Supporting Collaboration in Software Development with Discrete Event Streams

Jeremy Handcock and Gregory V. Wilson
University of Toronto
Department of Computer Science
10 King's College Road, Toronto, Ontario, M5S 3G4, Canada
jeremy@aperte.org, gvwilson@cs.toronto.edu

Abstract

Awareness of group activities and of changes to artifacts in a shared workspace is known to be a key component in successful collaboration. We implemented a tool called Aufait that aggregates and displays streams of discrete artifact modifications—or events—that occur in a shared software development workspace. In a three-week empirical study at two different organizations, we found that developers used event streams in Aufait to support well-established mechanics of collaboration. We also found that developers viewed different types of event streams disproportionately and that they were primarily interested in very recent events. Our results demonstrate that discrete event streams are a promising way to support collaboration in software development, and furthermore, our results pose important implications for the design of future tools.

1. Introduction

In software teams, developers maintain awareness of peer activities and discrete changes to shared artifacts by consulting numerous sources of information including source code, email, and bug reports [15]. Developers are known to maintain awareness of their projects regardless of whether they are in a co-located or distributed work environment [10].

A lack of awareness can contribute to delays and software quality issues [12, 2], and in response, researchers have proposed numerous tools designed to promote awareness in software teams. Many of these tools have a dashboard-like interface that conveys high-level project status through aggregate statistics of developer activity [8, 7]. Although dashboards can provide useful information [22]—especially to developers performing managerial tasks—few awareness tools are optimized to convey information about *discrete* project events. Consequently, there is little understanding about how awareness of these events can support collaboration. A discrete event is an atomic, individually distinct modification by a single developer to an artifact or group of artifacts in a software project. Examples of common discrete events include a change in the status of a bug report, an automated build failure, or a source code commit that might affect multiple files (*e.g.*, a changeset).

We developed a tool called Aufait that integrates *event streams* from multiple sources including source code repositories, bug tracking systems, team communication archives, automated build systems, and document repositories. It provides a visual timeline of discrete events and allows developers to obtain detailed information about peer activities and changes to shared artifacts.

We deployed Aufait in two different software development organizations and conducted an empirical study of how developers use discrete event streams to support collaboration within their teams. Our study involved a total of nine participants and we collected both quantitative and qualitative data over a period of three weeks by recording user interface actions and conducting regular, structured interviews. We found that developers used event streams in Aufait to support mechanics of collaboration that are well defined in a framework for groupware evaluation [9]. They primarily used the information provided by Aufait to monitor the state of the shared project workspace. They also coordinated actions, planned future work, communicated amongst themselves, and performed related tasks after becoming aware of events in Aufait. Developers were disproportionately interested in source code events and events that occurred within the past day. Our findings have concrete design implications for future tools and demonstrate that discrete event streams can play a useful role in supporting awareness and collaboration among software developers.

2. Background

The study of awareness and collaboration in software development has many of its roots in the field of computer-supported cooperative work, where it is widely accepted that awareness of peer activities is central to successful collaboration. Awareness provides essential context to an individual's contributions in a shared workspace to ensure that such contributions are relevant to and consistent with a group's overall activity [5].

Shared artifacts also play a crucial role in awareness. Participants in a collaborative process exert control over shared artifacts and receive feedback about their state as others modify them. Shared artifacts therefore provide a medium of indirect communication in a collaborative process: when one participant acts upon a shared artifact, others observe the effects of the action and thus are able to communicate through the artifact. This type of indirect communication is called *feedthrough* [4] or consequential communication [9]. In software development, discrete events such as individual source code commits combine information about peer activities and the state of shared artifacts as they contain information about the developer who initiated the change as well as the shared artifacts that he/she modified. In this sense, streams of discrete events represent a medium of indirect, feedthrough communication between developers.

