Delft University of Technology
Software Engineering Research Group
Technical Report Series

Interactive Views for Analyzing Problem Reports

Patrick Knab, Beat Fluri, Harald C. Gall, and Martin Pinzger

Report TUD-SERG-2009-030

TUD-SERG-2009-030

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Accepted for publication in the Proceedings of the International Conference on Software Maintenance (ICSM), 2009, IEEE Computer Society.

Interactive Views for Analyzing Problem Reports

Patrick Knab, Beat Fluri, Harald C. Gall
Dep. of Informatics, Univ. of Zurich, Switzerland
{knab,fluri,gall}@ifi.uzh.ch

Martin Pinzger
Dep. of Software Technology, Delft Univ. of Tech., The Netherlands
{m.pinzger@tudelft.nl}

Abstract

Issue tracking repositories contain a wealth of information for reasoning about various aspects of software development processes. In this paper, we focus on bug triaging and provide visual means to explore the effort estimation quality and the bug life-cycle of reported problems.

Our approach follows the Micro/Macro reading technique and uses a combination of graphical views to investigate details of individual problem reports while maintaining the context provided by the surrounding data population. This enables the detection and detailed analysis of hidden patterns and facilitates the analysis of problem report outliers.

In an industrial study, we use our approach in various problem report analysis scenarios and answer questions related to effort estimation and resource planning.

1. Introduction

In the EUREKA/ITEA[1] SERIOUS[2] project we worked together with industrial partners to *Investigate means to improve the planning and resource allocation in a software project by analysing change request and problem report data*.

We were given access to an issue tracking repository containing thousands of problem reports (PR) from a multi-year, multi-site software development project in the consumer electronics domain.

Since the statistical analysis of the data showed ambiguous results, mainly due to a strong presence of outliers and a heavily right-skewed data distribution, we developed a flexible interactive visualization approach which we present in this paper. The approach is based on the Micro/Macro Reading idea from Tufte [6] and provides a combination of Overview and Details-on-demand. It currently comprises four simple and easy to understand views which can be linked in various ways. This integration enables sophisticated analyses by drilling down to details while preserving the big picture.

With this approach, we can answer questions such as: What is the overall quality of effort estimates? Are there more and less accurate analyzers? What are the patterns of the less accurate analyzers? Do deviations from the main process path have an effect on the quality of the estimates? Which process steps take the most time? What is the influence of priority on the duration of the process steps?

To answer these questions we used problem report information from our industrial data set. Out of the many attributes associated with a problem report we focus mainly on the following six:

estimatedEffort	The estimated total effort in person hours to fix the problem
actualEffort	The actual total effort in person hours
analyzer	The person responsible to analyze the problem and estimate the effort
owner	The manager responsible for this problem report
priority	The priority assigned to the problem, possible values are: low, medium, high, and top
severity	The severity assigned to the problem, possible values are: A, B, C, or D. A means that the customer does not accept the product and D means that the customer does not notice the problem.

The *estimatedEffort* is an estimate done by the *analyzer* who can be either a specially assigned person or the problem report owner himself. The *actualEffort* is the total effort actually used to fix the problem. It includes the analysis, the

[1] http://www.itea2.org/

[2] Software Evolution, Refactoring, Improvement of Operational & Usable Systems

resolution as well as the evaluation effort.

In addition to the problem report attributes, we also extracted PR life-cycle data from the log files. The log files contain all changes to all the PR fields including status changes, *e.g.*, a PR is changed from *submitted* to *in_analysis*. All the PR (problem report) life-cycle states are shown in Figure 3.

2. Visualization Building Blocks

In this section, we present our visualization approach which integrates multiple views that can be combined in various ways.

For the visualization of effort measures, we use **Polymetric Views** [5]. In Figure 1 the basic concepts of our visualization are shown: the width of the boxes is determined by the value of the *estimatedEffort* and the height by the value of the *actualEffort*. The effort measure is the sum of all efforts that were exerted to resolve the issue described by the problem report.

