DASHboards: Enhancing Developer Situational Awareness

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

ABSTRACT

Issue trackers monitor the progress of software development "issues", such as bug fixes and discussions about features. Typically, developers subscribe to issues they are interested in through the tracker, and are informed of changes and new developments via automated email. In practice, however, this approach does not scale well, as developers may receive large volumes of messages that they must sort through using their mail client; over time, it becomes increasingly challenging for them to maintain awareness of the issues that are relevant to their activities and tasks. To address this problem, we present a tool called called DASH that is implemented in the form of personalized views of issues; developers indicate issues of interest and DASH presents customized views of their progress and informs them of changes as they occur.

Video: `http://youtu.be/Jka_MsZet20`

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.2.6 [**Software Engineering**]: Programming Environments

General Terms

Design, Human Factors

Keywords

Developer dashboards, Bugzilla, Elasticsearch, situational awareness

1. INTRODUCTION AND BACKGROUND

Issue trackers play a central role in software development; they are used for a number of collaborative project-related activities, such as reporting and fixing bugs, discussing how outstanding concerns might be resolved, and reviewing proposed patches. Bugzilla is a well-known issue tracking tool that is used by a number of projects, including Mozilla. However, Mozilla developers have previously expressed concerns that Bugzilla is slow and poorly designed to meet many of their day-to-day needs [2].

Typically, developers who use Bugzilla keep abreast of changes in their working environment in one of two ways: they either write custom queries to run on Bugzilla, or they create custom filters in their mail client to sort through the deluge of email that Bugzilla typically sends them. Neither of these is ideal: since the amount of information stored in Bugzilla grows monotonically over time, the already-slow searching mechanisms supported by Bugzilla become increasingly overwhelmed. At the same time, the volume of email that a typical Mozilla developer must be able to process can approach 50 to 500 message per day; developers have reported that it is increasingly hard to identify the truly important information from this daily deluge [2].

In principle, many common questions that developers have — such as *Who is reviewing my patch? What patches are pending reviews and do I need to ping someone?* — can be answered by querying the issue tracking system. In practice, this is problematic: Bugzilla's performance is often painfully slow, and its query interface includes many fields that are irrelevant much of the time while at the same time lacks the expressiveness to form queries that developers seek answers to. For example, developers often wish to know what has changed in an issue since the last time they queried, or if they are currently blocking progress in some issue. Such information is challenging to get from Bugzilla since it requires writing a complex request multiple times, e.g., for every product a developer working on.

Our tool, DASH, aims to offer developers customized views of issue tracking, and in so doing allow them to retain better awareness of the key issues that are involved in. Its development originated after conducting a qualitative study [1] on the data collected by the Mozilla Anthropology project [2]. This project was started in late 2011 to explore how various Mozilla community users make use of the Bugzilla issue tracking system, and to gain a sense of how Bugzilla could be improved in the future to better support the rapidly-growing Mozilla community.

During this process, Martin Best of the Mozilla Corporation conducted 20 one-hour interviews with active developers from various Mozilla projects. These interviews included developers' insights on their interactions and experience with the Bugzilla issue tracking system. The main goal of the Anthropology project was to identify trends that could help locate key problem areas with issue management, as well as best practices related to the use of Bugzilla.

Our qualitative analysis of the interviews revealed that developers seek improved support for situational awareness about issues they are involved with and the changes relevant to their daily tasks such as fixing bugs, making patches, conducting code review jobs and following up on the issues of their interest [1].

2. AWARENESS SCENARIOS

We describe four scenarios to highlight key tasks Mozilla developers perform every day in working on their projects.

Bug fixing — Developers need to know what issues have been assigned to them, so they can track their work items. Most developers start their day by determining if any issues are currently pending on them to resolve: *"How many issues do I need to fix?"* Another question developers often ask is *"What has changed since yesterday?"* Currently, Bugzilla can provide developers with a list of issues assigned to them, but cannot provide insights into the context of the change.

Following up on issues — Developers sometimes wish to track issues that they may not be working on directly; they can do this by subscribing to the issue and by participating in discussions.

Tracking patches — One of the main activities developers are involved with is writing patches: code modifications that fix defects. Developers want to be able to track their own patch activity, as well as determine which other patches are awaiting reviews or who is blocking their reviews.

