No Issue Left Behind: Reducing Information Overload in Issue Tracking

Olga Baysal
DIRO
Université de Montréal
Montréal, QC, Canada
obaysal@iro.umontreal.ca

Reid Holmes
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
rtholmes@uwaterloo.ca

Michael W. Godfrey
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
migod@uwaterloo.ca

ABSTRACT

Modern software development tools such as issue trackers are often complex and multi-purpose tools that provide access to an immense amount of raw information. Unfortunately, developers sometimes feel frustrated when they cannot easily obtain the particular information they need for a given task; furthermore, the constant influx of new data — the vast majority of which is irrelevant to their task at hand — may result in issues being "dropped on the floor".

In this paper, we present a developer-centric approach to issue tracking that aims to reduce information overload and improve developers' situational awareness. Our approach is motivated by a grounded theory study of developer comments, which suggests that customized views of a project's repositories that are tailored to developer-specific tasks can help developers better track their progress and understand the surrounding technical context. From the qualitative study, we uncovered a model of the kinds of information elements that are essential for developers in completing their daily tasks, and from this model we built a tool organized around customized issue-tracking dashboards. Further quantitative and qualitative evaluation demonstrated that this dashboard-like approach to issue tracking can reduce the volume of irrelevant emails by over 99% and also improve support for specific issue-tracking tasks.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques; D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement

General Terms

Design, Experimentation, Human Factors

Keywords

Developer dashboards, situational awareness, issue tracking, personalization, information needs

1. INTRODUCTION

Software developers work in complex, heterogeneous technical environments that involve a wide variety of tools and artifact repositories [23, 33]. These tools frequently provide generalized interfaces that can be used by many kinds of stakeholders including not only developers but also managers, QA, technical support, and marketing staff. Since these tools aim to provide a unified experience for every user, they may present information in a generic way, perhaps augmented by complex querying mechanisms to aid in narrowing the focus of the user's queries. At the same time, the complexity of these systems continues to increase [45], as does the amount of information any single user[1] needs to consider. For example, at the start of 2002 the Mozilla project issue-tracking repository contained around 117,500 issues, but by January 2013 it had reached 825,734, an average of over 175 new issues per day over 11 years (Figure 2). The flood of information that developers need to keep track of is always increasing because any update on an issue triggers an automatic email being sent to the developers involved with this issue (either by reporting it, leaving a comment, being assigned to fix it, or being on the CC list). Since much of this data is related only tangentially to the task at hand, there is an increased risk that a developer may miss something truly important amid the deluge: *"Bugzilla doesn't let you control the flow enough, 5000 email in a month and most of it doesn't relate to my work."*[2]

One way to help developers manage the increasing flow of information is to provide them with personalizable development tools that work to highlight details that are most important to their specific needs, while eliding the rest [41]. In this paper, we describe a qualitative study that identified an industrial desire for this kind of customization for issue-tracking systems. From this study we derived a model that captures the notion of the data and tasks developers want to have personalization support for. We then created and validated a high-fidelity prototype organized around personalized dashboards that provides a custom view of the issue-tracking repository; we do this by filtering irrelevant details and metadata from the issue-tracking system, and equipping developers with information relevant to their current tasks. Through qualitative and quantitative measures we evaluated our approach for its ability to reduce information

[1]For the remainder of the paper we will restrict our discussion to developers.
[2]All italicized quotes are verbatim comments made by Mozilla developers as part of the study described in Section 2.

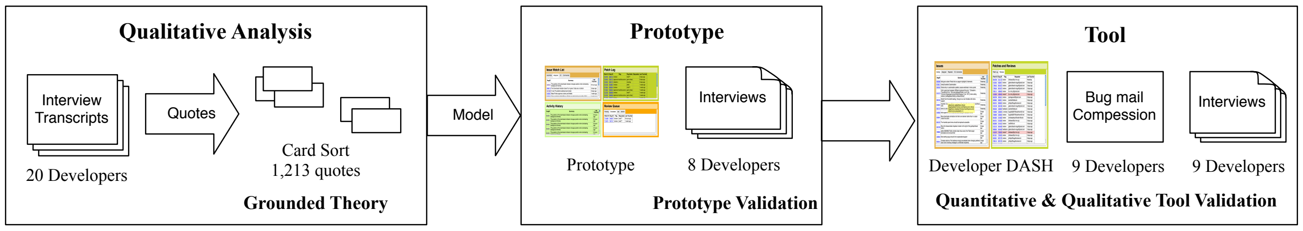


Figure 1: Our research method. In a qualitative analysis of the data from the interviews with industrial developers, we uncovered a model of the information needs for issue tracking based on the concepts of the key limitations of issue tracking that emerged during the open coding analysis (grounded theory). From this model we constructed a prototype, validated it, refined in into a tool that supports developers' situational awareness and validated this tool with the industrial developers.

overload while being able to improve developers' awareness of the issues they are working on and support their daily tasks.

Issue trackers have long been used by software teams to report, discuss, and track both defects and new features. While the collaborative demands of this problem space are well met by modern issue trackers [1, 21, 39], some routine kinds of tasks are poorly supported; a particular sore point is the weak support provided to developers in building and maintaining a detailed understanding of the current state of their system: What is the current status of my issues? Who is blocking my progress? How am I blocking others?.

These limitations can be overcome by adding detailed information that is specific to developers and their tasks, and by providing the technical means for developers to create these personalized views themselves: *"Querying in Bugzilla is hard; have to spend a few minutes to figure out how to do the query."* That is, offering developers customizable means of filtering information — such as via dashboards — can help them better maintain situational awareness about both the issues they are working on and how they are blocking other developers' progress on their own issues [3]. This finding is similar to the one described by Treude and Storey [42] who demonstrated that dashboards are used mainly for providing developers with high-level awareness (e.g., tracking the overall project status). In this work, we propose a model of the information needs for issue tracking based on the concepts and themes that emerged from a qualitative analysis of interviews with industrial developers. This model highlights the information needed to support daily development activities and tasks. From this model we constructed a prototype, refined this prototype as a tool implemented as developer dashboards, and validated the tool with the Mozilla developers. We show that these dashboards can help developers maintain their awareness on the project, as well as overcome the burden of managing a high volume of mostly irrelevant changes.

