A Closer Look at Bugs

Tommaso Dal Sasso and Michele Lanza

REVEAL @ Faculty of Informatics — University of Lugano, Switzerland

Abstract—The evolution of non-trivial software systems is accompanied by unexpected behaviour and side-effects, referred as bugs or defects. These defects are reported to and stored in bug tracking systems, which contain descriptions of the problems that have been encountered. However, bug tracking systems store and present bug reports in textual form, which makes their understanding dispersive and unintuitive.

We present an approach to display bug reports through a web-based visual analytics platform, named in*Bug. in*Bug allows users to navigate and inspect the vast information space created by bug tracking systems, with the goal of easing the comprehension of bug reports in detail and also obtain an understanding "in the large" of how bugs are reported with respect to one system or to an entire software ecosystem.

I. INTRODUCTION

Modern non-trivial software projects use bug tracking systems (also known as bug trackers), such as Jira and BugZilla, to manage the bugs that are reported. The repositories created by such bug trackers are a valuable source of information, and they have been used in the past to perform several types of analyses, such as predicting future defects [1], performing traceability linking [2], locating features [3], ameliorating bug triaging decisions [4], etc.

Bug reports contain both structured and unstructured data. Examples of the former are the author who reported the bug, an id, a timestamp, etc. Examples of the latter are all natural language comments that come with such reports, such as the description of how and when the bug was encountered, and also comments posted by others to discuss the bug report. Figure 1 depicts an example bug report[1], in terms of how it is presented to a user who wants inspect such a bug report: It is essentially a web page, which can also become very long. In the example report we cut out 13 more events that happened during its life time: Essentially, the report is a wall of text that spans over multiple screens, increasing the effort needed to get a complete picture of the history of the bug.

Indeed, bug reports are complex constructs: D'Ambros et al. have shown that they possess complicated life cycles, which makes them non-trivial to comprehend [5].

We present a novel approach to visualize bug reports, through in*Bug, a web-based software visual analytics platform. in*Bug allows users to navigate and inspect the vast information space created by bug tracking systems, with the goal of easing the understanding of bug reports in detail and also obtain an understanding "in the large" of how bugs are reported with respect to one system or to an entire software ecosystem.

[1]Taken from the bug tracker *FogBugz*, which is used by the *Pharo open source community (http://pharo-project.org)*.

Fig. 1. Partial visualization of a *FogBugz* bug report

II. IN*BUG: AN APPROACH FOR BUG VISUALIZATION

in*Bug is a web application built on top of the *Pharo Smalltalk*[2] environment. It uses the *Seaside*[3] 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*[4].

