in*Bug: Visual Analytics of Bug Repositories

Tommaso Dal Sasso, Michele Lanza
REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

***Abstract*—Bug tracking systems are used to track and store the defects reported during the life of software projects. The underlying repositories represent a valuable source of information used for example for defect prediction and program comprehension. However, bug tracking systems present the actual bugs essentially in textual form, which is not only cumbersome to navigate, but also hinders the understanding of the intricate pieces of information that revolve around software bugs.**
We present in*Bug, a web-based visual analytics platform to navigate and inspect bug repositories. in*Bug provides several interactive views to understand detailed information about the bugs and the people that report them.
The tool can be downloaded at *http://inbug.inf.usi.ch*

I. Introduction

Due to the complexity and size of non-trivial software projects, the development of a system is always accompanied by software defects, or bugs. To manage these defects, modern software projects use bug tracking systems (also known as bug trackers or issue trackers), such as Jira or Bugzilla. With bug trackers, end users and developers can report bugs they encountered while using the system, usually by means of custom web interfaces, where one can enter details about a specific bug, creating a so-called *bug report*. A typical bug report, such as the one depicted in Figure 1, contains information about (1) the title and id of the bug, (2) the user who reported the bug and the people involved in its history, (3) its current status, (4) its opening and closing date, (5) its last modification date, (6) the project to which the bug report pertains, (7) events (such as changes of the people assigned to the bug report, etc.) during the life cycle of the bug, etc. The example bug report depicted in Figure 1 is from a specific bug tracker, FogBugz [1], but it does not differ significantly from the reports recorded with other bug trackers.

Various researchers have mined and used the information stored by bug trackers to perform several types of analyses, such as identifying duplicate bug reports [1], measuring the quality of a report [2], predicting future defects [3], performing traceability linking [4], locating features [5], ameliorating bug triaging decisions [6], etc. The actual goal however is to ease the life of developers in the handling of bug reports, as part of the development process.

One problem is that bug reports are disconnected from the software system they pertain to, and it is up to the developers to restore the link between a bug report and the interested components of a system. Another problem is that bug reports, such as the one depicted in Figure 1, are displayed on individual web pages that list their properties, making them cumbersome to handle and making it also difficult to obtain a "big picture" of the existing open bug reports and how they overall affect the system they pertain to. Moreover, this information is stored and presented as text, which makes it hard to understand the properties of a bug report.

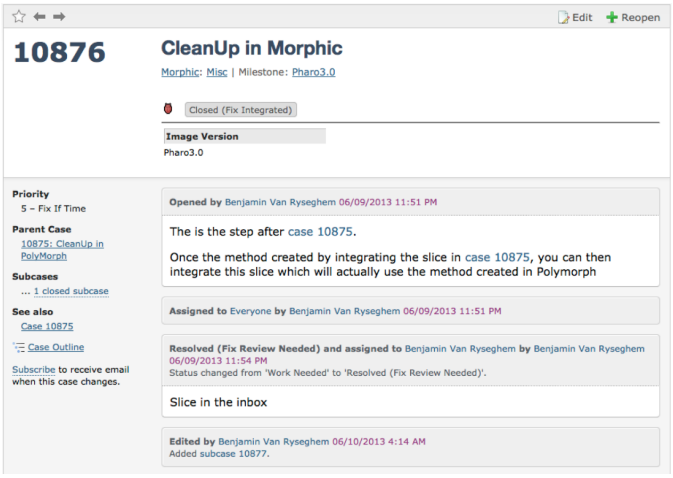

Fig. 1. Example bug report in the FogBugz bug tracking system.

We present in*Bug, a web-based bug analytics platform, that eases the inspection, navigation, and comprehension of bug repositories, mostly by means of interactive visualizations. in*Bug provides an entry-level big picture overview to browse the content of a repository, and a detailed, complementary, interactive, and finer-grained view to understand detailed information about the bugs and the people that report them.