2.1. Awareness in Software Teams

Multiple case studies in diverse settings confirm that software developers explicitly seek out awareness information during their work. In an observational study of co-located software teams, Ko *et al.* found that recent co-worker activity and recent artifact changes were among the most common information needs for developers [15]. They also identified specific sources of awareness information: bug reports, email, source code, developer tools, and direct communication with co-workers. Gutwin *et al.* found that geographically-distributed developers also maintain awareness, especially using project mailing lists [10]. Further, awareness information seems to play a role in coordinating dependencies and changes between team members [3].

Cases describing software project dysfunction when there is a lack of awareness among team members are plentiful. Damian *et al.* found that a lack of awareness contributed to consequences such as broken integration builds [2]. Moreover, Herbsleb and Grinter found that a lack of awareness in modular software development contributed to unnecessarily delayed integration periods [12].

2.2. Related Work

Researchers have proposed numerous tools to promote awareness in software development teams [21]. These tools typically exhibit a graphical dashboard interface that provides a high-level overview of project activity. The majority of tools focus solely on awareness as it pertains to source code, although some support awareness of changes to multiple artifact types. Augur provides a line-oriented view of source code changes showing relationships between software structure and activity [7]. CodeSaw offers an aggregate, retrospective view of source code commits and email communications [8]. Jazz provides configurable dashboards of team activity with information such as open work items [22]. Other tools are optimized for monitoring shared workspace activity in real time [1, 17] and for viewing open work items [13].

Few of these tools, however, are optimized to convey information about discrete events as in Aufait. Although Jazz supports feeds of automated build and work item events, how developers use feeds to support collaboration in Jazz has not been studied. Fitzpatrick *et al.* developed a simple discrete event awareness tool for source code commits [6], although their study examined only a handful of developers' interactions with the tool and they did not evaluate how the tool supports specific mechanics of collaboration.

3. Discrete Event Streams in Aufait

The default view in Aufait provides a visual representation of event streams that is partially inspired by the Simile Timeline [20]. As shown in Figure 1, discrete events are displayed in horizontal timelines that are stacked vertically. Events for different artifact types are displayed in different timelines. Each circle on a timeline represents an event and Aufait displays a short textual label beside each circle. For example, a source code commit event is labeled with a short summary of the comment that the author provided when checking in the code. Detailed information about an event, including how the artifact was modified, is available by clicking on the event in the timeline. Team members are displayed to the right of the timeline and each event is annotated with a color to encode the author of the event. Users can filter the timelines by selecting a team member, dragging to select events, and by keyword search.

In addition to the visual timeline, Aufait includes a more content-oriented view that is accessible by clicking on the 'Details' tab. The details view is similar in design to an email client or news reader in that it displays events in a list and details of a selected event are displayed in a panel.

Aufait can display events within any custom time interval. Users can view events within the last day or within the last week by using shortcut buttons. By default, the interface shows events within the last day.

4. Study Methods and Design

We deployed Aufait at two separate software development organizations and conducted an empirical field study