[2]http://www.pharo-project.org
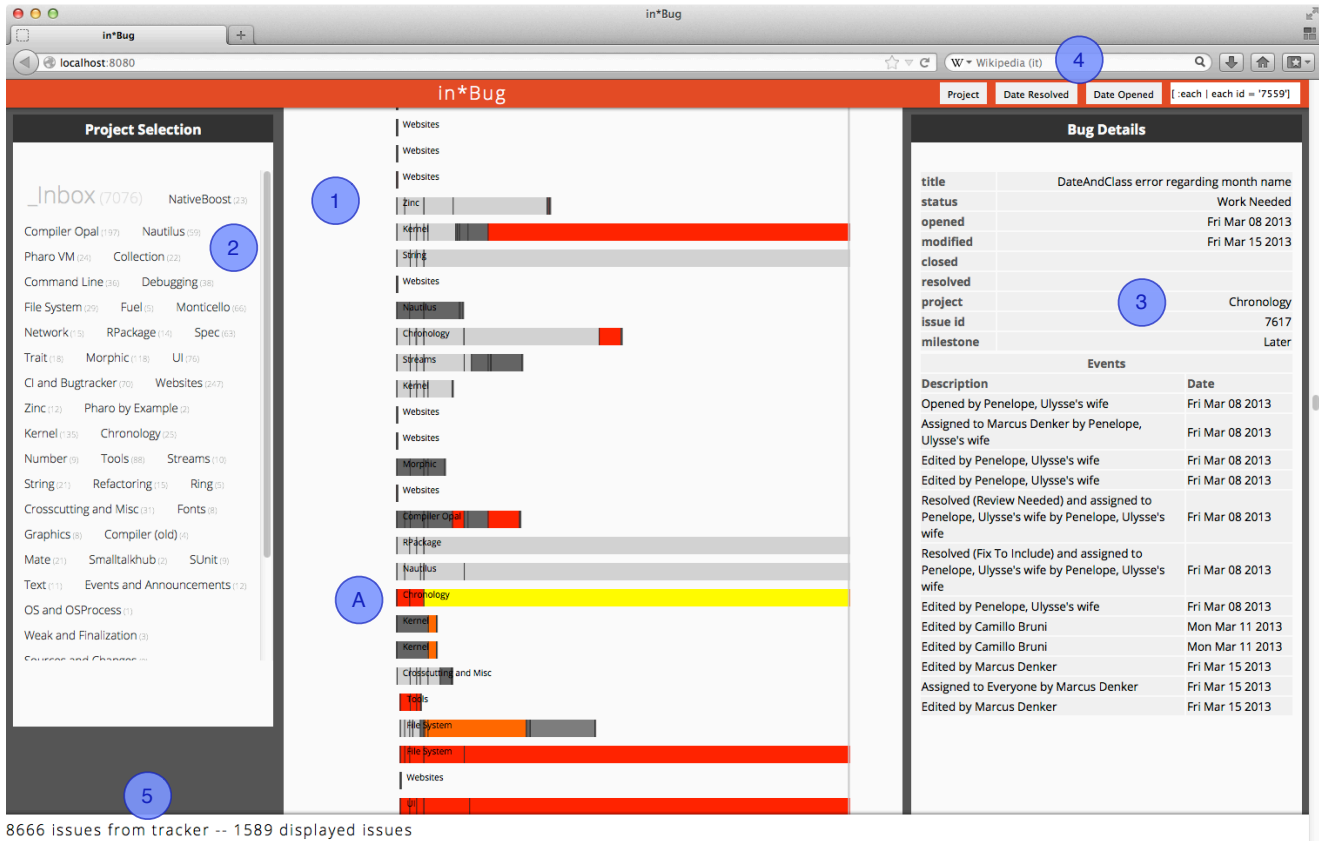[3]http://seaside.st
[4]http://d3js.org/

Fig. 2. in*Bug Main View

in*Bug is currently targeted at a specific FogBugz repository revolving around the Pharo ecosystem. In Table I we provide a summary of the currently available data.

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

A. in*Bug Main View

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

1) **Bug lifetime panel.** This view depicts the bugs contained in the bug repository, showing their duration (as a horizontal stacked bar chart) and status (using different colors). Each bug tracking system proposes a set of statuses that an issue can acquire. We grouped these statuses into 5 categories, and assigned them the following color codes:

Active		orange	Work Needed		red
Closed		gray	Resolved		dark gray
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.

2) **Project selection panel.** 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 size indicates the number of bugs reported for the project, also indicated with numbers between parentheses close to the name of the projects.

3) **Details panel.** This panel provides all the information reported about the bug report under focus in the bug lifetime panel. This panel presents 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 or the last modification date. The user can highlight an event to view a popup with the complete description of the event and its comments.

4) **Filter and options panel.** 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.