Some issue trackers already offer awareness features to help users to track recent activity on the projects. For example, in GitHub this feature is expressed by allowing users to manually 'watch' and 'follow' issues of interest. Bitbucket has recently (added April 2014) offered the ability to organize issues by a developer's involvement with them. While issue trackers provide filters to organize issues and tasks, this is a manual process that requires explicit user interaction and effort. Thus, this paper aims to 1) identify developer needs to support their issue reporting and fixing activities (expressed in a new model of issue tracking) and 2) offer a working solution that overcomes current limitations of many modern issue trackers by automatically providing customized views that do not require user interaction to configure that enable them to remain aware of the issues relevant to them in their repositories.

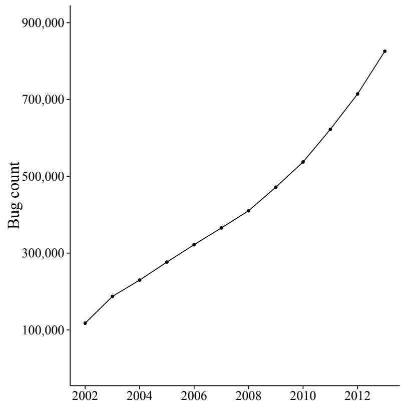


Figure 2: Mozilla's bug growth.

This paper makes the following contributions:

- A qualitative analysis of industrial developers' perceptions of the shortcomings of their widely-deployed issue tracker (Section 2).
- A model of information needs that describes the data and tasks developers want to have personalized within their issue-tracking system (Section 3).
- A tool that reifies the model for the Bugzilla issue-tracking system and an industrial validation of this tool (Section 4).

Based on the industrial feedback for our developer dashboard we are currently in the process of deploying the tool to industrial teams working with the Bugzilla issue-tracking systems.

The rest of the paper is organized as follows. Section 2 describes our methodology and the study setting. In Section 3, we discuss the results of the qualitative study on the set of interviews with developers and introduce our model of issue tracking. Section 4 provides an illustration of our high-fidelity prototype and describes our final tool together with its quantitative validation (bug mail compression) and qualitative evaluation (via interviews with developers). Section 5 summarizes prior related work. Section 6 offers a high-level discussion on improving issue-tracking systems that is motivated by our experiences, discusses future research directions and also addresses possible threats to validity of our work. Finally, Section 7 summarizes the contributions of this work.

2. QUALITATIVE STUDY

We have used a mixed method approach [14] consisting of three main steps (Figure 1): 1) data collection and analysis (open coding of industrial interviews); 2) a model of issue tracking that emerged during the qualitative analysis; this model supports customizable means of filtering software development tool information and is instantiated in a prototype which has been validated via interviews; and 3) tool development and its industrial validation (bug mail compression and interviews).

2.1 Data Collection

The initial data collection was performed as a part of the Mozilla Anthropology project [5]; this project was started in late 2011 to examine how various Mozilla community stakeholders 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 solicited feedback on various aspects of how developers interact with issues throughout their life cycle. 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. The full text of these transcripts is available online [5]; as far as we know ours is the first analysis of this data set.

2.2 Data Analysis

As the data collection was driven by an industrial initiative and conducted by a Mozilla employee, our analysis of the interview transcripts was exploratory: we had no predefined goals or research questions. Our interest was purely to understand how developers use their issue tracking systems.

We applied a grounded theory methodology to analyze the interview transcripts; as we had no predefined groups or categories, we performed an open coding approach to analyze the data. As we analyzed the quotes, sub-themes, themes and concept categories emerged and evolved during the open coding process [36].

The first author (Baysal) 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 (Baysal and Holmes) acted as coders to group cards into sub-themes, merging sub-themes into themes and developing concept categories. We proceeded with this analysis in three steps:

1. The two coders independently performed card sorts on the cards extracted from the interview transcripts of three participants — P1, P2, and P3 — to identify initial card groups (we refer to these groups as sub-themes in Section 2). The coders then met to compare and discuss their identified groups.

2. The two coders performed another independent round, sorting the comments of participants P4, P5, and P6 into the groups that were agreed-upon in the previous step. We then calculated and report the intercoder reliability to ensure the integrity of the card sort. We selected two of the most popular reliability coefficients for nominal data: percent agreement and Krippendorff's Alpha (see Table 1). Intercoder reliability is a measure of agreement among multiple coders for how they apply codes to text data. To calculate agreement, we counted the number of cards for each emerged group for both coders and used ReCal2 [24] for calculations. The coders achieved a high degree of agreement; on average two coders agreed on the coding of the content 98.5% of the time.

3. The rest of the card sort — the comment of participants P7–P20 — was performed by both coders together.

To make sense of the groups identified in the card sort, the 91 groups were clustered into 15 related themes, which were in turn organized into 4 concept categories; this was done using an affinity diagram, which helps to sort a large number of ideas into related clusters for analysis and review [10]. We generated the affinity diagram in three steps: 1) the names of the groups were printed on cards and attached to a whiteboard; 2) the groups were clustered on the whiteboard into related themes, and these themes were further clustered into high-level concept categories; and 3) each of the themes and concept categories was given a representative name. The final organization of concept categories, themes and sub-themes, as well as the results of the open coding and affinity diagram methods are available [2].

Table 1: Average Scores of Intercoder Reliability.

	Percent Agreement	Krippendorff's Alpha
Median	98.9%	0.871
Average	98.5%	0.865

3. A NEW MODEL OF ISSUE TRACKING

As a result of our exploratory study, we devised a developer-centric model of issue-tracking systems that addresses many of the key challenges that the study uncovered. The improvements are organized around the concepts that emerged during the card sort process. The model was then instantiated as a high-fidelity prototype (described in Section 4.1.1) that addresses the perceived limitations of the issue tracking platform and its incumbent processes; this was done by providing developer-specific enhancements to aid in task-specific decision making.

3.1 Model Categories

During our qualitative study of the twenty Mozilla interviews, four high-level concept categories emerged from the data, along with 15 themes and 91 sub-themes. Each concept category consisted of between two and six themes, and related to a different aspect of how Mozilla developers interact with Bugzilla while performing their daily tasks. Table 2

Table 2: Overview of Concept Categories.

Category	Participants	Quotes	Sub-themes
Situational awareness	19	208	14
Task Support	20	700	53
Expressiveness	20	188	12
Everything else	20	166	12

presents an overview of the concept categories, as well as the count of the participants, quotes, and sub-themes.