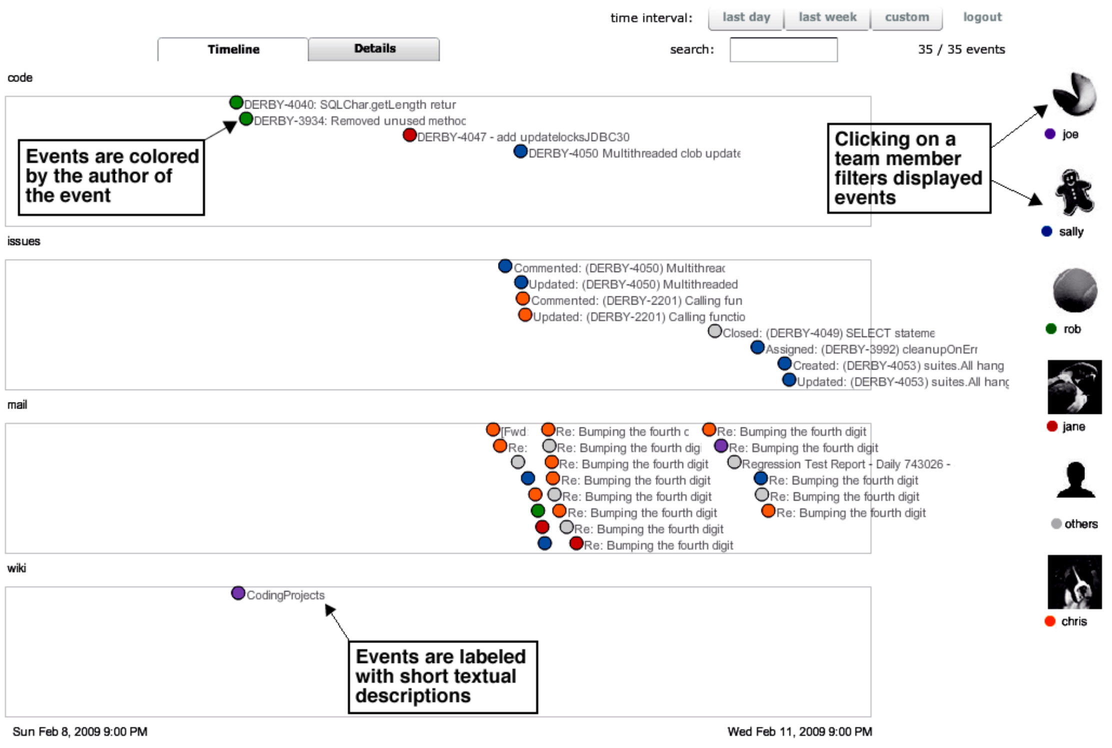
time interval:
last day
last week
custom
logout
Timeline
Details
search:
35 / 35 events
code
DERBY-4040: SQLChar.getLength retur
DERBY-3934: Removed unused methoc
DERBY-4047 - add updatelocksJDBC30
DERBY-4050 Multithreaded clob update
Events are colored by the author of the event
Clicking on a team member filters displayed events
issues
Commented: (DERBY-4050) Multithreac
Updated: (DERBY-4050) Multithreaded
Commented: (DERBY-2201) Calling fun
Updated: (DERBY-2201) Calling functio
Closed: (DERBY-4049) SELECT stateme
Assigned: (DERBY-3992) cleanupOnErr
Created: (DERBY-4053) suites.All hang
Updated: (DERBY-4053) suites.All hang
mail
[Fwd
Re:
Re: Bumping the fourth c
Re: Bumping the fourth dig
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Regression Test Report - Daily 743026 -
Re: Bumping the fourth digit
Re: Bumping the fourth digit
Re: Bumping the fourth digit
wiki
CodingProjects
Events are labeled with short textual descriptions
Sun Feb 8, 2009 9:00 PM
Wed Feb 11, 2009 9:00 PM
joe
sally
rob
jane
others
chris

Figure 1. The default timeline view in Aufait.

of how developers used it to support collaboration within their teams over a three-week period. We intentionally selected one organization with a co-located team and one organization with a geographically-distributed team to gain a broad understanding of how developers use discrete event streams across different environments. In both organizations, we instructed developers to use Aufait as much or as little as they found to be useful over the study period.

4.1. Case Study Venues

Zerofootprint [23] develops environmental impact management software for individuals and organizations. We solicited the five developers in software development roles at Zerofootprint and four agreed to take part in our study. The participants had an average of nine years of experience in professional software development and had been working together for less than one year. All four worked in a co-located setting and we gave them a live demo of Aufait before the study began. We integrated Aufait with Zerofootprint's internal systems and collected events for source code modifications, automated builds, bug reports, and project documents in a wiki.