5) **Status bar.** This panel shows a quick status of the application. It displays the total number of the bugs in the repository, as for the number of bugs currently selected
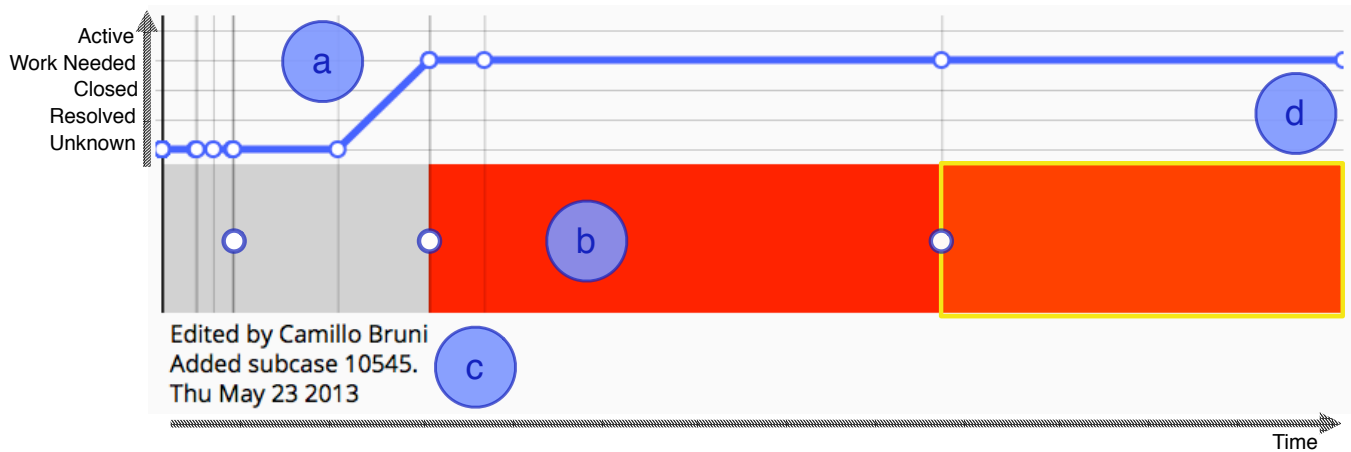
Fig. 3. Fine-grained View of a Bug Report

and visualized.

B. A Fine-grained View of a Bug Report

Aside from the interpretation difficulties that a text based interface presents, Figure 1 shows that a bug can be composed of a significant number of events. These events are sequential and are triggered by people. They describe the changes in the properties of a bug, such as the opening of a bug, a change of status, the addition of other pieces of information, such as stack traces and screenshots, the assignation to a user who is supposed to fix it, the registration of other people who are interested in that bug, and ultimately also the release (fix) of a bug, who usually comes with a slice of source code. These events are the building blocks of a bug, and visualizing them is the key for its understanding.

Figure 3 shows our current approach for the detailed depiction of a bug report, composed of three elements that illustrate its different aspects.

(a) **Bug Health** presents the evolution of status of a bug during time. The changes are presented as a line chart that shows the different states in time. The lowest possible value represents the *closed* status, the highest represents the *active* status.

(b) **Bug History** shows a bug report as a timeline composed of its events. If an event changes the status of the bug report, it also changes the color to render the bar. Hovering over a block of the timeline renders the details of the bug report in the details panel. If a new user is involved in the event (*e.g.,* a bug is reassigned to someone else), a small circle is rendered at the beginning of the event.

(c) **Bug Event description** is used to display the details of the events. It shows the date and the description of the highlighted event (in yellow in Figure 3), such as comments or other pieces of information (*e.g.,* stack traces, screenshots) that were added for that event.

(d) **Present time**. The right side of the chart represents the current time. In this case we can observe that the bug is currently still open and its status is "Work needed".

C. A Coarse-Grained View of all Bug Reports

This is the first view, depicted in Figure 4, that is presented to the user, a time-ordered multi-layer stacked chart, which shows how many bug reports were in which status and at which time.
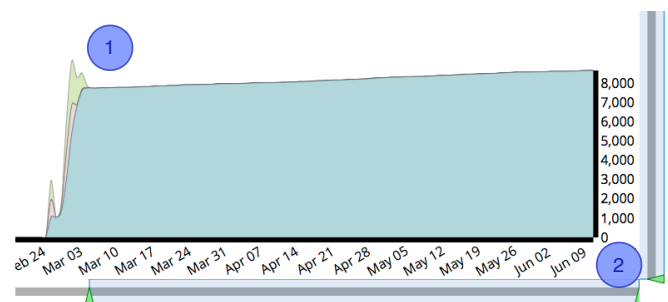

Fig. 4. in*Bug summary view

Alternatively, the user can also display the various systems to which the bugs belong. This view offers means of interaction to home in on a subset of the bug reports, by using the two sliders at the bottom and on the right. Alternatively, the user can also click on one specific layer, and then jump to the main view depicted in Figure 2.