The four concept categories that emerged during the card sort do not correspond directly to the tasks developers perform daily; rather, they are a combination of actions developers perform when considering and executing these tasks. In this section, we provide an overview of the first three concept categories; the last one, which we named "Everything else", includes Mozilla internal topics that are not relevant to this discussion. We now highlight some of the key observations raised by a plurality of the interviewed developers, particularly those relevant to the notion of reducing information overload. In our report [2][3], for each theme and sub-theme we list the number of individual participants commenting on a certain issue and the total number of quotes given. For each sub-theme we suggest a synthetic quote that summarizes the common motif.

3.1.1 Situational Awareness

Email is a primary way of receiving bugs and communication updates. Developers have to manage the flood of emails they receive daily. As a result, developers find it hard to determine the current status of the bug they are working on where it is bug is in the bug-fixing process.

One of the most surprising concept categories we found clustered a series of themes and sub-themes that relate the idea of *situational awareness*. Situational awareness is a term from cognitive psychology; it refers to a state of mind where a person is highly attuned to changes in their environment [20]. The term is an apt description of how software developers must maintain awareness of what is happening on their project, to be able to manage a constant flow of information as issues evolve, and be able to plan appropriate actions. Developers often find themselves trying to identify the status of a bug — What is the current status of the issue? What is blocking this issue? What issues am I blocking? Who is the best person to review the patch? — as well as trying to track their own tasks — How many bugs do I need to triage, fix, review, or follow up on?

15 of the 20 participants expressed a desire in having private dashboards that would allow them to track their own activity and quickly determine the what changes had occurred since the last time they had examined the issue: *"[A] gigantic spreadsheet of bugs he is looking at. It would be useful to know how the bugs have changed since he last looked"* (P11). Since developers can track only a limited number of issues in their heads, they desired *"a personal list of bugs without claiming them"* (P8) and ability to *"get info based on what has changed since the last time I looked at it"* (P6).

[3]For a full description of the categories, themes, and sub-themes, please refer to [2].

Since tasks such as code review require collaboration or otherwise depend on others, developers expressed a desire: *"to better understand what people are working on and how it fits with [their tasks]"* (P11). Knowing what others are working on can enable assigning more relevant tasks to people: *"[I] frequently uses the review queue length to see who might be the quickest"* (P17). In addition, workload transparency can enable better load balancing: *"to spread the load on key reviewers"* (P3).

Situational awareness also crosscuts many of the other sub-themes from the other categories such as *Supporting Tasks::Code Review::Recommending Reviewers* where developers wanted the ability to determine the work load of a reviewer before requesting a review; this could be captured in a public dashboard that showed the reviewer's current queue.

3.1.2 Task Support

Bugzilla is the main venue for developers to collaborate on bug reporting and triage, code review activities, communication, release management, etc. However, the lack of good sorting and filtering mechanisms increases the effort required to perform these tasks.

Developers also had a number of comments that related directly to their daily tasks, including code review, triage, reporting, testing, and release management.

All interviewed developers expressed concerns with current support for tasks related to code review. Code review is an essential part of the development process at Mozilla; every patch is formally reviewed before it can be committed. *"The review system doesn't seem to be tightly integrated into Bugzilla"* (P10).

When a submitting patch for review, a developer must specify the name of a person they are requesting a review from. Deciding who is the "best" reviewer to send a patch to can be challenging since this requires estimating the workloads of others in addition to having an understanding of their expertise. Being able to get a list of reviewers along with their workloads would reduce the amount of time required to get a patch approved: *"He will go to the review requests page and look up people's review queue lengths to see who might be quickest"* (P17). Developers also want to be sure that they are not blocking others; they want to be able to easily assess their own review queues. One developer explained that he sorts his bug mail: *"so that it shows only the reviews that are asked of him"* (P20).

While developers face challenges managing a large flow of issues they are working with, Bugzilla also lacks good sorting and filtering mechanisms needed for tasks such as bug triage and sorting: *"The lack of ability to filter mixed with the volume makes it an overwhelming task. A lot of rework in sorting bugs rather than actually triaging it and moving it along"* (P9). One of the limitation of Bugzilla is that its interface is *"cluttered with rarely-used fields"* (P5). The amount of the metadata displayed to its users makes it difficult to search for the information that the developers need. Developers wanted to be able to sort information *"based on tag values"* (P5).

3.1.3 Expressiveness

Bugzilla stores a wide variety of metadata that can be overwhelming and intimidating to the developers reporting and fixing issues. A good tagging system could eliminate many fields such as OS, severity, priority, platform, etc.

Developers did not want Bugzilla to prevent them from modifying issues in a way that worked from them; in particular, they wanted to be able to express themselves in a variety of ways to convey information to other stakeholders. Expressiveness in Bugzilla is achieved in part through the large number of labelled fields available for entry on issues including whiteboard terms, keywords, tracking flags, priorities, components, etc. These fields are used as *"sort of a tagging system"* (P4) during issue management including tracking status, seeking approval, highlighting important information on a bug, or making version names and numbers. While these fields are important for grouping and organizing issues in addition to communicating awareness and interest on issues, they are often used in a project- or team-specific manner. Therefore, developers desired a good tagging system that *"could get rid of many fields"* (P5) as *"a cross between the keyword field and whiteboard field"* (P5).

Prioritization is another concept that appears to be poorly supported in Bugzilla. Some fields such as `severity` and `priority` do not have a clear definition and thus are often used incorrectly. Developers explained that *"priority and severity are too vague to be useful"* (P5) and *"everyone doesn't use priority levels consistently"* (P11). Instead of *"prioritizing en masse"* (P5), developers seek a means to *"set our own priorities on bugs"* (P15) or team priorities on issues and tasks and sort them based on their importance: *"team decides priority as a group, P1: forget about everything else, P2: have to do, P3: don't look until later"* (P12).

3.2 Key Information Sets

We hypothesize that many of the developers' comments in the concept categories of situational awareness, supporting tasks, and expressiveness can be tackled through the creation of custom views of issue-tracking systems; these views can be tailored for individual developers by filtering relevant information from the issue-tracking system. In general, the information that developers are trying to keep up with is usually stored within the issue-tracking repository, it is just hard to access. The primary means that developers currently do this is through 'bug mail'; that is, the developers subscribe to a large number of bugs and are sent email notifications whenever something in the bug changes. Unfortunately, this results in hundreds of emails per day, more than even a complex array of filters can hope to keep up with without some important updates being lost.