Cytoscape [19] is a geographically-distributed, open source project that develops software for visualizing and analyzing complex networks, especially biological networks. Many developers from around the world contribute to the Cytoscape project and we solicited the ten most recently active—based on source code contributions—to participate in our study. Of those ten, seven agreed to take part; however, we only included the five developers who actively contributed code over the study period. The participants had an average of fifteen years of experience in software development and most had been working together for over three years. We provided them with a demo video describing the features of Aufait along with a brief user manual before the study began. In addition to the artifact types we supported at Zerofootprint, we collected email events from the Cytoscape project mailing list and displayed them in Aufait.

4.2. Data Collection

We instrumented Aufait to record all user interface activities in a usage log over the study period. On a weekly basis, we reviewed the usage log and segregated it into a series of *interactions* for each participant. We defined an interaction as a sequence of clicks in the user interface with less than two minutes of idle time between them. We were specifically interested in interactions where developers viewed specific discrete events or where they employed features of Aufait such as event filtering. We purposively sampled the usage log for such interactions and asked developers a series of structured questions about them via email each week. Herein, we refer to these samples as *interaction samples*.

We used the weekly email interviews to explore the detailed collaborative context behind the interactions: specifically, what motivated the developer to view certain events in the interface, what information did he/she take note of, and what actions (if any) did he/she take as a result of that information? We limited the email questions to a maximum of five per week to avoid placing an excessive time burden on participants. We used cues such as the time the interaction occurred, the textual content for events that participants selected, and links to artifacts in order to trigger participants' memories of the context. The following is an excerpt of an email to a Zerofootprint developer (Z1) with the response that we received in-line:

> *JH:* You accessed Aufait on Wednesday around 10am and looked at a code revision from Z2 (check-in message: "partial implementation of [feature]"). You then looked at the associated build event and followed its link to the build summary webpage [link]. Why did you select these specific events?
>
> *Z1:* I was reviewing possible check-ins to determine if Z2 had completed a commit. I'm not sure why I looked at the builds.
>
> *JH:* Did you take any action as a result of the information you saw?
>
> *Z1:* Started coding on the component that Z2 committed.

In addition to the weekly email questions, participants had the option to report the context behind an interaction through an open-ended form directly in the user interface. To motivate participants to provide rich information in their emails and self-submitted reports, we offered \$50 gift certificates to the two participants in each organization who provided the most detailed context throughout the study.

We also conducted short, semi-structured interviews with participants at the end of the study to gain a better understanding of participants' background, their work context, and to gather feedback on Aufait's design. During the interviews, we verbally administered a short survey to evaluate features of Aufait and its overall usability[1].

4.3. Categorizing Interactions

We grounded our analysis of the interaction samples in a conceptual framework for groupware evaluation proposed by Gutwin and Greenberg [9]. The framework is based on well-established mechanics of collaboration that have been observed in empirical studies and reported in previous literature. It has been widely cited and successfully applied in previous research [14]. According to the framework, collaborative work involves both *taskwork* and *teamwork* activities. Taskwork is the actual execution of a task while teamwork is "the work of working together." Teamwork is defined by specific mechanics of collaboration that groups perform collectively in order to carry out their work: monitoring the shared project workspace, communicating, coordinating actions, planning, protecting one's work from the actions of others, and providing assistance to others.

We used the mechanics of collaboration in Gutwin and Greenberg's framework to define a preformed coding scheme for categorizing each interaction sample based on its reported context. We categorized the email interview response or self-submitted report for each interaction sample using the constant comparison method [18] and the codes summarized in Figure 2. In some cases, we assigned multiple codes to an interaction where a participant viewed multiple events or had different motives for viewing events in a single interaction.

We made slight adjustments to our preformed coding scheme as we analyzed the data. We augmented the 'monitoring' mechanic with additional classes as we observed distinct variations in why developers monitored certain events. We omitted the 'assistance' mechanic from our coding scheme as we did not observe it in our data.

5. Results

Out of the 63 total interactions in the Zerofootprint usage log, we sampled 31 through weekly email interviews and self-submitted reports. We categorized six as 'Other', leaving 25 interactions for analysis. We sampled 37 of the 84 total interactions in the Cytoscape usage log. We did not receive a response from a participant for five interactions and an additional five fell into the 'Other' category, leaving 27 Cytoscape interactions for analysis.

