Developer Dashboards: The Need For Qualitative Analytics

Olga Baysal, Reid Holmes, and Michael W. Godfrey
David R. Cheriton School of Computer Science
University of Waterloo, Canada
{obaysal, rtholmes, migod}@cs.uwaterloo.ca

***Abstract*—Prominent tech companies including IBM, Microsoft, and Google have embraced an analytics-driven culture to help improve their decision making. Analytics aim to help practitioners answer questions critical to their projects, e.g., "Are we on track to deliver the next release on schedule?" and "Of the recent features added, which are the most defect prone?" by providing fact-based views about their projects. Analytic results are often quantitative in nature, presenting data as graphical dashboards using reports and charts. While current dashboards are often geared towards project managers, they are not well suited to help individual developers. By analyzing Mozilla developer interviews we noted that developers face challenges maintaining a global understanding of the tasks they are working on and that they desire improved support for *situational awareness*, a form of qualitative analytics that is difficult to achieve with the current quantitative tools.**

In this article, we motivate the need for qualitative dashboards designed to improve developers' situational awareness by providing task tracking and prioritizing capabilities, presenting insights on the workloads of others, listing individual actions, and providing custom views to help them manage their workload while performing their day-to-day development tasks.

I. Software Analytics in Practice

Many organizations have adopted data-driven decision-making processes and technologies. Embedding analytics into an organization's culture can be used to enhance competitive advantage [1]. Analytic approaches strive to provide actionable real-time insights; these insights are often presented as quantitative multi-dimensional reports. However, analytic approaches can be both quantitative and qualitative in nature. While quantitative analytics can highlight high-level trends of the data, qualitative analytics enable real-time decision making for tasks that are lower level and more frequent.

Most analytics approaches focus on quantitative historical analysis, often using chart-like dashboards (see examples in Figure 1). These dashboards are often geared towards helping project managers monitor and measure performance, for example, "to provide module owners with a reliable tool with which to more effective manage their part of the community" [2].

David Eaves, a member of the Mozilla Metrics team who worked on community contribution dashboards (a quantitative analytic tool), states: "we wanted to create a dashboard that would allow us to identify some broader trends in the Mozilla Community, as well as provide tangible, useful data to Module Owners particularly around identifying contributors who may be participating less frequently." Figure 1 demonstrates quantitatively-oriented dashboards for the IBM Jazz development environment (left) and the Mozilla Metrics project (right). The Jazz dashboard gives high-level charts describing various project statistics, e.g., how issues have been open and closed (bottom left), and how various test cases execute (bottom middle). The Mozilla dashboard provides a quantitative description of various community contributions, including a graph at the top showing where these contributions were made and a table that describes per-developer statistics below. While both of these dashboards effectively convey information about specific aspects of the project, they would likely be more applicable to managers than to individual developers as they perform their daily tasks.

Our own study (described in Section III with the complete results available in [3]) suggests that current dashboards poorly support developers' day-to-day development tasks. Developers require a different kind of dashboard to improve their situational awareness of their tasks. Ehsan Akhgari, a Mozilla developer, says, "What I really want is a dashboard to show the important stuff I need to see — review status on the dashboard, assigned bug with a new comment or change since the last time I looked at it, bugs that have a certain status could go on the dashboard, etc." [4].

By definition, a qualitative property is one that is described rather than measured. While quantitative dashboards provide statistical summaries of various development metrics, qualitative dashboards emphasize the attributes and relationships of a set of artifacts of interest to a developer. Thus, qualitative dashboards provide developers with means to define, organize, and monitor their personal tasks and activities on the project. Unlike quantitative dashboards that address developer questions such as "How many bugs are pending on me?" or "How well did I perform last week?", qualitative dashboards provide insights into the specific items developers are working on: "You look at the bug and you think, who has the ball? What do we do next?" (P7) and "What has changed since the last time I have looked at it?" (P6). Developers want to be able to get information based on what has changed since the last time they looked at it. Being able to keep abreast of the volume of changes taking place on active development teams can be challenging; by helping developers focus on the evolution of their issues, and those they are interested in, they can better prioritize their own tasks.