Based on the qualitative analysis, we identified several ways in which we hypothesized that the needs of users interacting with issue tracking could be better met, largely by easing access to key information that already exists within the system but can be hard to obtain through the web interface or email filtering alone.

We identified two groups of work items that developers are engaged with: issues and patches. Developers work with both of these on a daily basis. Issues contain bug reports and new features that need to be implemented, while patches contain the reification of issues in source code that can then be reviewed by other developers. Both the management of issues and patches are of key concerns to developers.

In this section, we describe the key information elements that developers stated a desire to keep appraised of, and we explain how these elements can change over time. One of the reasons these requests arose is that people themselves are essentially metadata on issues: a person files an issue, an issue is assigned to a person, a person comments on an issue, requests a review, and files a patch. While the issue is the central artifact, all *actions* on issues are generated manually by people.

3.2.1 Issues

For developers who use Bugzilla, email is the key communication mechanism for discussing bugs and the bug fixing process. Any change on an issue results in an email being sent to the developers whose names are on the issue's CC list. For many developers, these emails are the primary means for maintaining awareness of issue evolution in the issue-tracking repository. Developers receive an email every time they submit or edit an issue, someone comments or votes on a bug, or any metadata field is altered. An individual developer can track only a limited number of bugs in their head; 10 of the developers who were interviewed wanted to be able to watch bugs and sort them by activity date. One said, *"[I] would like to have a personal list of bugs without claiming them"* (P8).

Many developers wanted "watch lists" for indicating their interest in an issue without taking ownership of it. Watch lists provide means to track bugs privately by adding them to their private watch list without developers having to CC themselves on the bugs. Bugs are ordered by "last touched" time as *"last touched time a key metric for tracking if work is being done on a bug"* (P1).

We found that developers face challenges in determining what has happened since the last time an issue was examined; this was noted by 12 participants, whose comments included: *"[I want] to get info based on what has changed since last time I looked at it"* (P6), and *"You look at the bug and you think, who has the ball? What do we do next?"* (P7).

Ultimately, developers wanted more flexibility in tracking, querying, and exploring the issues that are stored in the repository. While Bugzilla provides a web interface that developers must use to modify issues, developers also rely heavily on automated bug mail, as email clients support flexible sorting and filtering of messages. Unfortunately, in an active bug, many changes may occur simultaneously, resulting in a large number of emails, only some of which may be interesting to the developer; also an email message can be easily missed amid the deluge, causing an issue to "fall on the floor."

To track issues effectively, developers need access to the issue and its metadata presented to them in a meaningful way. From the expressiveness concept category, we know that different developers often desire access to different pieces of metadata. In reality, these requests arise because developers are thinking about how they will filter the issue's bug mail. What the developers seek is a customized list of issues — implemented through whatever technical means might work — that keeps track of a variety of issues and can specify how "interesting" they consider each one to be. For example, developers want a list of issues that are assigned to them, sorted by when they last changed. Additionally, they would like lists of issues they have commented on, are CCed on, and have voted on. They would also like to be able to have component-level lists that they can then select issues to move into private watch lists to keep track of. As these issues evolve, the lists should continually update "live" so that the most recently updated issues appear first.

3.2.2 Patches and Reviews

Bug fixing tasks involve making *patches.* While working on an issue, developers will often split a single conceptual fix into multiple patches: *"People are moving to having multiple patches rather than one large patch. This really helps with the review. Bugzilla isn't really setup for this model"* (P16). Ten developers expressed a desire to improve the way Bugzilla handles patches: *"It would be good if [Bugzilla] could tell you that your patch is stale"* (P13). Developers were primarily interested in tracking their own patch activity, as well as determining what patches are awaiting reviews, or who is blocking their reviews.

12 participants indicated that they felt Bugzilla is ill-suited for conducting code review: *"The review system doesn't seem to be tightly integrated into Bugzilla"* (P10). A common task is determining who the "right" reviewer would be to request a review from: this may may be the one having faster review turnaround or the one having a shorter review queue. In order to address this question, developers need to be informed about reviewers' work loads and average response time: *"I can be more helpful if I can better understand what people are working on and how it fits with [their tasks]"* (P11). While Bugzilla keeps track of all of the reviews outstanding on all issues, developers cannot query to find out what the review queue of an individual developer is. Supporting this task 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).

Developers also need some means to observe and track their own tasks, such as their review queues: *"He has a query that shows all his open review queue"* (P16), *"The review queues are very useful, he will check that every few days just to double check he didn't miss an email"* (P8).

Code review is a particularly important task, where developers do not want to miss key events related to it. While their patches are awaiting review they are effectively blocked for that issue. If their review request is missed or lost, a significant delay can result. The converse also happens, if a developer misses a review request they can block the progress of other developers.

Rather than having code review notifications flow through bug mail, the developers requested a dedicated review queue. As developers usually have a limited number of outstanding review requests, bringing these all together can ensure that none are missing; for example, in a list of five requests, the one that is a month older than the other four tends to be noticeable. Developers also wanted to be able to observe the review queues for other developers so they can estimate whether they would be able to turn around a review for them in a reasonable amount of time.

3.3 Model Summary

Our conceptual model of issue tracking is derived from the data of interviews with industrial developers, and reflects the concepts that emerged during the qualitative study described in Section 2:

Issues are work items that developers are involved with during the active development of a software system; issue tracking issue is one of the key tasks developers perform daily.

Patches are code modifications that developers "produce" to resolve issues or implement new features; a typical developer's daily activities may include writing and tracking their own patches, as well as conducting code reviews of the patches of others.

Both issues and patches relate to the themes of *task support* (Section 3.1.2).

Identifying relevance involves discerning work items, such as issues and patches, that are of concern to a developer; it relates to *situational awareness* (described in Section 3.1.1).

Information reduction concerns filtering out irrelevant metadata fields — such as priority, severity, product, etc. — so that the more important fields — such as bug ID and summary fields for issues, and patch ID, issue ID, flags, and requester fields for patches — are more prominent; it relates to *situational awareness.*

Temporality concerns displaying issues sorted by the "Last touched" field, adding visual clues to the items that require attention (e.g., patches awaiting reviews), adding context to work items (e.g., what has changed on the issue?); it relates to both *expressiveness* (Section 3.1.3) and *situational awareness.*

Roles concern organizing issues by developer role on the project; roles can include bug reporter, bug fixer, and reviewer.

Ownership relates to separating issues that developers are responsible for from those in which they are only observers.