5.1. Supporting Collaboration

We found that developers in our study used Aufait to support a number of collaborative mechanics defined by Gutwin and Greenberg's framework. As shown in Figure 3, our interaction categorizations demonstrate that developers in both organizations predominantly used Aufait to monitor their shared project workspaces. We discuss each collaborative mechanic that we observed in the interaction samples and present concrete examples of each mechanic below.

MONITORING. Participants had four distinct motives for monitoring events in our interaction samples: exploration, monitoring peer activity, monitoring the status of

[1]The complete survey results are available as an appendix at http://aperte.org/papers/aufait-survey.pdf

Category	Class	Description
Monitoring	Exploration	Monitoring events with no concrete motive
	Peer Activity	Monitoring activities of peers
	Artifact Status	Monitoring status of specific artifacts
	Impact	Monitoring events that impact one's work
Communication	n/a	Explicit communication between team members
Coordination	n/a	Coordinating actions between team members
Planning	n/a	Planning future actions
Protection	n/a	Protecting one's work from actions of others
Taskwork	n/a	Executing task after viewing event
Other	No Response	Participant did not respond to email
	Unknown	Participant could not recall context
	Testing	Participant was testing Aufait

Figure 2. Collaboration categories and classes used in coding interaction samples.

specific artifacts, and monitoring events that impact their work.

Developers in both organizations used Aufait to monitor and explore events in their shared workspaces without a concrete motive. Participants frequently viewed events simply because they were curious about what had happened. For instance, a Cytoscape developer viewed a document event because he was curious about what a colleague added to the wiki.

When monitoring peer activity, developers most commonly used Aufait to determine the activities of other developers that they did not have direct contact with. For example, Zerofootprint developers used Aufait to get updates on the activities of a contractor who worked off-site as well as for a team member who was only transiently involved in their project. Similarly, Cytoscape developers used Aufait to monitor the activities of remote developers. In both organizations, we found that participants in a leadership role used Aufait to monitor the activities of their subordinates.

Participants at Zerofootprint often used Aufait to monitor the status of artifacts that needed to be in a specific state before a task could be executed. For example, developer Z1 was interested in whether Z2 had checked in code that he depended on for developing a new feature. Further, Zerofootprint developers used Aufait to inquire about why an artifact was in a specific state, for example to determine the cause of an automated build failure. Cytoscape developers also used Aufait to monitor the status of artifacts. After becoming aware of a change to source code in Aufait, C1 and C2 reviewed the corresponding source code diffs to determine how the artifact state had changed.

Cytoscape developers used Aufait to monitor events in the shared workspace that impact their own work. For instance, C2 was working on a separate source code branch and used Aufait to determine how changes to code on the trunk impacted his work. Other developers examined the details of specific events when they determined that the event might impact their own work after examining the textual summary in the timeline.

COMMUNICATION. Viewing an event in Aufait triggered explicit communication actions among Zerofootprint developers on multiple occasions. In three of these instances, the nature of the communication was to orient a developer to the information that he/she viewed. For example, Z2 viewed a bug report update in Aufait because it contained requirements for a feature that he was assigned to implement. He then asked Z3 and Z4 to clarify the information in the bug report. In two other instances, Zerofootprint developers communicated about a build failure after noticing the status of the build in Aufait.

COORDINATION. In both organizations, developers coordinated actions amongst themselves after becoming aware of an event in Aufait. For instance, Zerofootprint participant Z2 noticed that a recent source code check-in from Z3 was a fix for a bug he had just reported. Z2 then resolved the bug as being fixed by Z3. Zerofootprint developers also coordinated actions to fix build failures after becoming aware of them in Aufait. Cytoscape developers C2 and C4 coordinated their pending source code changes with source code events from their peers after becoming aware of them in Aufait.