Other researchers have produced custom visualization of bugs, such as D'Ambros et al. [7], [8] who proposed visualizations that tried to depict the complex information revolving around bugs, which are de facto independent entities when it comes to program comprehension, and not mere side effects of the evolutionary process that software systems are subjected to. While D'Ambros et al. only created standalone depictions of information taken from BugZilla, our goal with in*Bug is to depict live data from a bug tracker, namely FogBugz. In the near future, we plan to offer in*Bug as a complementary means to inspect and analyze information pertaining to bugs reported in the context of the many projects that make up the software ecosystem revolving around the Pharo[2] open-source community.

We present the current features of in*Bug, discuss its current implementation, and illustrate its usage.

[1] http://www.fogcreek.com/fogbugz/

[2] http://www.pharo-project.org

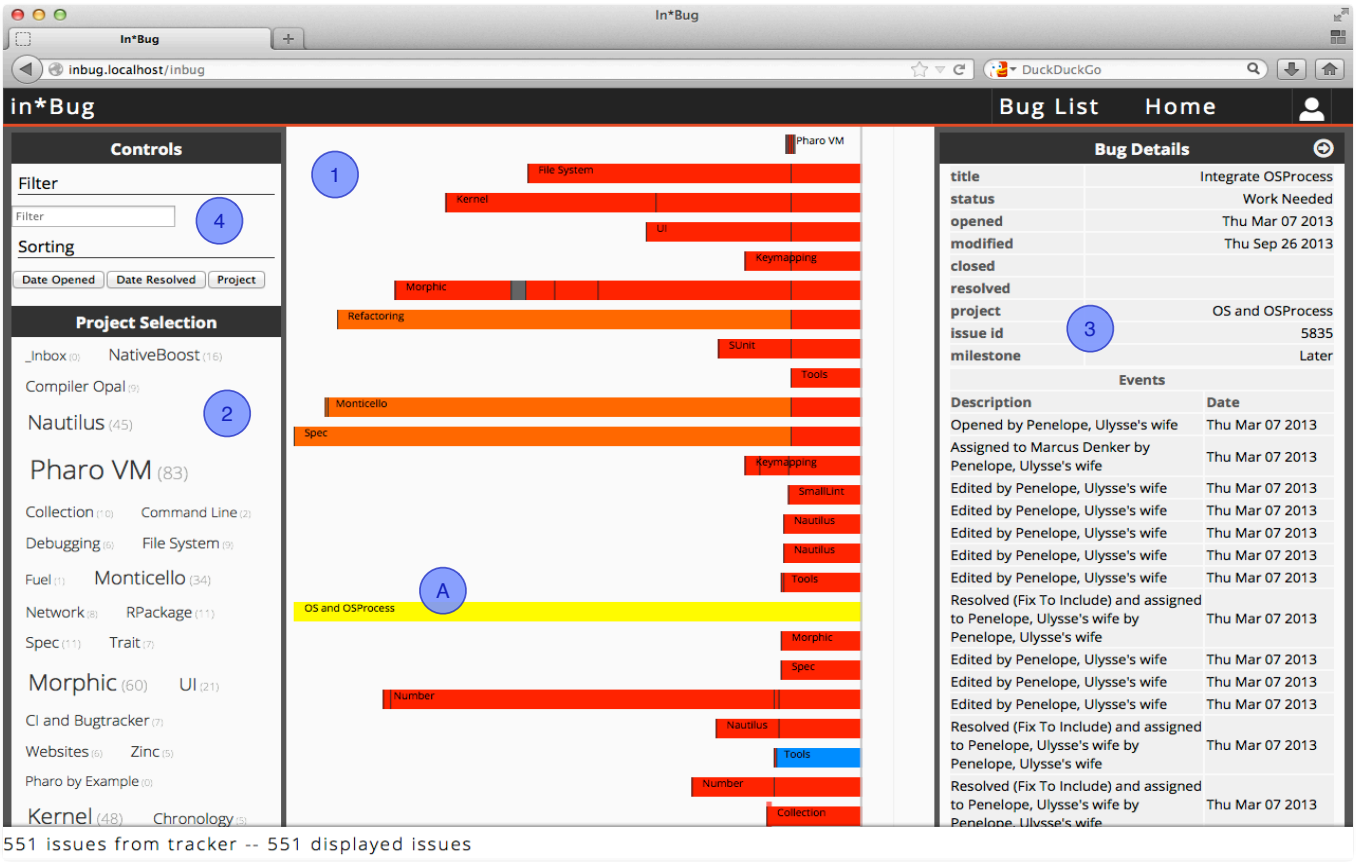

Fig. 2. Main user interface of in*Bug

II. IN*BUG IN DETAIL

A. Main view

Figure 2 depicts the main user interface, composed of the following panels:

Bug lifetime panel (1). This view depicts the bug reports contained in the bug repository, showing their duration (as a horizontal stacked bar chart) and status (using different colors, listed in Table I).

TABLE I
BUG REPORT EVENT COLOR CODES

Active		orange	Work Needed		red
Closed		gray	Resolved		dark gray
Working On		blue	On Hold		cyan
Unknown		light grey	Selected		yellow