Roles and ownership information needs relate to both *situational awareness* and *task support.*

This model highlights the key information needs much desired by the industrial developers for the dashboards that are designed to overcome current limitations of the issue-tracking systems and provide developers with better awareness of their working environment.

4. DASH: REDUCING INFORMATION OVERLOAD

In this section, we present our tool organized around customized issue-tracking dashboards and describe an industrial validation of the tool.

4.1 Background

To validate the concepts that emerged from the qualitative analysis of the interview data, as well as the prototype (see Section 4.1.2) we interviewed eight (D1-D8) Mozilla developers for 20-60 minutes. None of these developers were involved in the initial set of interviews.

The developers confirmed that they face challenges keeping track of tasks they are involved with, organizing issues they are working on, *"I have no way of parsing or prioritizing the component information so I can't watch that very well"* (D8). They observed that developers often created their own ad hoc solutions for organizing tasks and keeping track of issues, such as public work diaries, notes on paper, "homebrew" tools filtering bug mail, saved Bugzilla searches, mail clients with better filtering capabilities. For example, one developer explained *"I use etherpad to keep track of my list of bugs for each project listed here and I move them up the chain. I put little notes on something so I can keep track of what I'm waiting for; it's all very ad hoc"* (D1). Another developer tried several applications before he *"switched to paper now; [I use] paper for one-day tasks, Bugzilla for longer tasks"* (D7).

Developers were keen to ensure that patches did not *"fall off the radar"* (D1), and that important issues were not allowed to *"fall through the cracks"* (D5).

Ultimately, these interviews ended up echoing most of the major themes identified during the first round of interviews.

4.1.1 High-Fidelity Prototype

We have implemented a prototype of our model in the form of a personalized dashboard that provides developers with public watch lists, patch logs, and an activity history feature; the active history can help Bugzilla users maintain better awareness of the issues they are working on, as well as other issues that interest them, and common tasks they perform daily. Our solution is organized around custom views of the Bugzilla repository supporting ongoing situational awareness of what is happening on the project. Figure 3 illustrates developer dashboards; it shows a custom view of the Bugzilla repository generated for Mike Conley, a Mozilla developer.

The dashboard serves as a template for displaying information to assist in developers' common tasks. This template contains all the key elements that are important to the developers as they capture the concepts derived from the qualitative analysis (as described in Section 3.3). There are two panels: `Issues` and `Patches and Reviews`. The `Issues` panel includes four tabs: `Reported`, for issues that are reported by a developer; `Assigned`, for bugs that a developer needs to resolve; `CC, Comments`, for issues that developer expressed interest in following up on either by putting his name on its CC list or by voting on a bug; as well as issues that the developer participated in a discussion on. The `Activity` panel displays the developer's activity on the project including all issues from `Assigned`, `Reported` and `CC,Comments` tabs with duplicates removed. The `Patches and Reviews` panel consists of two tabs. The `Patch Log` tab displays the list of recently submitted patches, the outcome of a code review (positive or negative) with the name of the reviewer, or the current status of the patch (approval for committing the patch to the master tree) with the name of the person who set the flag. Finally, the `Reviews` tab shows patches that await the developer's review, together with the name of the person who made this request and it also lists recently reviewed patches together with the review decision. The layout of the prototype displays the same information in four panels: `Activity`, `Issues`, `Patch Log`, and `Reviews`.

By default, all of the tabs are sorted by `Last Touched`, the timestamp of the most recent change on an issue. If the developer prefers to see the date and time, they can hover over the last touched cell.

4.1.2 Prototype Evaluation

This step of our study aimed at validating our high-fidelity prototype. As mentioned earlier (in Sec. 4.1), we conducted eight interviews with Mozilla developers lasting 20–60 minutes each; none of these developers had participated in the original interviews. Our interviews included questions relating to how the developers managed their daily tasks, patches, and code reviews. During the interviews, we provided the developers with our prototype, personalized for their work from the previous day. We asked the developers to comment on the prototype and how it could be extended in the future.

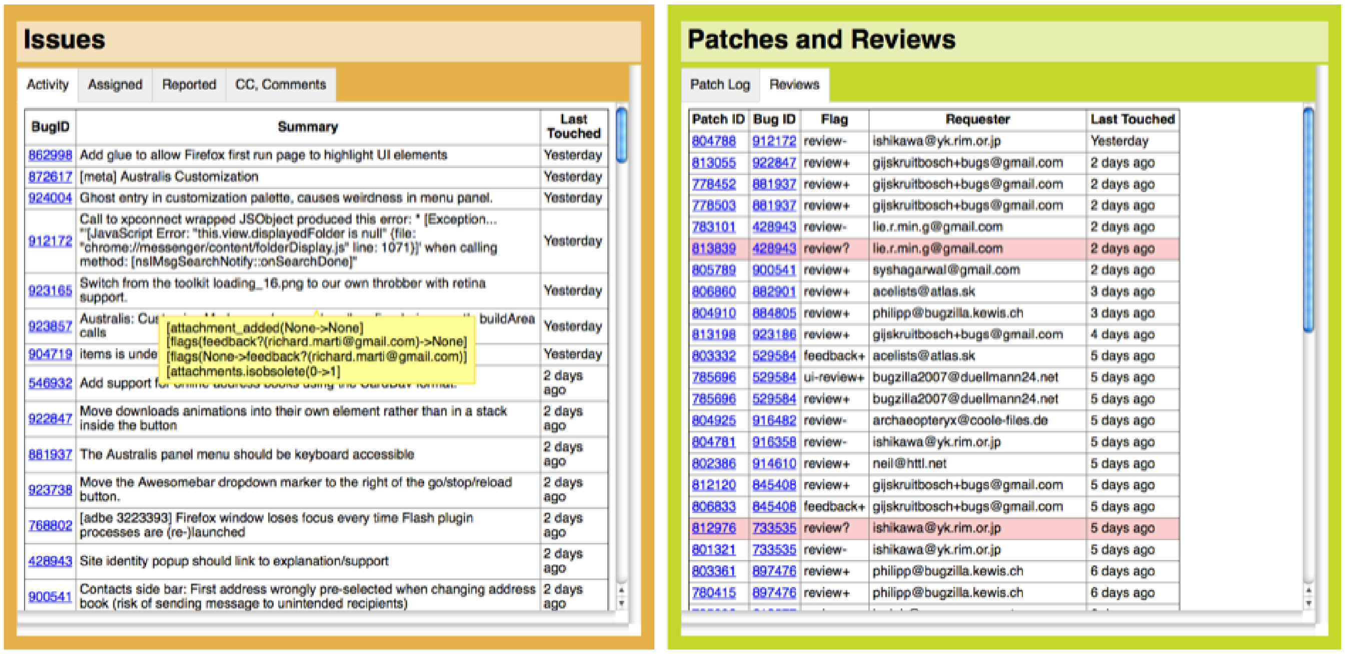
Our prototype focused on extracting metadata from Bugzilla and displaying it for users in a meaningful way. The intent was to provide a dashboard that developers could use throughout the day to keep track of their issues. The night before we met with the developers we generated two versions of the prototype (a four-panel layout (not shown) and a two-panel layout shown in Figure 3) that were specific to the issues they were working on at that time.