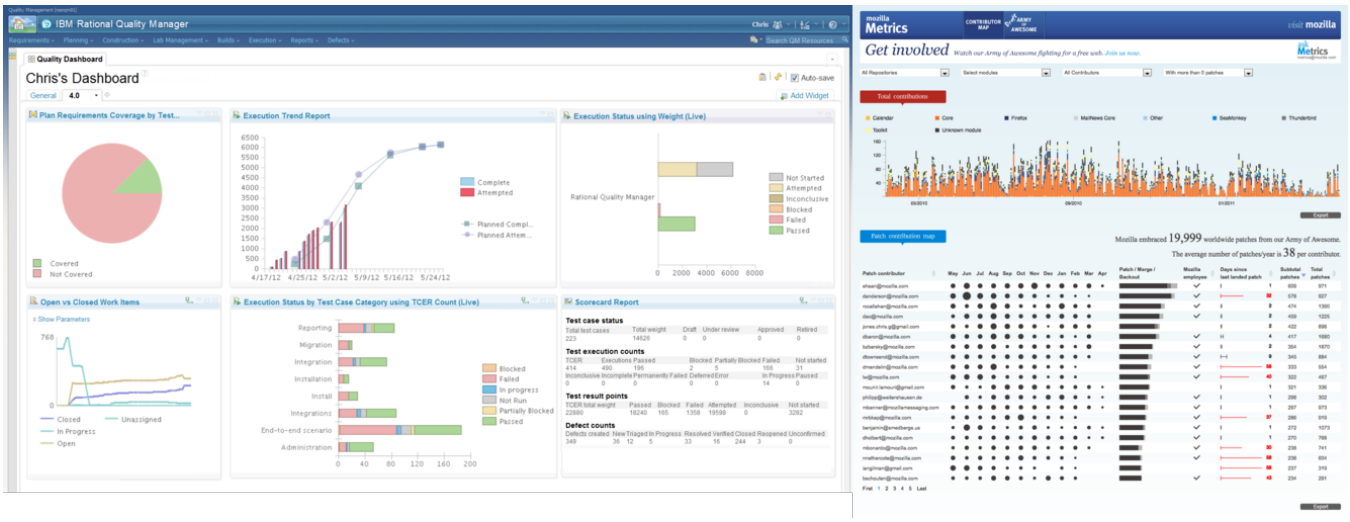

Fig. 1. Quantitative dashboards: Jazz (left) and Mozilla Metrics (right).

II. Awareness in Software Development

The problem of maintaining developer awareness on the projects and tasks has been previously recognized by the research community. Cherubini et al. [5] looked at how and why developers use drawing during software development. Treude and Storey [6] investigated the role of awareness tools such as Jazz dashboards and feeds in supporting development activities. Further more, Fritz and Murphy [7] studied how developers assess the relevancy of these feeds to help users deal with the vast amount of information flowing to them in this form.

A number of tools have been developed to assist developers with daily tasks and activities [8]–[12]. FASTDash [8] offers an interactive visualization to enhance team awareness during collaborative programming tasks. The workspace awareness tool Palantir [10] follows a similar approach by providing insight into workspaces of other developers, in particular artifact changes. Mylyn [9] is a task management tool for Eclipse that integrates various repositories such as GitHub, Bugzilla, JIRA, etc. It offers a task-focused interface to developers to ease activities such as searching, navigation, multitasking, planning and sharing expertise. Yoohoo [11] monitors changes across many different projects and creates a developer-specific notification for any changes in the depend-upon projects that are likely to impact their code. Similarly, Crystal [12] increases developer awareness of version control conflicts during collaborative project development. While these tools provide insight into the current collaborative activities on a project, we motive the need for a custom view of the project to help developers maintain awareness of their own working context.

III. Research Study

To understand how developers engage and interact with the Bugzilla issue tracking system, we performed a qualitative study of interviews with 20 core Mozilla developers [4]. The study captures developers' insights into the strengths, weaknesses, and possible future enhancements of the Bugzilla platform, the primary collaboration platform for Mozilla developers. We applied a grounded theory methodology on the set of interveiws as we had no predefined themes or categories. We first created all of the 'cards', splitting 20 interview transcripts into 1,213 individual units; these generally corresponded to individual cohesive statements, which we call comments. In further analysis, the first two authors (coders) performed two rounds of independent card sort reporting the intercoder reliability (i.e., degree of agreement) to ensure the integrity of the card sort. We calculated average scores for four most popular reliability coefficients for nominal data: percent agreement (98.5%), Scott's Pi (0.865), Cohen's Kappa (0.865), and Krippendorff's Alpha (0.865). On average, two coders agreed on the coding of the content 98.5% of the time.

Through open coding we identified four high-level themes, among which situational awareness emerged as one of the major missing pieces developers requested from Bugzilla (19 of the 20 developers provided 208 quotes in support of issues surrounding situational awareness). The complete results of the qualitative study are available online [3].

