assigning coding responsibilities in this way allowed us to balance the ability to code a large amount of behavior, while minimizing the disruption of having coders present in the subjects' workspace.

In addition to the coding scheme, we developed a pre- and post-visualization survey to collect subjective satisfaction data and gauge each programmer's situational awareness of the ongoing activities of fellow programmers. Among many existing measures, we used Taylor's classic situation awareness rating test (SART) [35] in our survey. Figure 5 summarizes the process that we followed in this field study.

Workspace configuration

The room was laid out as shown in Figure 4. All

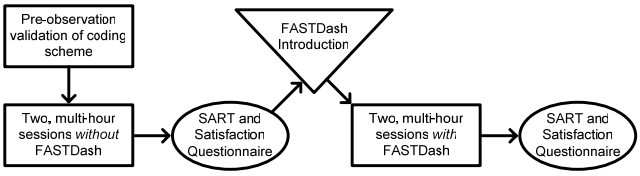

Figure 5: A high-level flowchart of the stages of our study.

TIME	ACTIVITY	SUBJECTS >>>>>				DETAILS/NOTES
14:16:55	COM-AGREEMENT	S3	S5			Discuss past wo
14:28:36	COM-STATUS	S3	S1	S2		announced leavi
14:31:13	COM-COLLABORATION REQUEST	S4				
14:32:15	COM-ADVICE/INSTRUCTIONAL	S2	S4			
14:32:43	COM-ADVICE/INSTRUCTIONAL	S1	S6			
14:32:43	TYP-CO-LOCATED|VISIBLE	S1	S6			
14:32:43	CNF-SINGLE DEVICE|SHARED CONTRO	S1	S6			
14:33:19	COM-INFORMATION|BOTTLENECK	S1	S6			
14:33:40	COM-STATUS	S1	S6	S4		
14:35:01	COM-STATUS	S4	S2			
14:36:07	COM-STATUS	S2	S4			
14:36:17	COM-ORIENTATION	S2	S6			
14:37:01	COM-ADVICE/INSTRUCTIONAL	S2	S3			
14:38:36	SART-CREATE	S2				
14:39:50	SART-MODIFY	S2				
14:40:14	COM-INFORMATION|BOTTLENECK	S1				
14:40:42	COM-INFORMATION|BOTTLENECK	S2				
14:41:27	COM-STATUS	S4	S2			
14:41:47	COM-INFORMATION|BOTTLENECK	S2	S4			Discusses reas
14:43:30	COM-ORIENTATION	S2				"Ready to go?"
14:44:21	TYP-CO-LOCATED|VISIBLE	S2	S1	S6		
14:44:46	COM-ADVICE/INSTRUCTIONAL	S2				

Figure 6: Segment of observation log.

programmers were located in the same physical space, configured in two rows of desks facing each other. All programmers had a dual-monitor configuration and state-of-the-art systems. In addition, there was a shared high resolution projected display that was located at the front of the room; which everyone in the room could view. This is where we placed the visualization when it was introduced, in addition to installing it on their personal workstations. There were also many whiteboard surfaces in the room.

Observations were carried out by two experimenters at a time, with three total experimenters used for the study. One coder was responsible for the communication events, while the other coded the remaining events, such as group configuration and shared artifacts. The coding scheme was implemented as an Excel spreadsheet which automatically time-stamped entries as the coder recorded them. Figure 6 shows a segment of the observation log from one of our observation sessions.

Observations were conducted over the course of four days; with each session lasting several hours. Two afternoon sessions were conducted before FASTDash was introduced (610 observation minutes) and two afternoon sessions with FASTDash running in the workspace (443 observation minutes). Between the two conditions, we installed and configured the visualization runtime and the VS plug-in on the programmers' machines. The team was given an overview of the features of the system and was encouraged to investigate and ask questions about the visualization's functionality. We gave the programmers about an hour to use the tool before we started our observations.

To reduce variability in our results, we coordinated with the team we studied to pick days and times where work tasks, group activities, and number of team members present in the room would remain relatively consistent across observation sessions. In our analysis, we did not observe any major variations across observation sessions.

Results

In the sections below we discuss the various results from our observational study.

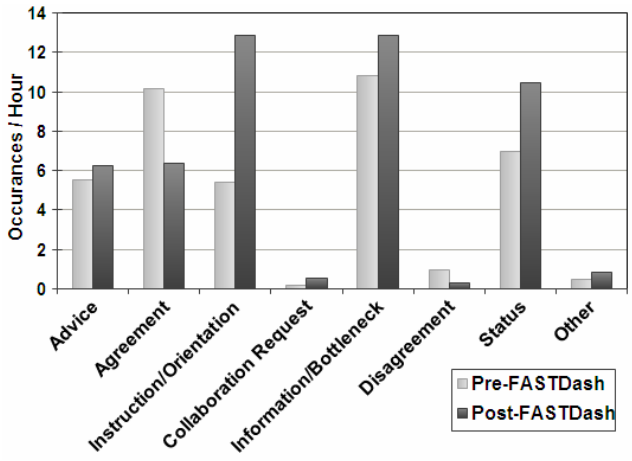

Figure 7: Communication category distribution for pre- and post- conditions.