The comments made during the interviews suggest that the high-fidelity prototype was well received: *"this [prototype] is really cool! I think this is great, something like this would be fantastic for sorting through all the things I need to take care of. When can I start using it?"* (D6).

Developers were glad to see that most of the tasks they perform are explicitly supported: tracking issues, tracking assigned tasks, assigning reviewers, prioritizing tasks, etc. *"I really like the issue watch list. I have my own custom Bugzilla queries for the four columns [Submitted, Assigned, CC, and Commented]. It's a saved search I have but when I do use it it is hard to look through the list"* (D2). *"I really like the idea of having peers tab"* (D7). *"Oh, and activity history so I can see everything I've contributed to the project"* (D6). *"I like the patch log as well because I find that the bugs I care about often have active patches in them"* (D2). These were all ways in which the prototype helped developers access information that was crucial to their day-to-day development tasks.

Most developers said that being able to determine what issues are assigned to them is useful to have a quick start each day. While some developers would choose to open developer dashboards once or twice during the day: *"If I have small tasks more often [to visit], one big thing — once every morning and evening"* (D6) , others expressed interests in frequently checking dashboards for any updates: *"I'd like to keep this [dashboard] open all the time, all day long"* (D7), *"Honestly, if we had better dashboards I would keep them open and come back to it frequently, several times a day"* (D8).

During the interviews we demonstrated two versions of our prototype: a two-panel view (similar to the one shown in Figure 3) and four-panel view (not shown due to space constraints). Both versions contained the same information and tabs. The only difference was the grouping of the displayed tabs and the way in which the page was divided into the panels. Developer feedback on the page layout was unanimous that the two-panel layout of the landing page was better: *"The two views make more sense. Reducing the need to scroll is good."* (D1), *"I work on a mobile a lot and the*



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: Cu[illegible] buildArea calls	Yesterday
904719	items is unde[illegible]	Yesterday
546932	Add support [illegible]	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 3: Developer dashboard generated for the Mozilla developer Mike Conley.

two-column one is nice because I can see a lot more data" (D6).

The interviews also revealed that further refinement of the prototype's functionality is needed. While we expected requests for richer customization of the base template and the interface, we were surprised to hear that most improvements involved questions of *expressiveness* (discussed in Section 3.1.3), including:

- *A customizable template* — Developers expressed a variety of preferences for what tabs should be present by default; providing a template that developers can modify according to their individual needs is important.

- *Search functionality* — Developers stated a desire to be able to perform a quick search on the various bug fields, both the visible ones and the underlying fields that are not shown.

- *Time range options* — While custom pages were generated for the past three weeks, we received a variety of answers with respect to this setting: *"3 months is useful, for thinking about our quarterly goals"* (D7), *"for last month"* (D6), *"every year or every six months is useful [...] to recall things I have worked on"* (D7).

Apart from some common recommendations on further improvements, we also received requests to meet the specialized needs of some developers and teams. One developer wanted support for other tasks, such as triage of issues of the component they are responsible for: *"We have bi-monthly person for a week who is responsible for bug triage: review bugs, triage them — these bugs are important, these are not, check may be I have missed something"* (D7). Another developer wanted to be able to send email from the landing page, e.g., to reply to a recently added comment on an issue without having to enter Bugzilla.

All of the desired improvements that were discussed in the interviews we supported in the implementation of our tool.

4.2 DASH Tool

To motivate and define all DASH features, we used the model (described in Section 3) that emerged from the qualitative study.

Based on our prototype and the feedback we received while validating it, we implemented a number of changes to our final tool. These modifications include:

- a component-based query to allow developers to organize work items for a particular product they are involved in,

- a time-range option to generate custom views of Bugzilla for a specific time frame, and

- a two-list layout that separates issue-tracking activities from tracking updates for patches and reviews.

These improvements enhance our model of the information needs for issue tracking (summarized in Section 3.3) by including additional features related to the topics of situational awareness and expressiveness. Further details about the architecture and implementation of DASH[4] can be found elsewhere [35].

4.3 Tool Validation

To investigate the advantages of the tool for the developer over their day-to-day practices, we visited the Mozilla office again and talked to the most active users of our tool (developers A–I in Table 3).

During these interviews, we asked them to count the number of work-related emails they received the previous day from Bugzilla (carefully stating the assumption "if yesterday

[4]Tool Demo Video: `http://youtu.be/Jka_MsZet20`

was a typical day for you") to determine their average daily and weekly bug mail. We then generated developer-specific dashboards for each developer for the period of one week and counted the number of items displayed on the dashboard; these include issues, patches, and their reviews that the developer was involved or interested in. The results can be found in Table 3, and show that the compression of the received bug mail is over 99%.

Table 3: Scale of bug mail and number of items displayed on the dashboard for the week-long period.

Developer	Bug mail (one week)	Dashboard (one week)	Reduction Percent
A	435	16	99.96%
B	605	25	99.95%
C	2500	28	99.98%
D	1200	5	99.99%
E	525	9	99.98%
F	1500	26	99.98%
G	235	2	99.99%
H	1000	14	99.98%
I	250	13	99.94%
Average:	917	15	99.97%

We then asked developers for their feedback on the relevance of the filtered information, the correctness of our filtering approach, and the accuracy of the bug mail compression ratio. Interviews with the developers confirmed that vast majority of the received emails are not relevant to their work: *"Easily 200+ bug mails a day ... in fact most of them I do not need to read"* (H).