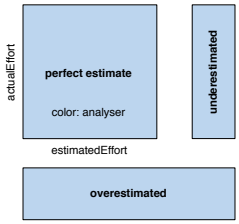

Figure 1. Polymetric effort shape patterns

With this information mapping we can get a quick and effective overview over the quality of estimates: balanced estimates (square boxes) constitute problems that were estimated accurately with respect to the actual resolution effort needed; underestimated (boxes that are thin and tall) and overestimated (boxes that are short and broad) PRs can also be spotted easily.

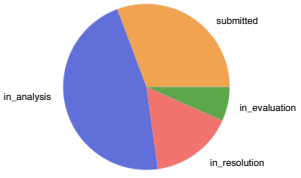

Figure 2. Pie chart with process step lengths

To visualize the duration of process steps we use a **pie chart visualization**. In Figure 2 we show a single pie with the mapping to the four process steps: *submitted, in_analysis, in_resolution, in_evaluation*. The size (*i.e.*, the area) of the pie is mapped to the total time from the creation of the PR until it was closed.

With the addition of a **PR state transition** view shown in Figure 3 we provide a new angle and starting point for

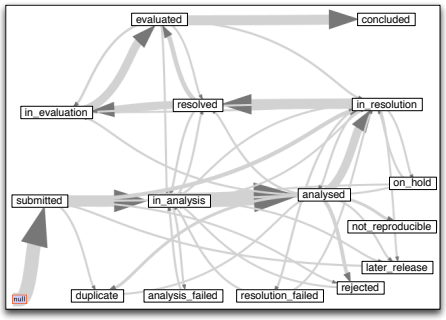

Figure 3. PR life-cycle view

an analysis. In this view configuration all occurring state transitions are displayed in an aggregated form. The width of the arrows between the states is mapped to the number of problem reports that exhibit the corresponding transition. With the presented layout we can see the main path that most PRs take, and that there are quite a few PRs deviating from this standard path.

3. Case Study

To better represent the typical analysis process, we state an overall goal that summarizes the intended direction for each exploration step. The case study is based on a multisite

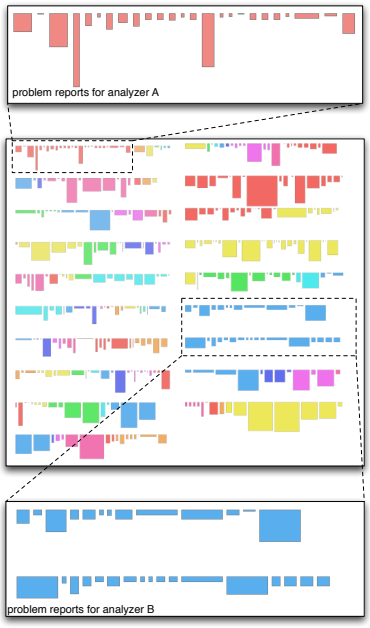

Figure 4. Effort view on PRs grouped and colored according to the analyzer

project spanning five years. The issue tracking repository contains approx. 20'000 problem reports (PRs) that were handled by 368 distinct *analyzers*.

Goal: Get an overview of the quality of estimates and how different analyzers are doing In Figure 4 we configured a view that groups and colorizes PRs according to the analyzer that did the estimates. Looking for patterns we can see, that there is a mix of estimation errors as well as some fairly well estimated PRs. We can also see, that there are about six analyzers that do the most analyses (more than 10) and a few others that have only a small number of associated PRs. There are obviously some differences in the performance of the analyzers. Considering the selections for *Analyzer A* and *Analyzer B*, that we present in a detail view above and below the main view, we can see that *Analyzer A* mainly underestimates the effort, whereas *Analyzer B* mainly overestimates. Looking at the differences in the width of the boxes (*estimatedEffort*) of *Analyzer A* we get the impression that he uses mostly one standard estimate for all his PRs where *Analyzer B* has more variability in his estimates. It might be worthwhile to discuss their estimation processes with other team members to improve the quality of their estimates.