PLANNING. Participants in both organizations used the information provided by events in Aufait to plan future work. After becoming aware of a wiki modification in Aufait, Z2 used information in the wiki to plan the implementation of an upcoming feature. Cytoscape participant C5 used information contained in a wiki modification to prepare for an upcoming meeting. Similarly, C3 used information in a wiki modification to plan action items following a group retreat.

PROTECTION. Cytoscape developers sometimes took action to protect changes in their own workspace from in-

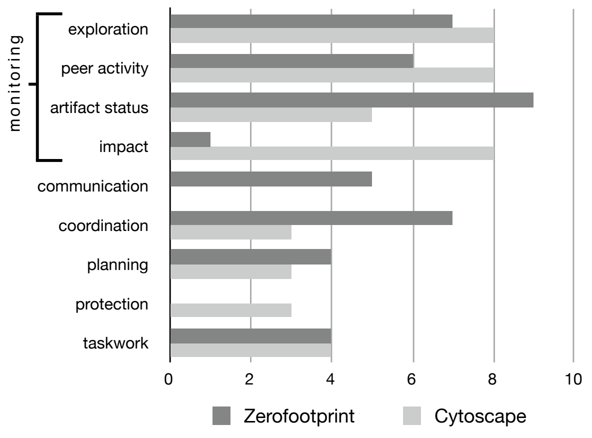

Figure 3. Counts of interaction samples by collaboration code.

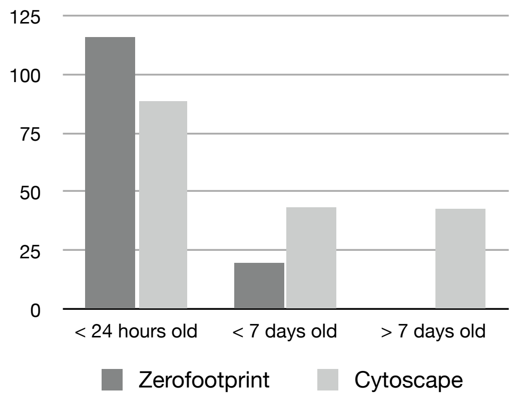

Figure 4. Counts of events viewed in Aufait by age of the event.

coming changes in the shared workspace. C2 was working on a branch, and after becoming aware of source code changes on the trunk using Aufait, he merged those source code changes into his branch to ensure they were compatible.

TASKWORK. On multiple occasions, developers in both organizations performed related taskwork after becoming aware of an event in Aufait. Zerofootprint developers took corrective action to fix a broken build and Cytoscape developers added new content to shared documents after becoming aware of changes to those documents in Aufait.

5.2. Properties of Events Viewed

We observed two distinct patterns in the events that developers viewed over the study period. Developers were overwhelmingly interested in very recent events versus older events, and furthermore, they viewed some event streams more often than others.

As previously described, users can click on events in Aufait to view details about how artifacts changed and who initiated the change. Although Aufait allows developers to select any historical time interval, we found they primarily viewed events that occurred within the past day, as shown in Figure 4. In fact, only a single Cytoscape developer viewed events older than one week and he did so only during the first few days of the study. Excluding the outlying events that were older than one week, the median age of events viewed among Cytoscape participants was 18 hours and 75% of events viewed were less than 36 hours old. Among Zerofootprint participants, the median age of events viewed was 10 hours and 75% of events viewed were less than 24 hours old.

According to our usage log, developers disproportionately viewed event streams for different types of artifacts. In both organizations, they clicked on and viewed source code events more often than others. Further, developers viewed a disproportionately high number of source code events relative to the total number of such events over the study period. At both organizations, source code events made up only one quarter of all events, however they made up approximately one half of all events that developers clicked on in Aufait. In a chi-square goodness of fit test, we found that the distribution of events viewed across artifact types in our study was significantly different from a uniform distribution of events viewed across all artifact types ($p < 0.01$ for both organizations). In other words, participants were not equally likely to view a document change event, for example, over a source code event.