In Figure 2 one specific bug (marked as A) is under focus. The vertical line to the right indicates the current date, making it also clear whether a bug report is still active or not (if it is, it will touch that line). This view also helps the developer to evaluate the complexity of a bug report by summarizing the events occurred during its lifetime.

Project selection panel (2). In this panel the user can pick the projects whose bugs she is interested in. All projects are shown as a tag cloud, where the tag size indicates the number of bugs reported for the project, also indicated with numbers between parentheses close to the name of the projects.

Details panel (3). This panel provides all the information reported about the bug report under focus in the bug lifetime panel. This panel present both the metadata and the list of events that happened during the lifetime of a bug, including description and date of each event. The metadata is presented as extracted from the bug repository, *e.g.,* the opening date, the status, the last modification date, etc.

Filter and options panel (4). This panel allows the user to sort and filter bugs. The three default sorting criteria order the issues by project, opening date, or date in which the bug has been resolved. The *filter* field offers the possibility to enter either regular expressions or pieces of *Smalltalk* code as queries, allowing the users to submit custom made queries to filter bugs.

B. Details of a bug

This view (see Figure 3) presents a detailed representation of a specific bug report. Each section provides a description of the element that compose the description of a defect.

Bug Report Metadata (1). The first panel summarizes the important metadata of the bug report: the id, the last modification, the current status, the opening date and possible