III. A USAGE EXAMPLE

We illustrate, through a brief scenario, how in*Bug can be used to search a bug repository for bugs that need work. We impersonate the user "Marcus Denker", who is looking for bugs assigned to him. In the main view of in*Bug he uses the query engine, composing the *Smalltalk* query shown in Figure 5 to find all the opened bugs that are assigned to him.

```
[ :each | each isActive
and: [ each involvesUser: 'Marcus Denker' ]]
```

Fig. 5. *Smalltalk* query to retrieve the issues assigned to "Marcus Denker"

The query returns 461 issues, which he can sort by date or by project. However, these are too many issues to be manually inspected, so he needs to further refine the query. He restricts the research to the reports opened on the 4th April 2013. Figure 6 shows the updated query.

```
[ :each | each isActive
and: [(each involvesUser: 'Marcus Denker')
and: [each dateOpened = (Date year: 2013 month: 3 day: 4)
]]]
```

Fig. 6. *Smalltalk* query to retrieve the issues assigned to "Marcus Denker" opened on April 4 2013

The query now returns the 13 issues depicted in Figure 7.

Fig. 7. The result of query in Figure 6

Mr. Denker sees that the first bug had some initial activity, but is stalling since then. He uses the fine-grained view to have a complete picture of the issue, depicted in Figure 8.
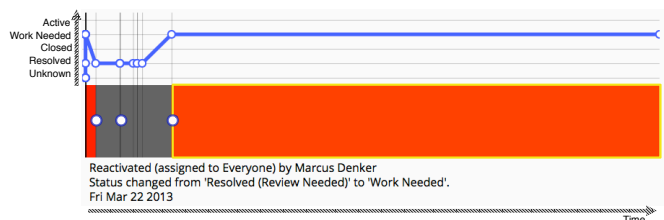

Fig. 8. Detailed view

The view shows that the bug was fixed and then reopened. It is now in status "Work Needed". Mr. Denker highlights the last event and sees that the bug was reopened by himself the 22nd of March 2013. Since the issue was marked as "Resolved" and then reopened, and it had no activity in a long time, Mr. Denker decides that this is the bug he wants to work on.

With a simple click he is now redirected to the bug's page in the bug tracker.

IV. RELATED WORK

Surprisingly, there is only little related work dedicated to the visualization of bug repositories.

D'Ambros *et al.* [6] proposed a visualization for bugs as independent entities. Knab *et al.* [7][8] proposed a set of visualization to ease the process of understanding the data in an issue tracker and find hidden patterns. They also tested their tool in an industrial context. Hora *et al.* [9] proposed a visual exploration of the bug repository, considering bugs as first class entities, and linking them to other software artifacts.

All these approaches focus on retrospective analyses. We believe that while conceptually interesting, there is little practical utility in such a thing, since after all the goal is not to look at bugs, but to actually fix them. This implies that even the most elaborated techniques are of limited actionability, since the bug fixing process takes place in a different space, namely the integrated development environment (IDE). In our case the goal is not to stop at the mere visualization, but then establish a first-class link to the development environment. We envision an environment where going from a specific bug report in in*Bug to the context where the reported bug can be fixed is a mere mouse click away. Our future efforts will be focused on this goal.

V. CONCLUSION

We presented a novel approach to visualize bug repositories, implemented in a visual analytics platform named in*Bug. in*Bug currently offers a set of diverse views and interaction means to analyze bug repositories "in the large" and then focus on specific bugs "in the small". Our short-term goal is to expand the functionalities offered by in*Bug, refine the visualizations, and then put in*Bug in "production mode", by offering it to the Pharo open-source community.

ACKNOWLEDGEMENTS

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] P. Knab, B. Fluri, H. C. Gall, and M. Pinzger, "Interactive views for analyzing problem reports," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 527–530.

[8] P. Knab, M. Pinzger, and H. C. Gall, "Visual patterns in issue tracking data," in *New Modeling Concepts for Todays Software Processes*. Springer, 2010, pp. 222–233.

[9] A. Hora, N. Anquetil, S. Ducasse, M. Bhatti, C. Couto, M. T. Valente, and J. Martins, "Bug maps: A tool for the visual exploration and analysis of bugs," in *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. IEEE, 2012, pp. 523–526.