6. Discussion

The key finding of our study is that discrete event streams as in Aufait support collaborative activities and mechanics of collaboration that are well established in previous research. Our findings also imply that the design of future tools could be enhanced by supporting the mechanics of collaboration that we found to be dominant in our study. We discuss these implications and provide concrete design recommendations below.

6.1. Monitoring Artifact Changes

In both organizations that we studied, developers frequently monitored changes to artifacts in the shared project workspace, for example to manage dependencies or to find events that might impact them. The frequency of this scenario is consistent with the findings of Ko *et al.* [15], who identified that the status of dependent artifacts was a common information need among developers. In order to satisfy this information need, we believe awareness tools should provide detailed descriptions of how an artifact has changed and emphasize events that are likely to impact a developer. For example, tools could highlight source code events affecting files that the developer has recently worked on. Further, to better inform developers about changes to dependent source code, tools should provide source code diffs. Multiple developers in our study suggested adding a code diff viewer to Aufait and all participants rated this proposed feature as being valuable or very valuable in our post-study survey. Tools should also provide a link so that developers can easily view modified artifacts, as participants in our study regularly used this feature of Aufait and rated it to be very valuable in our post-study survey.

Previous awareness tools such as Augur [7] focus solely on source code artifacts, and while developers were most interested in source code events in our study, they also monitored other types of artifacts. They monitored builds, documents, bug reports, and project emails when using Aufait, which suggests that future tools should incorporate more than just source code artifacts in their interfaces.

Some of the previously-proposed awareness tools were designed to support a retrospective, historical view of artifact changes. Specifically, CodeSaw [8] and Augur support exploration of artifact changes over long periods of time. As previously discussed, developers in our study most often viewed information about events that occurred within the last day and almost never viewed information about events older than one week. Although there may be isolated instances where developers seek information about historical events, our results suggest that future tools should be optimized to display recent events.

6.2. Monitoring Peer Activities

In addition to seeking information about artifacts, developers in both organizations frequently sought information about peer activity in Aufait. This usage scenario is also consistent with the study by Ko *et al.* [15], where the most common information need among a group of developers was peer activity information. This need suggests that annotating events with authorship information is an important feature in promoting awareness: such annotations connect an artifact modification to an individual. As previously described, Aufait colors each event on its timeline according to authorship and displays a photo of the author in the detailed content for an event. The results of our study suggest that this is an essential feature for future tools.

6.3. Supporting Communication

As previously discussed, we found multiple instances where Zerofootprint developers initiated communication with peers as a result of viewing information in Aufait. Biehl *et al.*'s study of FASTDash found that communication in a team increased significantly after the tool's deployment [1], suggesting that it may facilitate communication. The results of our study similarly suggest that discrete event streams may facilitate communication between team members.

Interestingly, we did not find any instances of explicit communication actions among Cytoscape developers using Aufait. Previous research on coordination in software teams has shown that developers often communicate informally during their work [16], but it is well known that there are barriers to informal communications between developers in distributed teams [11]. If tools could facilitate a developer to initiate informal communication at the time he/she becomes aware of an artifact modification, perhaps such communication would increase in distributed teams. For example, Aufait could display presence information for team members to facilitate instant messaging conversations about events.

7. Study Limitations

We studied how developers used Aufait within the context of real-world software development projects, thus we believe that our findings have high external validity. Also, we studied developers using Aufait in two very different organizations, which enhances the generalizability of our results to diverse software development settings.

Although we did not empirically evaluate Aufait's usability before deploying it at the organizations in our study, usability did not appear to be a major factor. Our survey results suggest that the lesser-used filtering features in Aufait may not have been easily discoverable, however these issues do not affect our major findings. Overall, our survey found that developers had no difficulty in using Aufait.

In analyzing the qualitative data from our interaction samples, a single researcher coded the interactions. Ideally, multiple researchers should code qualitative data to ensure accurate categorization. Even so, we feel that the relative magnitude of each categorization is correct.