We found that developers face challenges maintaining awareness of the status of their own issues (18 developers), for example:

> *"[I maintain a] gigantic spreadsheet of bugs I am looking at. It would be useful to know how the bugs have changed since I last looked at to track if any work was done [on them]"* (P11).

In addition to their own issues, several developers expressed the desire to be able to non-obtrusively observe the evolution of other issues without being forced to take an active role (15 developers):

> *"You don't want to CC yourself on every bug you triage"* (P15).

Developers also wanted to easily gain an understanding of

their colleagues' workloads, for instance when requesting code reviews (12 developers):

> *"If you could see how many reviews are pending on a person on that list, this would be a better way to load balance reviewers. Would be good to have an easy way to click on the name in some way and jump on their review queue to see if they have a lot of easy or hard issues to look at"* (P15).

Finally, developers found it challenging to assess other developers' roles in the Mozilla organization (12 developers):

> *"People profiles – you should be able to know more about them, how long they have been in the system, what is their ranking, are they module owners or peers. We need to know who we are talking to. We need some way to figure out who you are so that we can treat each other better. We depend on people so much and Bugzilla is all about bugs not people"* (P15).

We believe that the data developers seek is often available in the tools they currently use, it is just not accessible in a format that is amenable to the tasks they are trying to perform. Just as quantitative dashboards can be generated from a project's issue tracker, so can developer-specific qualitative dashboards.

IV. What Is Situational Awareness?

Situational awareness is a term from cognitive psychology referring to a state of mind where a person is aware of the elements of their immediate environment, has an understanding as to their greater meaning, and can anticipate (or plan to change) these elements in the near future [13]. The term is used in engineering, for example, to describe how air traffic controllers work as they track and route air traffic; it is also an apt description of how software developers must maintain awareness of what is happening on their project, as they manage a constant flow of information and react accordingly.

Developers often find themselves trying to identify the status of an issue — What is the issue waiting on? Who is working on it? What are the workloads of others? Who is the best person to review a patch? — as well as trying to track their own tasks — How many bugs do I need to triage, fix, review, or follow up on? Which issue should I work on next?

Our study suggests that supplementing quantitative dashboards with more developer-specific qualitative data can improve developer situational awareness of their working context. This awareness will enable developers to keep better track of the ever-increasing number of issues involved in complex software systems. From our own experience in analyzing the data available in Bugzilla, we also believe the data required to build developer-specific qualitative dashboards already exists; much like quantitative analytics, the data just needs to extracted, analyzed, and presented in the right format so developers can easily make use of it.

V. Qualitative Dashboards: Task-Oriented Views

Qualitative dashboards can enhance a developer's daily activities, such as issue tracking and prioritization, patch submission, personal workload management, colleague workload estimation, and maintaining lists of issues that need follow up, etc. While most of the data necessary to generate qualitative dashboards is present within the Bugzilla database, it is not easily accessible by the average user. There are several reasons for this: First, the Bugzilla user interface is notoriously unintuitive and poorly designed to support process management, presenting too much information to the user and little direction; this was noted by seven developers, for example, "The Bugzilla interface is bad, too many fields" (P15). Second, Bugzilla's slow performance hinders real-time exploration of the data; "the speed of Bugzilla is the major issue" (P14), "Bugzilla is too slow, this is wasting a lot of time, very frustrating" (P6). Third, developers are often unable to correctly formulate queries to access and correlate various pieces of metadata; "running searches on Bugzilla is kind of scary sometimes" (P9), "querying in Bugzilla is hard; he has to spend a few minutes to figure out how to do the query... no good way to query certain information" (P6).

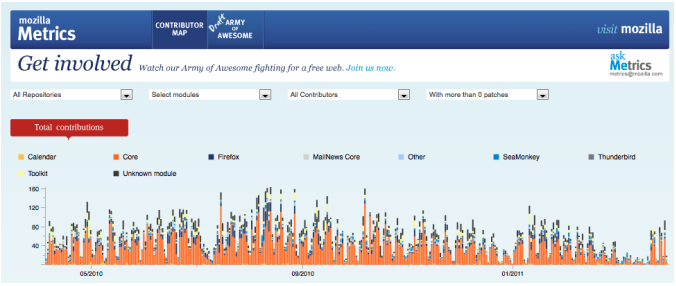

Fig. 2. Mozilla Community Metrics: total contributions

Mozilla has applied quantitative analytics through two initiatives: first, to gain insights on the evolution of the community contributions (Community Management Metrics) [2] and second, to analyze the project's performance (Bugzilla Anthropology) [4]. These initiatives have developed a series of dashboards that use historical information to monitor community contributions (Figures 2–4) or to measure and track trends in bug fixing efforts (Figure 6). These dashboards support tasks such as monitoring patch contributions and identifying bug trends (status- or priority-based). However, they are tailored towards project managers and their activities; they do not provide developers with any useful support for their typical daily tasks. We now describe the four tasks current dashboards provide support for and, based on them, we motivate the need for a complementary qualitative approach that is geared towards helping developers with the decisions they must make on a daily basis.