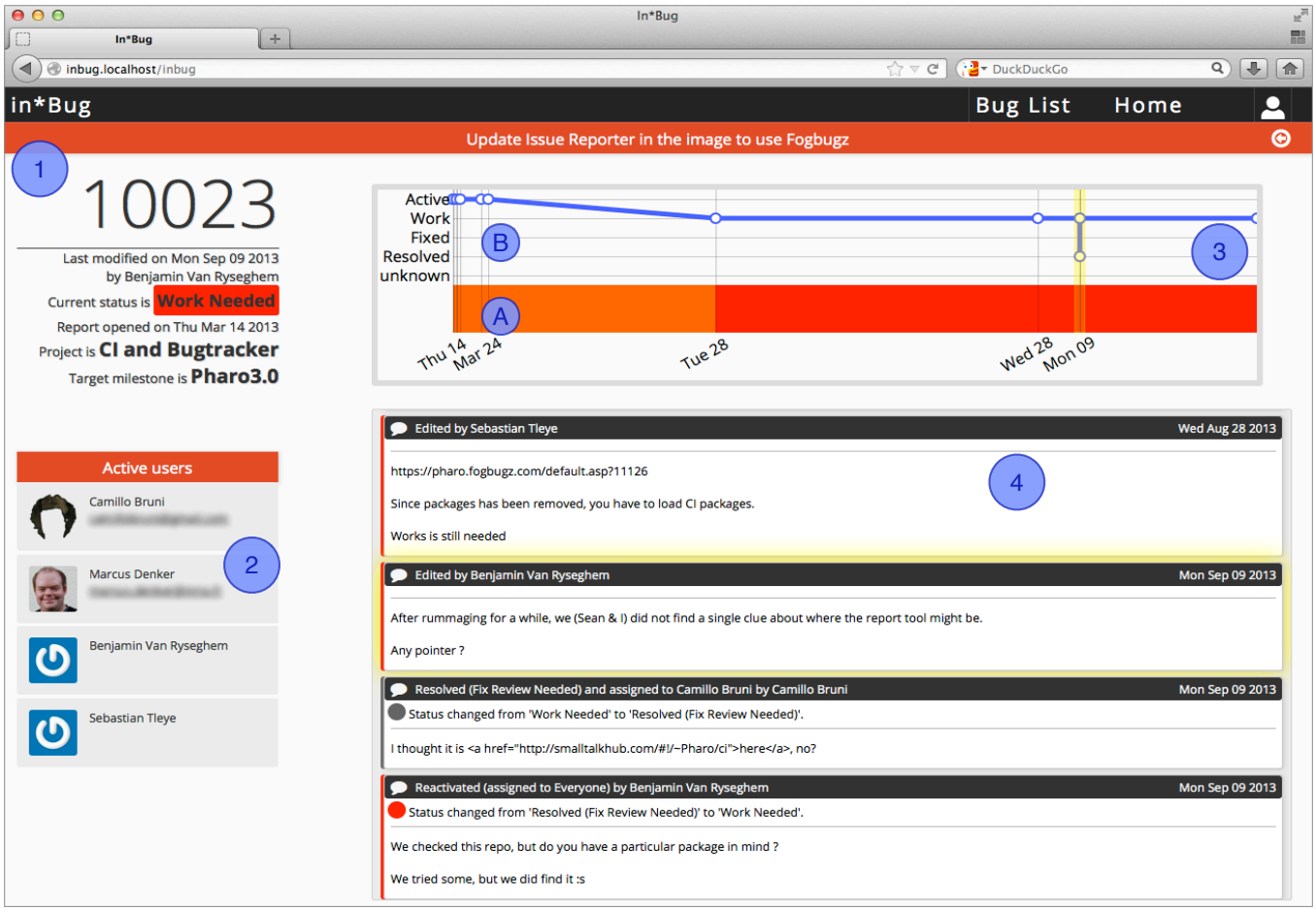

Fig. 3. in*Bug details page showing the properties of a bug report

closing date, the project and the target milestone for the issue resolution.

Users List (2). This panel gives an overview of the people involved in the evolution of the bug. In particular, the list displays the information of each user that performed an action on the issue, that was stored as an event. The details include the picture of the user, the user name and the user's email address[3], to contact the people working on an issue.

Bug Report Life Visualization (3). This panel shows a visualization of the life of a bug report during time. The left border represents the date the issue was opened, the right border represents the moment the bug was closed, or the current date if the bug report is still active. The section (A) proposes the same visualization of the list view in the main view (II-A), emphasising the status changes during time. The section (B) shows a line diagram where the height represents the criticality of the status (*i.e.,* fixed is the lowest and active is the hightest) and highlighting each event with a circle.

Event Interactive View (4). This is a list of all the events that compose a bug report. It shows the metadata of the event and whether it is an automatic event or an event generated by a user. It also detects and highlights the patches of code submitted to the tracker for the issue resolution, and provides a link to download and inspect the patch. The user can click on an event to highlight it both in the events list and in the bug report lifetime visualization. Figure 4 shows a detail of the event list, where we can observe the three types of events: (A) shows a comment by an user; (B) shows a submitted patch. The upper left icon offers a link to the repository page of the patch; (C) indicates events automatically generated from bots in the tracker.