Reviewing patches from others — Developers want to ensure that they are not blocking others needlessly, so they want to be quick at reviewing patches from other developers. They also want to be able to quickly assess their own review loads: *How many patches are pending on me? Who am I blocking?*

3. APPROACH

Mozilla developers can deal with these scenarios in several ways: they can use Bugzilla's built-in query mechanism; however, this has poor performance and is also not well designed for these tasks. They can use the bug email facility, and manage the results with hand-designed filters in their email clients; however, this is tedious, and important results can get lost amid the volume of details. And they can create an add-on tool, as we have done and will describe.

3.1 Bugzilla approach

While Bugzilla is widely used and deployed within many organizations, it is designed to track the issues and their changes. Since Bugzilla concentrates on issue tracking rather than informing developers on how issues evolve during project development, developers face challenges in keeping aware on what is happening on the issues they are involved with.

Indeed, Bugzilla includes a web-based search engine that allows users to construct a variety of queries on issues by specifying a list of filters. However, creating a custom query can be surprisingly time consuming as there are numerous metadata fields that can add complexity. For example, to answer the seemingly simple question *"Which bugs are assigned to me?"*, one needs to set six fields to appropriate values. For the query related to code reviews on patches *"Which patches do I need to review?"*, a developer would need to use a custom search option and understand how review flags are defined.

After the query is specified and executed, Bugzilla will return the list of issues relevant to the filters defined. While in some cases the displayed list of issues will answer the question reasonably (e.g., *What issues are assigned to me?*), in other cases, the results of the query will not provide desired information (e.g., *How have the issues changed?*) and additional mental effort is required to determine the context of the issues displayed. Developers need to go through the list manually and examine each issue in turn to figure out the answers to the questions describes in the example scenarios (Section 2). For example, if a developer wants to identify recent changes to the issues assigned to her, she first needs to receive results of a query, click on each issue in the list which will open a history page for each issue, and then scroll down to the bottom of the page to learn about recent changes.

Another challenge associated with the direct use of Bugzilla is that developers need to create a new query for each question they have, which means that they need to keep each query open in a separate browser tab. Moreover, developers need to spend time with each tab to process the results of the query every time they re-run their queries in order for them to get updates. These views and the time required to stay informed on the changes can interrupt developers from completing daily tasks.

3.2 DASH approach

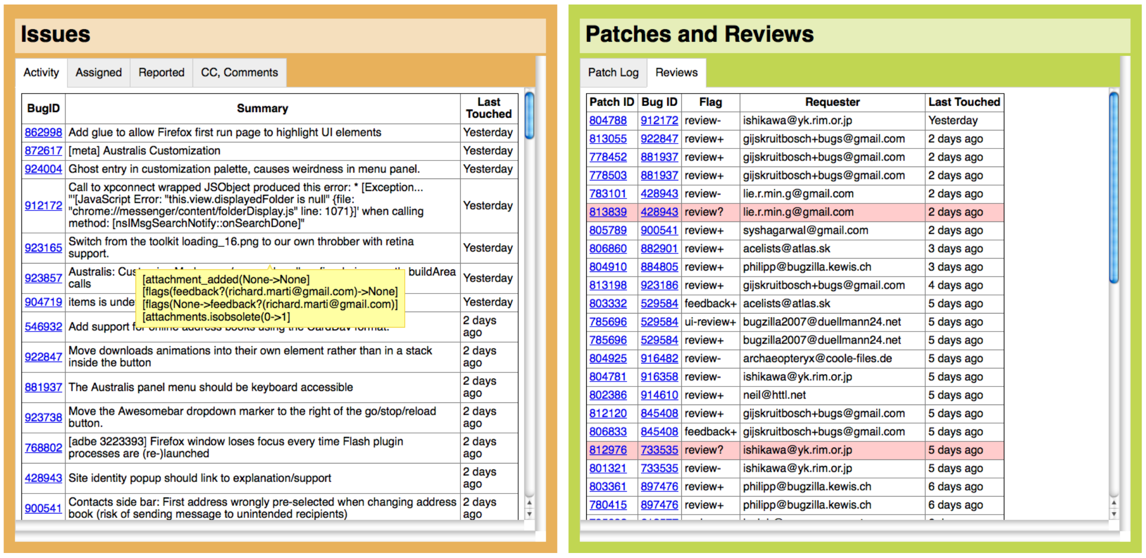
We chose to focus the UI of DASH[1] on usability, simplicity, and speed. We built our tool as a website, which can generate individualized views for each developer. The front end of the tool consists of a "login page" and a dynamically generated page that contains several views, with each view presented on its own in-page tab. We wanted to limit the number of steps a developer has to go through before she sees the views, so the only information we ask to type in is the email address of the developer and the time range for queries. Queries for all views (except otherwise stated) load only those items where the corresponding issue was modified within the specified time range.