1) *Assessing community effectiveness and evolution*

The current Mozilla community management dashboard provides insight into community patch contributions. Figure 2 shows the number of contributions submitted to various Mozilla modules; contributors can be sorted based on their involvement on the project (employees, volunteers). This view is useful for identifying

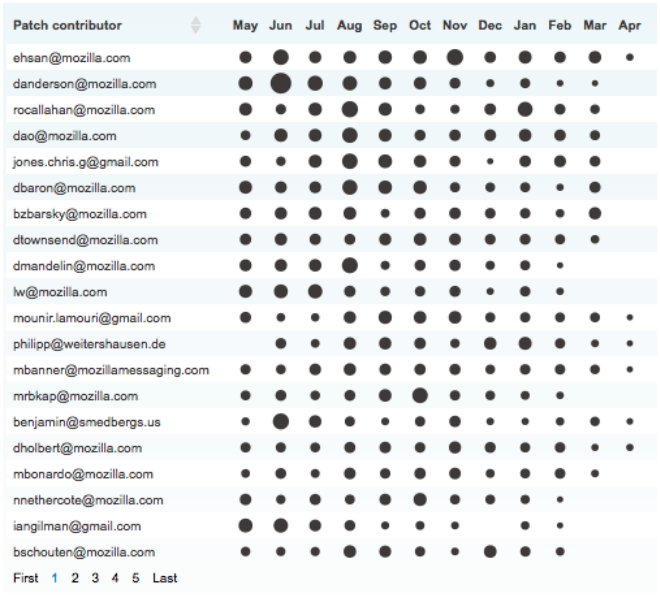

Fig. 3. Mozilla Community Metrics: patch contributions per developer

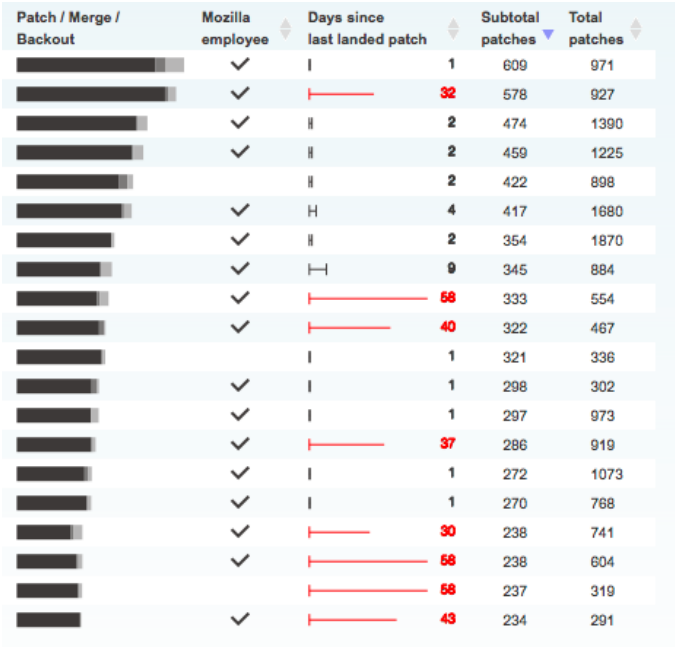

Fig. 4. Mozilla Community Metrics: developer effectiveness

broader trends in the Mozilla Community and assessing how much various members contribute to the project.

While this dashboard is designed to serve Mozilla module owners to more effectively gauge and manage their contributors and to support their strategic decisions around community engagement, developers are unlikely to benefit from the aggregate statistics of the received contributions.