Seven of the nine developers we interviewed (78%) expressed a desire to use the developer dashboard in place of their bug mail, *"... with a performant/live updating dashboard it seems feasible to largely replace bug mail"* (B). Of the developers who wanted to continue receiving all their bug mail, one cited a need to track bug-related conversations: *"One of the advantages of email is that I have a copy of the conversation that's going on in a bug — so I don't actually have to enter Bugzilla to read the comments. With your dashboard, I can know that a bug had a new comment posted, but then I have to go into Bugzilla to see if it's important. So I think that's a slight deficiency"* (A). The other developer wanted to be able to monitor component bug mail to identify new bugs he might be interested in fixing: *"99% of this [bug mail] is not about bugs I'm involved in, but Bugzilla components I'm watching to see if any bugs I am interested in have come up"* (D).

5. RELATED WORK

Improving Issue-Tracking Systems — The research community has provided several investigations of how issue-tracking systems can be improved. Most of this work has focused on understanding how issue management systems are being used in practice. For example, Bettenburg et al. surveyed 175 developers and users from the Apache, Eclipse, and Mozilla projects to investigate what makes a good quality bug report [7]. They developed a tool that measures the quality of new bug reports and recommends ways improve their quality. Bertram et al. performed a qualitative study of the use of issue-tracking systems by small, collocated software development teams and identified a number of social dimensions to augment issue trackers with [4]. They found that an issue tracker serves not only as a database for tracking bugs, features, and requests but also as a communication and coordination hub for many stakeholders.

Several studies have suggested a number of design improvements for developing future issue-tracking tools. Just et al. [32] performed a quantitative study on the responses from the previous survey [7] to suggest improvements to bug tracking systems. Zimmermann et al. addressed the limitations of bug tracking systems by proposing four themes for future enhancements: tool-centric, information-centric, process-centric, and user-centric [44]. They proposed a design for a system that gathers information from bug reports to identify defect locations in the source code. Breu et al. quantitatively and qualitatively analyzed the questions asked in a sample of 600 bug reports from the Mozilla and Eclipse projects [9]. They categorized the questions and analyzed response rates and times by category and project, and provided recommendations on how bug tracking systems could be improved. Rastkar et al. [37] and Czarnecki et al. [16] both worked on the problem of summarizing bug reports in the Bugzilla issue-tracking systems.

We note that while existing research has provided recommendations on how such systems can be improved, there is a relative lack of efforts in providing solutions to the developers that can help them overcome shortcomings of the existing issue management systems and assist them with their daily development tasks.

Increasing Awareness in Software Development — Much of the work in this area has focused on how communication and coordination affects awareness in global software development [17, 27, 29, 31, 40]. Several tools have been developed to enhance developer awareness and understanding of large source code bases [12, 18, 19, 22, 26, 28, 38] but none of them specifically targets issue-tracking systems.

Treude and Storey [42] investigated the role of awareness tools — such as IBM's Jazz dashboards and feeds — in supporting development activities. Their findings suggest that dashboards are used mainly to keep track of the overall project status, to provide awareness of the work of other teams, and to stir competition between teams; they found that feeds are used to track work at a small scale and to plan short term development activities. Our qualitative study (described in Section 2) also motivates the need for customized dashboards to support developers' awareness of their immediate environment and the issues that they work with. Further, our work led to the creation of a comprehensive model of the information needs for these dashboards, instantiated it in a prototype tool, and validated this tool with industrial developers.

Cherubini et al. [13] looked at how and why developers use drawing during software development. Furthermore, Fritz and Murphy [25] 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.

Existing research also offers a number of tools [8, 11, 15, 30, 34, 43] to assist developers with daily tasks and development activities; these tools are more relevant to our research goals. FASTDash [8] offers an interactive visualization to enhance team awareness during collaborative programming tasks. Hipikat [15] provides assistance to new developers on the project by recommending relevant artifacts (source code, bug reports, emails) for a given task. The Bridge [43] tool enables full-text search across multiple data sources including source code, SCM repositories, bug report, feature request, etc. Mylyn [34] 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 [30] 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 [11] increases developer awareness of version control conflicts during collaborative project development.

While our tool enhances the developers' situational awareness of their working environment, it also provides developers with the customizable means of managing a high volume of information in the issuer tracking systems.

6. DISCUSSION

In this section, we describe some of the key concerns raised by developers that we have not yet investigated, along with threats to validity.

6.1 Shortcomings

Our dashboards are implemented using custom views of the issue-tracking system filtering the data to identify developer-specific items. Currently, they do not support developer needs for private watch lists. Public watch lists (aka being on the CC list) provide a mechanism to indicate interest in an issue without taking ownership. In contrast, private watch lists enable developers to privately mark bugs to watch. Unfortunately, implementing these lists is not possible within the current Bugzilla architecture; we are currently investigating ways around this.

Our tool does not currently support personal tagging. Some developers wanted the ability to group issues by whiteboard flags, *"may be you could say what whiteboard flags you're interested in. But we don't want to get it to be replacing Bugzilla at all."* (D2) Others wanted to be able to *"have them grouped by colour and sorted within the colour by date"* (D5). These grouping enhancements were by far the most prevalent: developers wanted to be able to organize their issues in ways that were most relevant to them. Most developers expressed a willingness to manually move issues to specific groups if this preference could be maintained.

Bettenburg and Begel [6] recently described an automatic approach for classifying work items to reveal what is actually happening within a task. Their approach could add contextual rationale to the work items displayed on the dashboards to help developers interpret and differentiate their work activities.

6.2 Deployment

We are currently deploying our developer dashboards into the development environment of the Mozilla project. The tool is currently offered as a web-based service, hosted by one of our research group's servers. While most of the data stored in Bugzilla is public, Mozilla is interested in moving the tool to their internal network to be able to populate confidential data, such as security bugs. We are actively working on adding new features — such as custom tagging — to the dashboard.

6.3 Threats and Limitations

The first limitation lies in the validity of our findings from the qualitative study. However, Bugzilla is widely-deployed and used by thousands of organizations, including open-source projects such as Mozilla, the Linux kernel, and Eclipse, as well as NASA, Facebook, and Yahoo!.[5] Our investigation has focused strongly on Mozilla developers using the Bugzilla issue-tracking system. While some issue trackers provide better developer-oriented support (e.g., the Bitbucket issue tracker allows users to tag people so they get notifications to issues they are not subscribed to), our work aims to provide guidelines that we hope can improve the information filtering ability of all issue trackers.