The example of a dashboard that a developer is presented with is shown in the Figure 1. The dashboard displayed is generated for `mconley@mozilla.com` on October 8, 2013 for the last week period. DASH provides two views: on the left we placed items related to bug fixing and tracking, and on the right we display items related to patch making and code review tasks. Next we briefly describe each view and the information it contains.

3.2.1 Issues

The "Issues" view assist developers with keeping track of "important" issues. Each tab here has three columns: issue ID, its summary, and the time the issue was last modified. Issues are ordered by the last touched values with the most recent ones on top of the list. A tooltip appears when developers hover over an issue providing the information on the context of last change (e.g., a new comment or an updated review flag). This context of the recent action allows developers better understand how the issue was changed to make a decision on the next step. For example, developers can prioritize issues depending on the recent action.

[1]`http://claw.cs.uwaterloo.ca/~okononen/`



BugID	Summary	Last Touched
862998	Add glue to allow Firefox first run page to highlight UI elements	Yesterday
872617	[meta] Australis Customization	Yesterday
924004	Ghost entry in customization palette, causes weirdness in menu panel.	Yesterday
912172	Call to xpconnect wrapped JSObject produced this error: * [Exception... "'[JavaScript Error: "this.view.displayedFolder is null" {file: "chrome://messenger/content/folderDisplay.js" line: 1071}]' when calling method: [nsIMsgSearchNotify::onSearchDone]"	Yesterday
923165	Switch from the toolkit loading_16.png to our own throbber with retina support.	Yesterday
923857	Australis: Cus buildArea calls	Yesterday
904719	items is unde	Yesterday
546932	Add support f	2 days ago
922847	Move downloads animations into their own element rather than in a stack inside the button	2 days ago
881937	The Australis panel menu should be keyboard accessible	2 days ago
923738	Move the Awesomebar dropdown marker to the right of the go/stop/reload button.	2 days ago
768802	[adbe 3223393] Firefox window loses focus every time Flash plugin processes are (re-)launched	2 days ago
428943	Site identity popup should link to explanation/support	2 days ago
900541	Contacts side bar: First address wrongly pre-selected when changing address book (risk of sending message to unintended recipients)	2 days ago

```
[attachment_added(None->None]
[flags(feedback?(richard.marti@gmail.com)->None]
[flags(None->feedback?(richard.marti@gmail.com)]
[attachments.isobsolete(0->1]
```

Patch ID	Bug ID	Flag	Requester	Last Touched
804788	912172	review-	ishikawa@yk.rim.or.jp	Yesterday
813055	922847	review+	gijskruitbosch+bugs@gmail.com	2 days ago
778452	881937	review+	gijskruitbosch+bugs@gmail.com	2 days ago
778503	881937	review+	gijskruitbosch+bugs@gmail.com	2 days ago
783101	428943	review-	lie.r.min.g@gmail.com	2 days ago
813839	428943	review?	lie.r.min.g@gmail.com	2 days ago
805789	900541	review+	syshagarwal@gmail.com	2 days ago
806860	882901	review+	acelists@atlas.sk	3 days ago
804910	884805	review+	philipp@bugzilla.kewis.ch	3 days ago
813198	923186	review+	gijskruitbosch+bugs@gmail.com	4 days ago
803332	529584	feedback+	acelists@atlas.sk	5 days ago
785696	529584	ui-review+	bugzilla2007@duellmann24.net	5 days ago
785696	529584	review+	bugzilla2007@duellmann24.net	5 days ago
804925	916482	review-	archaeopteryx@coole-files.de	5 days ago
804781	916358	review-	ishikawa@yk.rim.or.jp	5 days ago
802386	914610	review+	neil@httl.net	5 days ago
812120	845408	review+	gijskruitbosch+bugs@gmail.com	5 days ago
806833	845408	feedback+	gijskruitbosch+bugs@gmail.com	5 days ago
812976	733535	review?	ishikawa@yk.rim.or.jp	5 days ago
801321	733535	review-	ishikawa@yk.rim.or.jp	5 days ago
803361	897476	review+	philipp@bugzilla.kewis.ch	6 days ago
780415	897476	review+	philipp@bugzilla.kewis.ch	6 days ago