2) *Measuring developer contributions*

Figure 3 illustrates a developer's patch contributions for a given month. While this information is useful to the project managers to explore developer activity, it does not provide concrete support for the developers' daily tasks. At the same time, our study found that developers desired better transparency on the work loads of others, which largely leverages the same data as the quantitative graph. Developers wanted this data in order to **determine who is the "right" reviewer** to request a review from ("it's hard to know when you request a review, which of these five people has time" (P8)). The right person to send a patch for review to may be either the one having faster review turnaround or having a shorter review queue. To identify the most suited reviewer for approving a patch, developers need to be informed about reviewers' workloads and average response time. While review queues are frequently used "to see who might be quickest" (P17), Bugzilla poorly supports this task ("you can check queues one at a time but it's a lot of work" (P8), "the current system punishes the guy who is the most responsible or that is doing the best job" (P15)). Apprising developers about the review queues of the key reviewers for a module is one way of balancing their work loads.

Being able to see workloads of the reviewers is particularly important if a developer is not familiar with the module/component reviewers ("When submitting a patch for another component, it's more difficult, he has to try to figure out who is good in that component, look up their review info" (P8)). Knowing what others are working on can enable assigning more relevant tasks to people as well as enabling better load balancing.

Developers also wants to be able to **track their own patch activity**, as well as determine which patches are awaiting reviews or who is blocking their reviews. Figure 5 illustrates a partial view of the qualitative dashboards that enable developers to track their patches (first tab called `Patches`).

Presenting the list of submitted patches and sorting them by last touched date can help developers stay aware of the recent changes on their patches such as new comments, review flag changes, reassignment of the patch reviewer, etc. In addition, the status of a patch needs to be more transparent, for example by displaying the name of the reviewer the patch is pending on.

3) *Measuring developer effectiveness*

Quantitative dashboards are often used to monitor developer productivity. Figure 4 demonstrates the productivity of a developer by providing details on how many patches he contributed, how many of them successfully landed into the project's code base, and how recently the developer contributed to the project (days since last landed patch).

Our study suggests that developers find it difficult to determine what has happened since the last time an issue was examined (as noted by 12 developers). Bugzilla

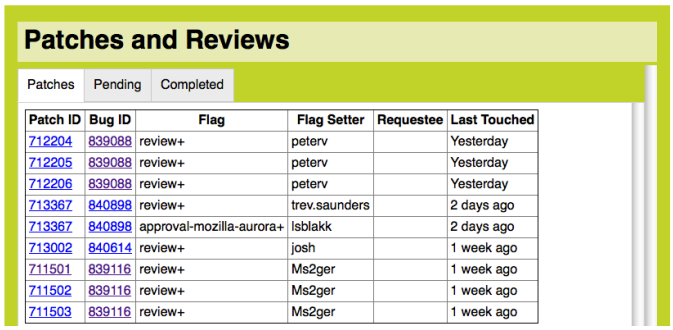

Fig. 5. An example of the qualitative dashboards supporting patch tracking and code review tasks.

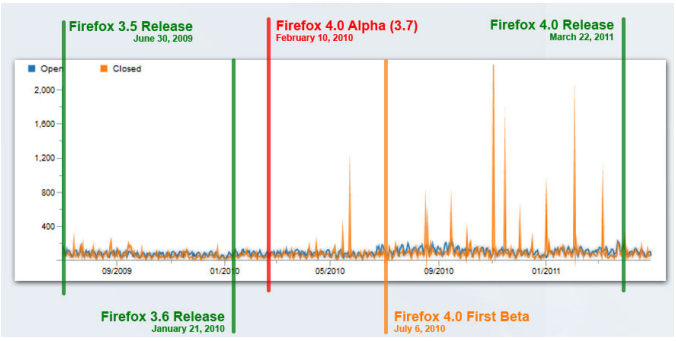

Fig. 6. Daily open vs close bug change in Firefox over time.

makes it hard for developers to **track their tasks such as bug fixing or code review**. Developers want to be able to see the list of issues they need to fix, review, or follow up by having task specific views ("It would be cool if Bugzilla people could be assigned with views that would setup good defaults for the task that they are working on" (P10)). As shown in Figure 5, the dashboard displays current (`Pending` tab) and past (`Completed` tab) code review tasks to increase developer awareness of what tasks are blocking others or what issues were recently resolved.

Qualitative dashboards such as these can organize information the way developers want, i.e., by displaying issues developers report, follow, need to resolve, listing patches submitted for review, as well as discussions on the issues (posted comments). Ordering issues by last touched date allows developers to better monitor recent changes on their tasks and activities.

4) *Determining performance trends*

Figure 6 shows a quantitative view displaying bug trends by status (open vs. close) and by priority (high vs. low). This view can be helpful for managers who are trying to get a sense of project momentum while leading up to various deadlines or to assess the overall health of the project.