Inspired by more semantically rich and elaborated views, such as Ogawa et al's storylines [9] or Kuhn and Stocker's storytelling timelines [10], the left border of each event is colored according to the status of the event, to help the user to keep track of the evolution of the bug while inspecting the list of events.

[3]We obfuscated the email addresses in the figure for privacy reasons.

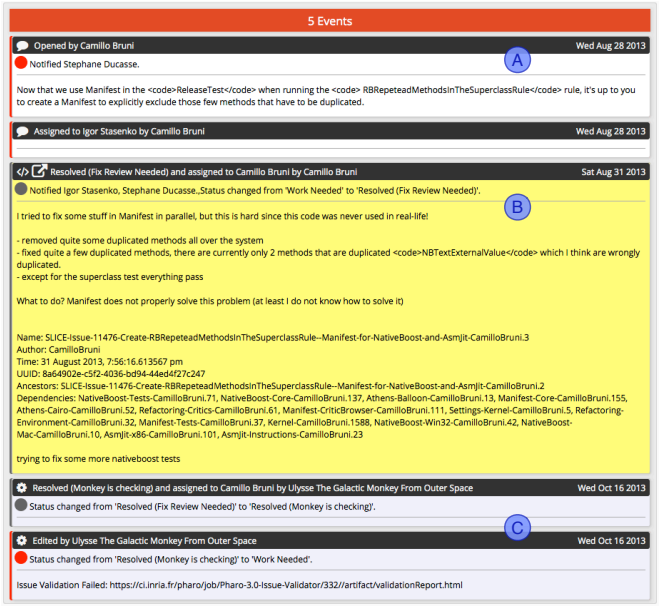

Fig. 4. List of events in a bug report

C. Implementation & Current Dataset

in*Bug is a web application built on top of the *Pharo Smalltalk*[4] environment. It uses the *Seaside*[5] web framework to provide the data stored in a *MongoDB* database and implements a *RESTful API* to communicate with the client. The client interface is implemented in *JavaScript* using the data manipulation and visualisation library *D3.js*[6].

in*Bug is currently targeted at a specific FogBugz repository revolving around the Pharo ecosystem. In Table II we provide a summary of the currently available data, which has already reached considerable complexity.

TABLE II
SUMMARY DATA OF THE *Pharo* BUG TRACKER

Number of projects	46
Number of bug reports	8,666
Number of open bug reports	613
Total number of events	79,437
Average events per issue	9

in*Bug also provides links to patches on *SmalltalkHub*, a source code repository to store versioned Smalltalk code. In Figure 5 we can see how these three services interact.

The bug reports data is imported from FogBugz and stored in the MongoDB repository. The web application then loads the data and present it in the list view of the main interface. The details of a single report are presented in the details view, where the user can follow a link that leads to a patch submitted to SmalltalkHub.

[4]http://www.pharo-project.org
[5]http://seaside.st
[6]http://d3js.org/

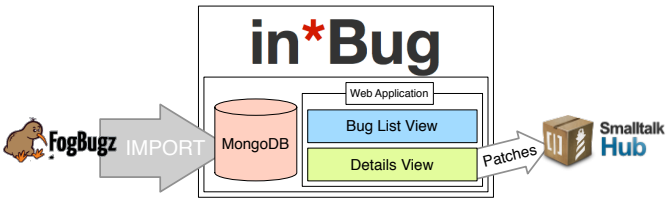

Fig. 5. The interactions of in*Bug with the *FogBugz* and *SmalltalkHub* services

III. Conclusion

We have presented in*Bug, a web-based visual analytics platform to explore the content of a bug repository. in*Bug allows to get a complete overview of a whole repository, as well as detailed and meaningful information on a single bug report, either through visualizations that allow to interact with the data, or with the query engine embedded in in*Bug that allows the user to submit queries and dialog directly with the bug reports.