Figure 1: The example of the DASHboard generated for mconley@mozilla.com.

The "Issues" view has four tabs: Activity, Assigned, Reported, and CC, Comments. "Activity" tab displays all the items that a developer is involved with. The second and third tabs represent issues that are assigned to a developer and issues that are reported by the developer respectively. And finally, "CC, Comments" tab lists issues that a developer is following up or was involved in a discussion by leaving comments.

3.2.2 *Patches and Reviews*

The main goal of the "Patches and Reviews" view is to provide developers with the information about their patches and review queues. The first tab "Patch Log" displays the list of patches that are written by the developer and submitted for reviews. Tracking review flags and their updates developers can determine whether her patch was reviewed and, if so, what the outcome of the review is. Developer is able to quickly learn who is blocking her patch and contact that person if the patch was pending a review for a long time.

The second tab "Reviews" displays patches that were reviewed or await review decisions from a developer. Patches that are pending reviews are highlighted by changing the background colour to prevent them from being forgotten. Items that were already reviewed include the outcome of the decision (`review+` or `review-`).

While this view is helpful to the developers in staying informed of the review assignments, it also lets developers determine the best person to request a review from. For example, if Alice created a patch and she knows that both Bob and Tom are appropriate people to ask for a review, she might want to choose the one who currently has lower review loads, since she is likely to receive a faster response. Thus, instead of requesting a review from both of them, she can use DASH to get insights into Bob and Tom's current review queues by looking at the items in their "Review" tabs.

Similar to the "Issues" view, all items in "Patches and Reviews" are ordered by the time of last touch and have tooltips with the information about the context of the last change.

4. IMPLEMENTATION DETAILS

We now provide technical details about the implementation of the DASH prototype and its current version.

4.1 Initial prototype

Our initial prototype was developed using Bugzilla's REST API ("BzAPI")[2]. While the prototype looked similar to the one shown in Figure 1, heavy loads of the requests on the API server affected its performance. From the discussions with the Mozilla developers we identified three main problems with the prototype:

1. the performance of our implementation was slow;
2. a user can run the tool only for one component at a time; and
3. there were filtering issues based on "created on" field instead of "modified on" (which caused some issues of not being displayed).

4.2 Elasticsearch

The Bugzilla team acknowledged performance issues, and they chose to adopt the Elasticsearch server[3] for internal use. The team also created software[4] for exporting data from Bugzilla into the Elasticsearch server. Developers from the Mozilla office in Toronto told us about Elasticsearch, and we decided to try it too.

Elasticsearch is a distributed open source server that allows near real-time search [11]. Similar to the notion of Database in a RBMS, Elasticsearch has a notion of Index. The server is document oriented, meaning that each Index on the server consists of the list of documents. In our case, each document represents an issue with all information about it (i.e., status, product, priority, attachments, etc.). Every time a change is made to a document (e.g., a new patch added), Elasticsearch automatically creates a new version of the document. Although this increases the number

[2]`https://wiki.mozilla.org/Bugzilla:REST_API`
[3]`http://www.elasticsearch.org/`
[4]`https://github.com/klahnakoski/Bugzilla-ETL`

of documents to be indexed, it also allows users to effortlessly see the history of all changes. By default, all fields in the documents are indexed, which allows for a faster search. Once we switched our backend to the Elasticsearch server, we saw an enormous improvement in the execution time.

We use the software from Mozilla to populate the server. This process is independent from our tool; we expect that this software will be constantly running to keep the server in sync with Bugzilla when the DASH is deployed for the real-life use.