Configuration and Use of FASTDash

In addition to having FASTDash displayed on the team's large shared display, the visualization was also installed on each programmer's workstation. We did not require the team to have the visualization running, but instead left it for each programmer to decide how to best display and use the tool. We believe this allowed us to gain a more realistic understanding of how programmers valued FASTDash as it did not change the team's existing practices.

Through the course of our observations we found that four out of the six programmers would consistently have the visualization running on their local workstation. Two programmers often configured the visualization to be displayed on their second monitor, while the other two would leave the visualization window open and bring it into focus occasionally to consult or use the annotation feature. The other two programmers often made visual scans of the shared projected display that showed the visualization. When asked about it in post-observation discussions, these programmers confirmed that they would glance at the display at natural breakpoints in their task.

Coded Behavior

Figure 7 shows the number of communication activity occurrences within the Communication category (see Table 1) for both the pre- and with-FASTDash conditions. Overall, the results show that programmers communicated more often with the FASTDash visualization present. Specifically, there was a 58% increase in instruction/orientation communications, a 15% increase in information/bottleneck communications, and 33% increase in status communications.

Use of shared physical artifacts (e.g., whiteboards, post-it notes) decreased considerably between the two conditions. Figure 8 shows creations were down 250%, modifications were down 117%, and references were down 160%.

The number and distribution of collaboration types was consistent across conditions. Overall, ¾ of collaborations were co-located with a shared visual workspace.

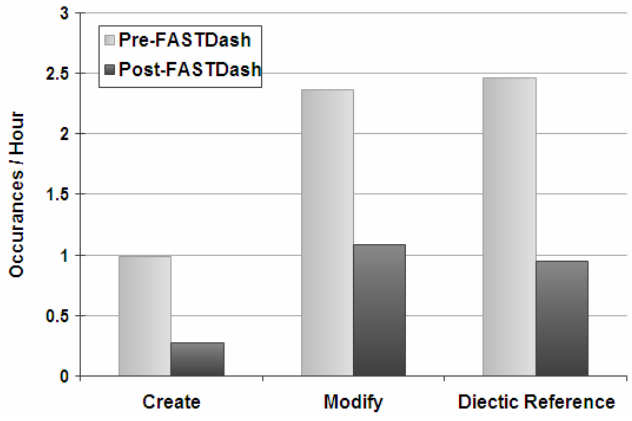

Figure 8: Use of shared artifacts.

We observed collaborations across all coded configurations except for the more elaborate configuration where both shared and personal devices are used in tandem to perform the task. The predominant configurations were the single device with shared control and the single device with personal control.

Overall, these are promising results. The communication increases may indicate that increased awareness from the FASTDash display motivated more interventions when and where needed. FASTDash also reduced the need for shared artifacts—this is promising because it could mean that the display was a useful summarization of team activity. Also, less reliance on shared artifacts may help un-clutter existing shared spaces and communication pathways.

Situation Awareness Rating Test (SART)

From the six participants in our study, we received four sets of completed pre- and post-visualization SART ratings. Two programmers' data were excluded from the analysis because they did not complete the survey immediately following each condition (we received their surveys four days after FASTDash was introduced).

We analyzed the ten individual questions and the overall SART category averages to get a better picture of the influence of the FASTDash visualization. All survey questions were on a scale of 1-7, with 1=less and 7=more. Paired Wilcoxon Signed Ranks tests for a question on Division of Attention showed a 30% reduction in attentional demands when the visualization was present (pre-FASTDash μ=4.0, SD=0; post-FASTDash μ=2.8, SD=0.6; z=-1.89, p=0.059). Subjects' ratings on the Instability of the Situation was also reduced by over 30% (pre- μ=3.0 SD=0; post- μ=2.0, SD=0; z=-2.0, p=0.046). These two individual questions showed the largest delta between pre- and post-surveys, though many of the other questions trended in a similar way. We felt it was encouraging we were able to obtain significant findings for such a small sample size and short-term use of FASTDash.

Satisfaction

FASTDash was generally positively received by the programmers who tested it. Five out of the six programmers

felt the tool improved their awareness of the actions of their fellow programmers. Three programmers said they were likely to continue using the tool while the remaining three were undecided.

Programmers appreciated having an entire view of the code base combined with activity information. One programmer commented "It is great to have all the files being displayed on the shared display along with the status. It makes it easier to verify if no item has been checked out by anybody else in the entire tree before making a build for the final release." Another commented that he "liked the real time info… and to know what developers are working on." Potential conflict detection was also well liked, as one programmer said, "The visualization of possible conflicts was the most useful."