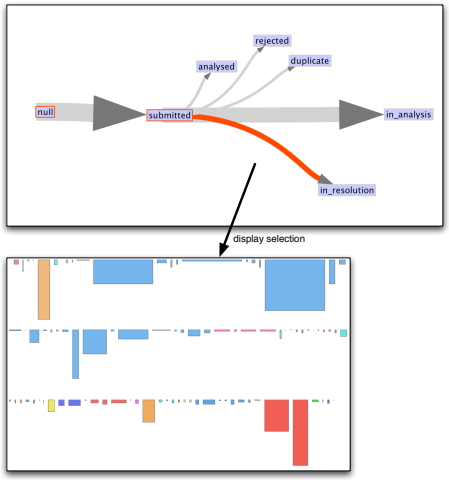

Figure 5. PRs that skipped the analysis phase, color indicates the PR's owner

Goal: Analyze process paths, find exceptional routes and analyze them with regard to estimation quality In Figure 5 one can see a detail view of our state transition visualization (top view) were the transition *submitted* to *in_resolution* is selected. Only a small amount of all PRs take this shortcut while most of them go first into the *in_analysis* phase. To assess the effects on estimation quality we created a polymetric view from the selected PRs (bottom view). In this view the PRs are grouped and colored according to the responsible problem report owner, as the owner has the task to assign an *analyzer*. Here we observe that there is one owner who is responsible for about one third of the PRs that skipped the analysis phase. Examining in more detail we see that the quality of the estimates is, at least, questionable. There are mostly over- and underestimated reports and only some square-like shapes denoting accurate estimates.

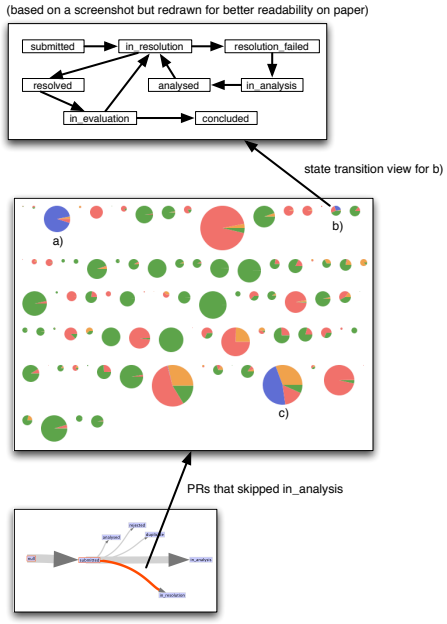

Figure 6. PRs that skipped the analysis phase

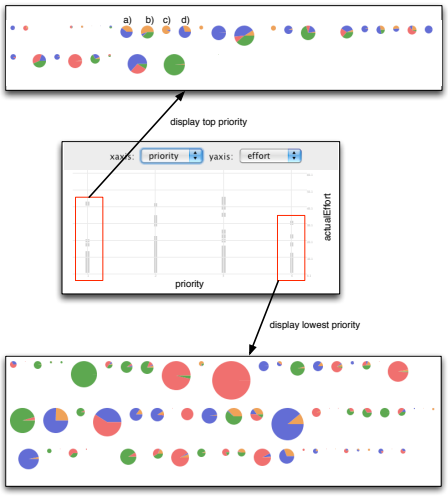

Figure 7. Pie view of PRs with priority top and low

Goal: Further explore the attributes of PRs that skipped the analysis phase Using our pie chart visualization, as shown in Figure 6, we can explore additional properties of the problem reports that skipped the initial analysis. The view shows that there are many almost totally green pies. Which means, that they spent the wast majority of their processing time *in_evaluation*. There are small and bigger pies indicating shorter and longer processing time, and that there are PRs that although they skipped the initial analysis phase

have nonetheless spent time in analysis. The pies labeled a), b), and c) have a blue wedge in their pie chart. Selecting the PR labeled b) and displaying it in a state transition view, we can examine its life-cycle in more detail. Obviously the first resolution attempt failed and the PR was transitioned into *in_analysis*. From there it went back to *in_resolution* and was declared as resolved. Then the evaluation showed that there was still a problem and it went back to *in_resolution* and after this third resolution attempt and a second evaluation it was finally concluded.