Again, our study found that developers are interested in status, but in a different way: they want to be able to quickly determine how their bugs have evolved recently. Developers want to be informed by the changes to the issues relevant to their work: "you want to get info based on what has changed since the last time you looked at. You wouldn't need to rely on bug mail, it's too easy to miss things in bug mail, could be embarrassing" (P6).

Private and public watch lists can help developers **to organize the large number of emails** they need to filter. Public lists display issues that a developer needs to resolve or wants to monitor (being a module owner or QA). Developers often want to track issues they are interested in "if there is something interesting, I will CC self on it" (P7). These lists also allow developers to communicate interests on issues without taking ownership, as developers "would like to have a personal list of bugs without claiming them" (P8).

While public lists are visible to anyone on the project, private lists are created by the developers themselves. As one of the developers puts: "we need a way for people to set their own priorities on bugs, so a team has one, and product has another, and security yet another, each one of the initiatives should be represented. If it's P1 for all 3 then you know it's a big deal" (P15). Prioritizing issues and daily tasks can improve developers' time management by accomplishing important tasks first.

Qualitative analytics can be organized around custom views of the Bugzilla repository supporting ongoing situational awareness on what is happening on the project. The qualitative approach is based on filtering important and relevant information from the repository and presenting it to the developers supporting their common tasks such as bug fixing, feature implementation, code review, triage, etc. Qualitative analytics enable accessing the data from the existing issue tracking system to enhance Bugzilla with the means to increase developers' situational awareness of the project.

VI. Summary

Quantitative dashboards are the norm within the growing field of software analytics; they aggregate different kinds of development information to support high-level business and management tasks. Qualitative dashboards, on the other hand, aim to provide rich and detailed support for lower-level development tasks; they do so by filtering and contextualizing information extracted from issue tracking systems such as Bugzilla. The situational awareness afforded by qualitative dashboards can help developers better manage the constant influx of data surrounding the evolution of the technical issues that concern their project, enabling them to better prioritize their efforts while performing their day-to-day development tasks.

Developer-oriented qualitative dashboards do not aim to supplant quantitative approaches; both kinds of dashboard exist to provide specialized views into the incredible wealth of information available concerning how modern software systems are developed and maintained.

REFERENCES

[1] S. LaValle, E. Lesser, R. Shockley, M. S. Hopkins, and N. Kruschwitz, "Big data, analytics and the path from insights to value," *MIT Sloan Management Review*, vol. 52, no. 2, pp. 21–31, 2011.

[2] D. Eaves, "Developing community management metrics and tools for mozilla," April 2011. [Online]. Available: http://eaves.ca/2011/04/07/developing-community-management-metrics-and-tools-for-mozilla/

[3] O. Baysal and R. Holmes, "A Qualitative Study of Mozillas Process Management Practices," David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, Tech. Rep. CS-2012-10, June 2012. [Online]. Available: http://www.cs.uwaterloo.ca/research/tr/2012/CS-2012-10.pdf

[4] M. Best, "Bugzilla anthropology," March 2012. [Online]. Available: https://wiki.mozilla.org/Bugzilla_Anthropology

[5] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2007, pp. 557–566. [Online]. Available: http://doi.acm.org/10.1145/1240624.1240714

[6] C. Treude and M.-A. Storey, "Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, 2010, pp. 365–374. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806854

[7] T. Fritz and G. C. Murphy, "Determining relevancy: how software developers determine relevant information in feeds," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011, pp. 1827–1830. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979206

[8] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "Fastdash: a visual dashboard for fostering awareness in software teams," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2007, pp. 1313–1322. [Online]. Available: http://doi.acm.org/10.1145/1240624.1240823

[9] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *Proc. of the ACM-SIGSOFT Intl. Symposium on Foundations of Software Engineering*, 2006, pp. 1–11.

[10] A. Sarma, Z. Noroozi, and A. V. D. Hoek, "Palantir: Raising awareness among configuration management workspaces," 2003, pp. 444–454.

[11] R. Holmes and R. J. Walker, "Customized awareness: Recommending relevant external change events," in *Proc. of the ACM/IEEE Intl. Conf. on Software Engineering*, 2010, pp. 465–474.

[12] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Crystal: Precise and unobtrusive conflict warnings," in *Proc. of ESEC-FSE Tool Demo*, 2011.

[13] M. R. Endsley, "Toward a theory of situation awareness in dynamic systems: Situation awareness," *Human factors*, vol. 37, no. 1, pp. 32–64, 1995.