We intend to provide further visualizations that describe the resume data of a single bug repository, to ease and improve the comprehension of the evolution of a software project during time.

The approach of in*Bug is general enough to be applied to any bug tracking system. Since we want to propose in*Bug as a tool for practical development, we focused on the Pharo platform and we targeted its community. We plan to improve in*Bug and refine the existing visualizations based on feedback obtained from Pharo users.

Acknowledgements. We acknowledge the Swiss National Science foundation's support for project 146734 "HI-SEA".

References

[1] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*. ACM, 2008, pp. 461–470.
[2] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *SIGSOFT FSE*, 2008, pp. 308–318.
[3] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.
[4] T. F. Bissyandé, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Réveillère, "Empirical evaluation of bug linking," in *CSMR*, 2013, pp. 89–98.
[5] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.
[6] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proceedings of the 28th International Conference on Software Engineering (ICSE 2006)*. ACM Press, 2006, pp. 361–370.
[7] M. D'Ambros and M. Lanza, "Bugcrawler: Visualizing evolving software systems," in *Proceedings of CSMR 2007 (11th IEEE European Conference on Software Maintenance and Reengineering)*. IEEE CS Press, 2007, pp. 333–334.
[8] M. D'Ambros, M. Lanza, and M. Pinzger, ""a bug's life" — visualizing a bug database," in *Proceedings of VISSOFT 2007 (4th IEEE International Workshop on Visualizing Software For Understanding and Analysis)*. IEEE CS Press, 2007, pp. 113–120.
[9] M. Ogawa and K.-L. Ma, "Software evolution storylines," in *SOFTVIS*, 2010, pp. 35–42.
[10] A. Kuhn and M. Stocker, "Codetimeline: Storytelling with versioning data," in *ICSE*, 2012, pp. 1333–1336.