The programmers also saw value in the ability to post comments into the visualization: "We usually make comments on several situations, but those 'verbal' comments are often lost. Placing [flags] with the comment on the context where it applies is cool."

Discussion

Overall, this evaluation of the FASTDash visualization was positive for a first iteration design. Our observations showed FASTDash was used by all team members and that it did indeed alter their behaviors, e.g. leading to decreased reliance on shared artifacts. Further, we found that the visualization increased intra-group communication. This was a surprising result since we anticipated the visualization would actually *decrease* the number of communication occurrences. Based on our observations and the comments we received from the questionnaire and post-observation discussions, we found evidence which suggests that the awareness provided by FASTDash enables communications to be more opportunistic and context relevant. An increase may therefore lead to greater efficiency for the team. We plan to investigate the impact of increased communication and productivity in a future long-term study with multiple teams. Such a study would also allow us to better understand which awareness cues in FASTDash are the most and least useful to programmers, and how the awareness cues complement the use of existing programmer tools.

The SART questionnaire showed situational awareness trends that suggest that the display decreased attentional demands in the environment and that the workroom seemed more stable. This could be indicative of reduced cognitive load, which could in fact lead to better decision making while programming. Again, this requires further experimental confirmation, and we intend to follow up with quantitative productivity measures in a controlled study.

Despite these promising findings we did observe several opportunities to improve the design of FASTDash. For instance, the space allocation algorithm in the visualization runtime is currently limited to considering only the file count in each folder. As we scale FASTDash to larger code bases, it may be more effective to use file activity itself as the metric, devoting more space to more active areas of the tree. We could then develop visualizations to represent aggregated activity in folders or subtrees that do not have room be fully displayed. Further, allowing the programmer to dynamically zoom and interactively navigate the visualization throughout the code tree, including exploring historical activity data, might prove to be valuable.

Finally, we plan to explore ways of extending the visualization to include work information items such as bug descriptions and build reports. With the addition of this information, programmers could more easily associate these items to files in the code base. This would provide a more complete single view of the team's activities, and establish a stronger overall awareness of project state.

CONCLUSION

We have presented FASTDash, a peripheral visualization designed to enhance team awareness while working on software projects. We have evaluated the design, and provided initial evidence that there is a significant contribution in the visualization's ability to improve programmers' sense of awareness and their feelings of stability of their environment. We also demonstrated that the visualization altered the developers' practices after using the visualization for only a short time. Further, the team of programmers who used FASTDash voluntarily continued to use it after the field study was complete.

In addition to these contributions, we presented a new coding scheme that can be used to quantify team work in real time. Other researchers in the HCI community can adapt and use the scheme for their studies, or extend it to be useful in other domains. For example, it could be useful while performing observational studies of multiple display environments and collaborative work.

Our work with FASTDash provides an important step towards understanding how to design interactive visualizations that can help groups acquire and maintain better awareness during collaborative programming and other problem solving activities.

ACKNOWLEDGMENTS

We would like to thank the Patterns & Practices Team for their willingness to allow us to observe them and insert FASTDash into their environment. We also thank Andrew Ko and Daniel Robbins for their help with our field study.

REFERENCES

1. Concurrent Versions System (CVS). http://www.nongnu.org/cvs/.
2. National Institute of Standards & Technology. The Economic Impacts of Inadequate Infrastructure for Software Testing. Planning Report 02-3. May 2002.
3. Microsoft NetMeeting. http://www.microsoft.com/windows/netmeeting/.
4. Microsoft Team Foundation Server. http://msdn.microsoft.com/vstudio/teamsystem/team/.
5. Subversion (SVN). http://subversion.tigris.org/.
6. Windows Live Messenger. http://messenger.live.com/.

7. Baker, M.J. and Eick, S.G. Space-Filling Software Visualization. *Journal of Visual Languages and Computing*, *6* (1995), 119-133.

8. Bales, R.F. *Interaction Process Analysis: A Method for the Study of Small Groups*. Addison-Wesley, Cambridge, MA, 1950.

9. Berry, L., Bartram, L., and Booth, K.S.. Role-Based Control of Shared Application Views. in *Proc. UIST,* 2005, 23-32.

10. Biehl, J.T. and Bailey, B.P. Improving Interfaces for Managing Applications in Multiple-Device Environments. in *Proc. AVI,* 2006, 35-42.

11. Bruls, M., Huizing, K., and van Wijk, J.J. Squarified Treemaps. in *Proc. Visualization*, 2000, 33-42.

12. Cheng, L.T., Hupfer, S., Ross, S. and Patterson, J., Jazzing up Eclipse with collaborative tools. in *OOPSLA Workshop on Eclipse Technology eXchange,* 2003, 45-49.