DASH itself is written in Python. The main module receives two arguments via an AJAX request: the email of a developer and the time range. After that, DASH performs several queries using the Elasticsearch server to get the data required for each view. It processes the data from these queries and creates a block of HTML code that will be returned to the developer. Finally, on the developer's side a simple JavaScript code embeds the returned HTML into a page.

4.2.1 Deployment

Because the Elasticsearch server plays a role of an intermediary between the tool and Bugzilla, it is necessary that it be kept up-to-date. During the development we loaded quarterly updated public dumps of Bugzilla into our own cluster as we did not have real-time data on the server. The source code of DASH is made publicly available[5]. Mozilla developers in Toronto are currently working on setting up DASH along with the Elasticsearch server with real-time data on their side.

[5]`https://github.com/okononen/dash`

5. RELATED WORK

Many researchers have addressed the problem of awareness in software development [6–8, 10, 12, 13]. A number of tools have been developed to support awareness in a collaborative work environment [3–5, 9, 14] with a majority of tools focusing on source code information. FASTDash [3] offers an interactive visualization of a shared source code to provide insight into activities aggregated at the level of files and methods. Hipikat [5] provides assistance to new developers on the project by recommending relevant artifacts (source code, bug reports, emails) for a given task. The Bridge [14] tool enables full-text search across multiple data sources including source code, SCM repositories, bug report, feature request, etc. Yoohoo [9] 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 [4] increases developer awareness of version control conflicts during collaborative project development.

DASH was developed based on input from the industrial developers to overcome current limitations of the Bugzilla issue tracking system. The tool was qualitatively evaluated in industry setting and is actively being maintained by adding new features.

6. CONCLUSION

We have described DASH, a tool that provides developer-specific custom views of issue tracking repository. DASH is implemented as dynamic dashboards that filter and display work items relevant to the working context of individual developers. Developers quickly learn if there are status changes on the issues they are working with and all the data to support their key daily activities on software development and maintenance. As a result, they can better manage the excess of information and stay more aware on the issues as the project evolves.

7. REFERENCES

[1] O. Baysal, R. Holmes, and M. W. Godfrey. Situational Awareness: Personalizing Issue Tracking Systems. In *Proc. of the New Ideas and Emerging Results (NIER) Track, at ICSE*, 2013.

[2] M. Best. The Bugzilla Anthropology. `https://wiki.mozilla.org/Bugzilla_Anthropology`.

[3] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: A visual dashboard for fostering awareness in software teams. In *Proc. of CHI*, pages 1313–1322, 2007.

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

[5] D. Cubranić and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proc. of ICSE*, pages 408–418, 2003.

[6] D. Damian, L. Izquierdo, J. Singer, and I. Kwan. Awareness in the wild: Why communication breakdowns occur. In *Global Software Engineering, 2007. ICGSE 2007. Second IEEE International Conference on*, pages 81–90, 2007.

[7] R. DeLine, M. Czerwinski, B. Meyers, G. Venolia, S. Drucker, and G. Robertson. Code thumbnails: Using spatial memory to navigate source code. In *Proc. of the Visual Languages and Human-Centric Computing*, pages 11–18, 2006.

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

[9] R. Holmes and R. J. Walker. Customized awareness: Recommending relevant external change events. In *Proc. of ICSE*, pages 465–474, 2010.

[10] C.-Y. Jang, C. Steinfield, and B. Pfaff. Virtual team awareness and groupware support: an evaluation of the teamscope system. *Int. J. Hum.-Comput. Stud.*, 56(1):109–126, Jan. 2002.

[11] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey. Mining modern repositories with elasticsearch. In *Proc. of the 11th IEEE Working Conference on Mining Software Repositories*, Hyderabad, India, May–June 2014.

[12] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantir: Raising awareness among configuration management workspaces. In *Proc. of the ACM/IEEE Intl. Conf. on Software Engineering*, pages 444–454, 2003.

[13] I. Steinmacher, A. P. Chaves, and M. A. Gerosa. Awareness support in global software development: a systematic review based on the 3c collaboration model. In *Proceedings of the 16th international conference on Collaboration and technology*, pages 185–201, Berlin, Heidelberg, 2010.

[14] G. Venolia. Textual allusions to artifacts in software-related repositories. In *Proc. of MSR*, pages 151–154, 2006.