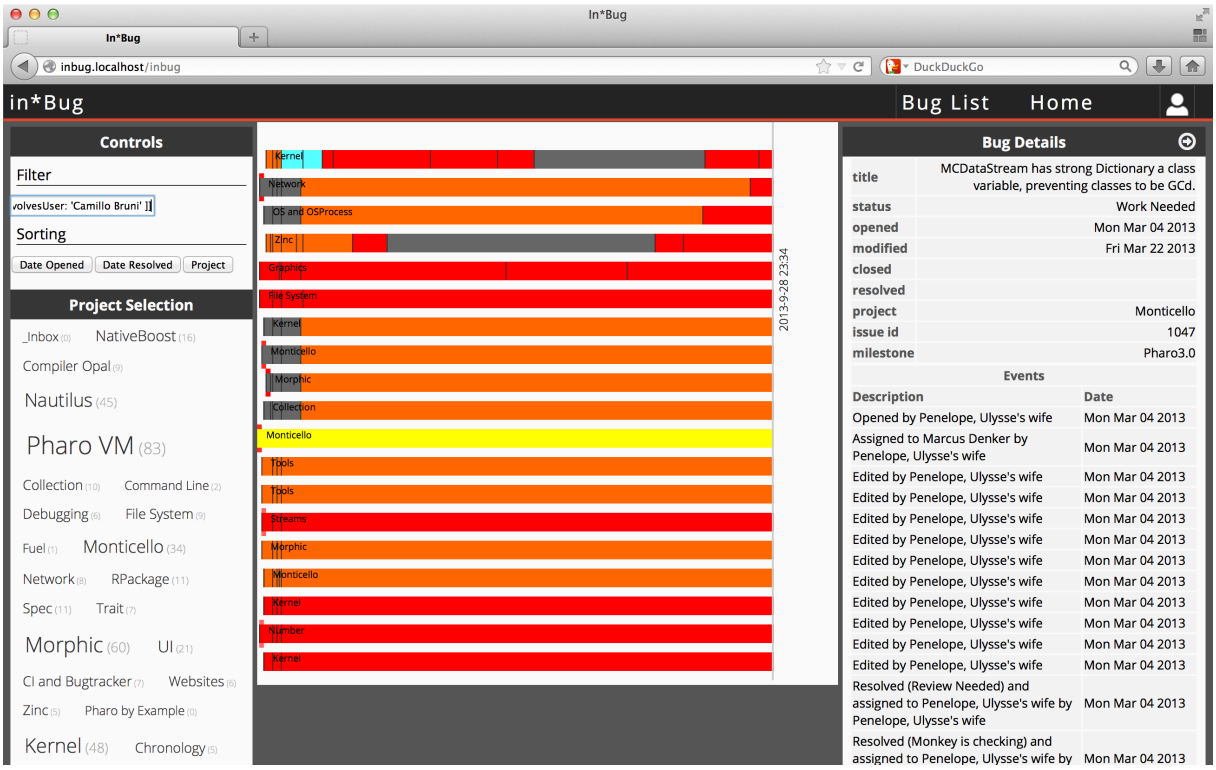

Fig. 6. The result of a Smalltalk query

Appendix - Description of the demo

We foresee a very interactive demo, where in front of the audience we would use in*Bug to navigate the bug repository mentioned in Table II. In the following we do provide a plausible demo scenario, but would much rather prefer to have the audience give input on which bugs we would analyze together with them.

1) **Bug List View.** The beginning of the demo is to present the concepts behind the main interface of in*Bug. We are going to introduce the idea of a bug report as an independent entity and describe the visual representation chosen to describe a report. We then describe purpose of each panel, starting from the bug list: the possibility to examine a set of issues and their relation in terms of time collocation (see Figure 2).
2) **Bug report manipulation filters.** We will show the capabilities of in*Bug to sort and filter the interesting reports. We will show an example of query submitted in Smalltalk, and show how we can find an interesting report with a visual inspection. Figure 6 shows the result of the query:

```
[ :each | each events size > 20 and: [
  each involvesUser: 'Camillo Bruni' ]]
```

3) **Details View.** We will select a bug report to inspect and visualize its properties with the details view presented in Figure 7. We will present the metadata displayed, the users involved in the report resolution, then we will see how the visualization can provide an immediate feedback on the life of a bug report, and its current status.

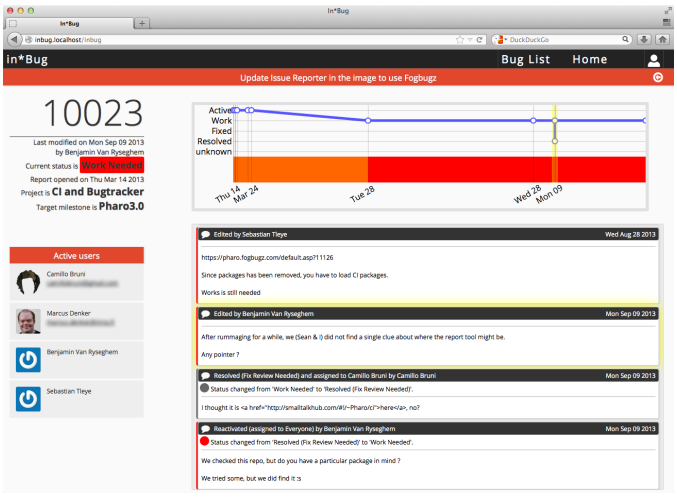

Fig. 7. in*Bug details page showing the properties of a bug report

4) **Events and Patches.** We will examine the list of the events and the possible types of events described. We will show when in*Bug can detect if an event is a user comment, an automatic event or a patch submitted to fix the problem. We will show how in*Bug provides a link to directly retrieve the patch and examine it.

We plan to wrap up by having a discussion about other extensions we are currently implementing, and showing off some of the directions we are currently pursuing.