13. Cockburn, A. and Williams, L. *Extreme Programming Examined*. Succi, G. and Marchesi, M. eds., Addison-Wesley, 2001, 223-243.

14. Curtis, B., Krasner, H., and Iscoe, N. A Field Study of the Software Design Process for Large Systems. *CACM*, *31*, 11 (1988), 1268-1287.

15. DeLine, R., Czerwinski M., and Robertson, G. Easing Program Comprehension by Sharing Navigation Data. in *Proc. VL/HCC,* 2005, 241-248.

16. DeLine, R., Czerwinski, M., Venolia, G., Drucker, S., and Robertson, G. Code Thumbnails: Using Spatial Memory to Navigate Source Code. in *Proc. VL/HCC,* 2006, 11-18.

17. Dourish, P. and Bellotti, V. Awareness and Coordination in Shared Workspaces. in *Proc. CSCW,* 1992 , 107-114.

18. Eick, S., Steffen, J., and Sumner, E. SeeSoft: A Tool for Visualizing Line-Oriented Software Statistics. *IEEE Transactions on Software Engineering*, *18*, 11 (1992), 957-968.

19. Froehlich, J. and Dourish, P. Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. in *Proc. ICSE,* 2004, 387-396.

20. Gutwin, C. and Greenberg, S. The Importance of Awareness for Team Cognition in Distributed Collaboration. Salas, E. and Fiore, S.M. eds. *Team Cognition: Understanding the Factors That Drive Process and Performance*, 2004, 177-201.

21. Gutwin, C. and Greenberg, S. The Mechanics of Collaboration: Developing Low Cost Usability Evaluation Methods for Shared Workspaces. in *IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises,* 2000, 98-103.

22. Gutwin, C., Penner, R., and Schneider, K. Group Awareness in Distributed Software Development. in *Proc. CSCW,* 2004, 72-81.

23. Handley, M. and Crowcroft, J. Network Text Editor (Nte): A Scalable Shared Text Editor for the Mbone. in *Proc. SIGCOMM,* 1997, 197-208.

24. Hannemann, J. and Kiczales, G. Overcoming the Prevalent Decomposition in Legacy Code. in *Workshop on Advanced Separation of Concerns held at ICSE,* 2001.

25. Jones, J.A., Harrold, M.J., and Stasko, J. Visualization of Text Information to Assist Fault Localization. in *Proc. ICSE,* 2002, 467-477.

26. Kraut, R.E. and Streeter, L.A. Coordination in Software Development. *CACM*, *38*, 3 (1995), 69-81.

27. LaToza, T.D., Venolia, G., and DeLine, R.. Maintaining Mental Models: A Study of Developer Work Habits. in *Proc. ICSE,* 2006, 492-501.

28. Olson, G.M., Olson, J.S., Carter, M.R., and Storrøsten, M. Small Group Design Meetings: An Analysis of Collaboration. *Human-Computer Interaction*, *7*, 4 (1992), 347-374.

29. Perry, D.E., Staudenmayer, N.A., and Votta, L.G. People, Organizations, and Process Improvement. *IEEE Software*, *11*, 4 (1994), 36-45.

30. Sarma, A., Noroozi, Z. and Hoek, A.v.d., Palantír: Raising Awareness among Configuration Management Workspaces. in *Proc. ICSE,* 2003, 444-454.

31. Schwaber, K. and Beedle, M. *Agile Software Development with Scrum*. Prentice Hall, 2002.

32. Shonle, M., Neddenriep, J., and Griswold, W. AspectBrowser for Eclipse: A Case Study in Plug-in Retargeting. in*OOPSLA Workshop on Eclipse Technology eXchange,* 2004, 78-82.

33. Singer, J. Practices of Software Maintenance. in *Proc. Software Maintenance,* 1998, 139-145.

34. Tang, J.C. Findings from Observational Studies of Collaborative Work. *International Journal of Man-Machine Studies*, *34*, 2 (1991), 143-160.

35. Taylor, R.M. Situational Awareness Rating Technique (SART): The Development of a Tool for Aircrew System Design. *Situational Awareness in Aerospace Operations, AGARD-CP-478*, 1989.

36. Tee, K., Greenberg, S., and Gutwin, C. Providing Artifact Awareness to a Distributed Group through Screen Sharing. in *Proc. CSCW,* 2006, 99-108.

37. Wu, J., Graham, T.C.N., and Smith, P.W. A Study of Collaboration in Software Design. in *Proc. Empirical Software Engineering (ISESE),* 2003*,* 304-313.

38. Xia, S., Sun, D., Sun, C., Chen, D., and Shen, H. Leveraging Single-User Applications for Multi-User Collaboration: The CoWord Approach. in *Proc. CSCW,* 2004, 162-171.