8. Conclusions

Our goal in this study was to define how discrete event streams in Aufait can support collaboration in software development teams. We found that developers in our study used Aufait to support collaborative activities that are well established in previous research: monitoring the shared workspace, communicating, coordinating work, planning, protecting work, and performing taskwork. We also found patterns in how developers used Aufait that have important implications for the design of future tools. Specifically, tools should be optimized to display very recent events as well as to monitor the shared project workspace. Most importantly, our results demonstrate that discrete event streams are a promising way to support collaboration in software teams.

9. Acknowledgements

We thank the software developers and organizations that participated in our study. We also thank Jorge Aranda and Gina Venolia for helpful conversations throughout the investigation. The first author was a graduate student at University of Toronto while conducting this research.

References

[1] 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, NY, USA, 2007. ACM.

[2] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *ICGSE '07: Proc. of the Intl. Conf. on Global Software Engineering*, pages 81–90, Washington, DC, USA, 2007. IEEE.

[3] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. Sometimes you need to see through walls: a field study of application programming interfaces. In *CSCW '04: Proc. of the Conf. on Computer Supported Cooperative Work*, pages 63–71, New York, NY, USA, 2004. ACM.

[4] A. Dix. Computer supported cooperative work - a framework. In D. Rosenburg and C. Hutchison, editors, *Design Issues in CSCW*, pages 23–37. Springer Verlag, 1994.

[5] 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, NY, USA, 1992. ACM.

[6] G. Fitzpatrick, P. Marshall, and A. Phillips. Cvs integration with notification and chat: lightweight software team collaboration. In *CSCW '06: Proc. of the Conf. on Computer Supported Cooperative Work*, pages 49–58, New York, NY, USA, 2006. ACM.

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

[8] E. Gilbert and K. Karahalios. CodeSaw: A social visualization of distributed software development. *Human-Computer Interaction INTERACT 2007*, pages 303–316, 2007.

[9] C. Gutwin and S. Greenberg. The mechanics of collaboration: Developing low cost usability evaluation methods for shared workspaces. In *WETICE '00: Proc. of the Intl. Workshops on Enabling Technologies*, pages 98–103, Washington, DC, USA, 2000. IEEE.

[10] 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, NY, USA, 2004. ACM.

[11] J. Herbsleb and R. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *ICSE '99: Proc. of the Intl. Conf. on Software Engineering*, pages 85–95, New York, NY, USA, May 1999. ACM.

[12] J. D. Herbsleb and R. E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software*, 16(5):63–70, 1999.

[13] M. Jakobsen, R. Fernandez, M. Czerwinski, K. Inkpen, O. Kulyk, and G. Robertson. Wipdash: Work item and people dashboard for software development teams. *Human-Computer Interaction INTERACT 2009*, pages 791–804, 2009.

[14] H. Johnson and J. Hyde. Towards modeling individual and collaborative construction of jigsaws using task knowledge structures (tks). *ACM Transactions on Computer-Human Interaction*, 10(4):339–387, 2003.

[15] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *ICSE '07: Proc. of the Intl. Conf. on Software Engineering*, pages 344–353, Washington, DC, USA, 2007. IEEE.

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

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

[18] C. B. Seaman. Qualitative methods. In F. Shull, J. Singer, and D. I. Sjoberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 35–62. Springer, London, 2008.

[19] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research*, 13(11):2498–2504, Nov. 2003.

[20] Simile timeline, 2010. http://www.simile-widgets.org/timeline.

[21] M.-A. D. Storey, D. Čubranić, and D. M. German. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In *SoftVis '05: Proc. of the Symposium on Software Visualization*, pages 193–202, New York, NY, USA, 2005. ACM.

[22] C. Treude and M.-A. Storey. Awareness 2.0: Staying aware of projects, developers and tasks using dashboards and feeds. In *ICSE '10: Proc. of the Intl. Conf. on Software Engineering (to appear)*. ACM, 2010.

[23] Zerofootprint software, 2010. http://www.zerofootprint.net.