Goal: Analyze the problem solving phases with respect to the priority of a PR To find out what impact the priority has on the duration of the process steps we use the view combination in Figure 7. Starting from the scatter plot view in the middle we select the highest and the lowest priority PRs and display them in our pie chart view. At first glance we see that the lower priority PRs have the biggest pies and therefore took the most time to resolve.

In Figure 7 we can spot pies with a relatively big yellow part, *e.g.*, the ones labeled a), b), c), and d), which means, that they stayed a long time in the submitted state. This is surprising since one would expect that on high priority problems work would start as soon as possible. It could be worthwhile to examine these reports more deeply and adjust the process to improve ramp up time.

Summarizing the results we found that: there is no clear tendency to accurately , under-, or overestimated PRs; most of the *analyzers* do not have a consistent pattern in their estimation quality; there are some analyzers with strong tendencies to either over- or underestimate the effort; there is one particular owner who often skips the initial analysis phase and that some of these PRs exhibit complications later on; there is a difference between the distributions of processing time for top and low priority PRs but there is no clear tendency considering the average open time.

4. Related Work

D'Ambros *et al.* used data from a release history database (RHDB) [3] to visualize the life-cycle of bugs [2]. With the System Radiography and the Bug Watch View, they provided a tool to focus on time, activity and severity/priority aspects of bugs.

Halverson *et al.* [4] devised a visualization that focuses on the social aspects of change requests and problem reports. Similar to our approach, they also visualize state changes and reveal problematic patterns such as multiple resolve/reopen cycles.

In [7] Weiss *et al.* predict the effort a problem requires to get fixed by analyzing the problem description and search for similar problem reports. The average effort it took these similar problems is than used as a prediction.

Anvik *et al.* describe the problems that arise when using open bug repositories, *e.g.*, duplicates, and irrelevant reports [1].

5. Conclusions

In this paper we presented an interactive visualization approach that is specifically tailored to uncover hidden patterns and exceptional entities in problem report data. It is based on the Micro/Macro Reading concept and uses easy to understand views. These views can be combined in a flexible way to allow sophisticated analyses.

We performed a first evaluation of our approach in an industrial setting with focus on improving effort estimation and resources planing. Discussions with the project managers of the industrial partner asserted the value of our approach and its potential for improving their software maintenance process.

Acknowledgments

This work was sponsored in part by the Eureka 2023 Programme under grants of the ITEA project if04032. Furthermore, this work was supported by the Swiss National Science Foundation (SNF).

References

[1] J. Anvik, L. Hiew, and G. C. Murphy. Coping with an open bug repository. In *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 35–39, New York, NY, USA, 2005. ACM.

[2] 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)*, pages 113–120. IEEE CS Press, June 2007.

[3] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of the International Conference on Software Maintenance*, pages 23–32, Amsterdam, Netherlands, September 2003. IEEE Computer Society Press.

[4] C. A. Halverson, J. B. Ellis, C. Danis, and W. A. Kellogg. Designing task visualizations to support the coordination of work in software development. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 39–48, New York, NY, USA, 2006. ACM.

[5] M. Lanza and S. Ducasse. Polymetric views — a lightweight visual approach to reverse engineering. *IEEE Transactions on Software Engineering*, 29(9):782–795, September 2003.

[6] E. R. Tufte. *Envisioning Information*. Graphics Press, May 1990.

[7] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 1, Washington, DC, USA, 2007. IEEE Computer Society.

TUD-SERG-2009-030
ISSN 1872-5392