As with all exploratory studies, there is a chance we may have biased the categories that were identified. We tried to minimize this by coding the first three subjects independently, performing an inter-coder reliability measurement on the next three, by validating our findings with a separate group of developers along with our high-level prototype and finally by evaluating our final tool both quantitatively (usage data analysis and bug mail compression) and qualitatively (interviews with active users).

7. CONCLUSION

In this paper, we described a qualitative study of industrial developers that identified a need for improved approaches to issue tracking. The model of these information needs was derived from the interview data in which it was "grounded". Based on this model we designed our tool implemented in the form of developer dashboards. These developer dashboards can enable developers to better focus on the evolution of their issues in a high-traffic environment, and to more easily learn about new issues that may be relevant to them. We have proposed an approach that improves support for particular tasks individual developers need to perform by presenting them with the custom views of the information stored in the issue management system. By improving issue management systems, developers can stay informed about the changes on the project, track their daily tasks and activities. We validated our initial prototype and later our tool with Mozilla developers and received feedback on the desired information and features. We are actively improving our tool to enable it to be deployed in an industrial setting.

8. ACKNOWLEDGEMENTS

We thank Oleksii Kononenko for moving DASH to Elasticsearch; Martin Best, Mike Hoye, and Kyle Lahnakoski from the Mozilla Corporation for their help in recruiting developers and providing environment for conducting interviews. We also thank all Mozilla developers participated in our study for their time and feedback.

9. REFERENCES

[1] Atlassian. Jira. `http://www.atlassian.com/software/jira/overview`.

[2] O. Baysal and R. Holmes. A Qualitative Study of Mozilla's Process Management Practices. Technical Report CS-2012-10, David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, June 2012. Also available online

[5] `http://www.bugzilla.org/installation-list/`

http://www.cs.uwaterloo.ca/research/tr/2012/CS-2012-10.pdf.

[3] 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, the 35th Int. Conf. on Soft. Eng.*, 2013.

[4] D. Bertram, A. Voida, S. Greenberg, and R. Walker. Communication, collaboration, and bugs: The social nature of issue tracking in small, collocated teams. In *Proc. of the ACM Conf. on Computer Supported Cooperative Work*, pages 291–300, 2010.

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

[6] N. Bettenburg and A. Begel. Deciphering the story of software development through frequent pattern mining. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1197–1200, 2013.

[7] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proc. of the ACM-SIGSOFT Intl. Symposium on Foundations of Software Engineering*, pages 308–318, 2008.

[8] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: A visual dashboard for fostering awareness in software teams. In *Proc. of the ACM-SIGCHI Conf. on Human Factors in Computing Systems*, pages 1313–1322, 2007.

[9] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *Proc. of the ACM Conf. on Computer Supported Cooperative Work*, pages 301–310, 2010.

[10] G. Britz. *Improving Performance Through Statistical Thinking.* ASQ Quality Press, 2000.

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

[12] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse with collaborative tools. In *Proc. of the 2003 OOPSLA Workshop on Eclipse Technology eXchange*, pages 45–49, 2003.

[13] 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*, pages 557–566, 2007.

[14] J. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* SAGE Publications, 2003.

[15] D. Cubranić and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proc. of the Intl. Conf. on Software Engineering*, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

[16] K. Czarnecki, Z. Malik, and R. Lotufo. Modelling the "hurried" bug report reading process to summarize bug reports. In *Proceedings of the 2012 IEEE International Conference on Software Maintenance (ICSM)*, pages 430–439, Washington, DC, USA, 2012. IEEE Computer Society.

[17] 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.

[18] 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.

[19] R. DeLine, M. Czerwinski, and G. Robertson. Easing program comprehension by sharing navigation data. In *Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 241–248, 2005.

[20] M. R. Endsley. Toward a theory of situation awareness in dynamic systems. *Human factors*, 37(1):32–64, 1995.

[21] ENTP. Lighthouse. https://lighthouseapp.com/.

[22] J. Espinosa, S. Slaughter, R. Kraut, and J. Herbsleb. Team knowledge and coordination in geographically distributed software development. *J. Manage. Inf. Syst.*, 24(1):135–169, July 2007.

[23] C. Fernstrom, K.-H. Narfelt, and L. Ohlsson. Software factory principles, architecture, and experiments. *Software, IEEE*, 9(2):36–44, march 1992.

[24] D. Freelon. ReCal2: Reliability for 2 coders. http://dfreelon.org/utils/recalfront/recal2/.

[25] 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*, pages 1827–1830, 2011.

[26] J. Froehlich and P. Dourish. Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proc. of the ACM/IEEE Intl. Conf. on Software Engineering*, pages 387–396, 2004.

[27] C. Godart, P. Molli, G. Oster, O. Perrin, H. Skaf-Molli, P. Ray, and F. Rabhi. The toxicfarm integrated cooperation framework for virtual teams. In *Distributed and parallel databases: special issue on teamware technologies*, pages 67–88, 2004.

[28] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, CSCW '04, pages 72–81, New York, NY, USA, 2004. ACM.

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

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

[31] 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.

[32] S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In *IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 82 –85, Sept. 2008.

[33] R. Kadia. Issues encountered in building a flexible

software development environment: lessons from the arcadia project. In *Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*, SDE 5, pages 169–180, 1992.

[34] 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*, pages 1–11, 2006.

[35] O. Kononenko, O. Baysal, R. Holmes, and M. W. Godfrey. DASHboards: Enhancing developer situational awareness. In *Proceedings of the Formal Demonstration Track, at the 36th International Conference on Software Engineering*, ICSE'14, 2013.

[36] M. Miles and A. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE Publications, 1994.

[37] S. Rastkar, G. C. Murphy, and G. Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 505–514, New York, NY, USA, 2010. ACM.

[38] 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.

[39] E. Software. Trac. `http://trac.edgewall.org/`.

[40] 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.

[41] M.-A. Storey, L.-T. Cheng, I. Bull, and P. Rigby. Shared waypoints and social tagging to support collaboration in software development. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW '06, pages 195–198, 2006.

[42] 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*, pages 365–374, 2010.

[43] G. Venolia. Textual allusions to artifacts in software-related repositories. In *Proc. of the Intl. Workshop on Mining Software Repositories*, pages 151–154, 2006.

[44] T. Zimmermann, R. Premraj, J. Sillito, and S. Breu. Improving bug tracking systems. In *Proc. of the Intl. Conf. on Software Engineering — Companion Volume*, pages 247 –250, May 2009.

[45] R. W. Zmud. Management of large software development efforts. *MIS Q.*, 4(2):45–55, June 1980.