Delft University of Technology
Software Engineering Research Group
Technical Report Series

Assessment of Issue Handling Efficiency

Bart Luijten, Joost Visser, Andy Zaidman

Report TUD-SERG-2010-004

TUD-SERG-2010-004

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: Paper accepted for publication in the proceedings of the 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)

Assessment of Issue Handling Efficiency

Bart Luijten
Delft University of Technology &
Software Improvement Group, The Netherlands
Email: b.j.h.luijten@student.tudelft.nl

Joost Visser
Software Improvement Group
The Netherlands
Email: j.visser@sig.eu

Andy Zaidman
Delft University of Technology
The Netherlands
Email: a.e.zaidman@tudelft.nl

***Abstract*—We mined the issue database of GNOME to assess how issues are handled. How many issues are submitted and resolved? Does the backlog grow or decrease? How fast are issues resolved? Does issue resolution speed increase or decrease over time? In which subproject are issues handled most efficiently? To answer such questions, we apply several visualization and quantification instruments to the raw issue data. In particular, we aggregate issues into four risk categories, based on their resolution time. These categories are the basis both for visualizing and ranking, which are used in concert for issue database exploration.**

***Keywords*-Defect resolution, Issue mining**

I. Introduction

Software development and maintenance are typically organized around issues (defects, feature requests, ...) that need to be resolved. These issues can be managed with an Issue Tracking System (ITS), which plays a central role in the communication between members of the development team and between developers and users of the system.

While not designed for mining purposes, issue trackers can be a valuable source of information. Through analysis of the historical data in the ITS, we are able to (1) gain insight into the issue handling process, (2) identify bottlenecks or problem areas and (3) find ways to optimize the process [1].

This leads us to our central research question for this paper: *How do developers and maintainers deal with issues?* More concretely, this paper addresses these questions:

RQ1 Can the efficiency of issue handling be assessed in an objective way?
RQ2 Are there significant fluctuations in bug solving efficiency throughout time?
RQ3 Are there significant differences in bug solving efficiency between software components or packages?

In order to answer these research questions, we create three views that offer alternative perspectives on the historical issue information. We also present an approach which allows software engineers to iteratively refine the analysis and which can be used to identify problematic areas (e.g., components in the system that suffer from many open issues) or bottlenecks in the issue handling process. Due to space restrictions, this paper focuses on a particular type of issue, namely bugs, but our methodology and views are equally suitable for studying other types of issues.

This paper is structured as follows. In Section II we describe our approach and tool for analyzing issue repositories. Section III describes the GNOME issue data, while the results and their interpretation are provided in Section IV. Related work is provided in Section V. We conclude the paper in Section VI.

II. Assessment approach and tools

We constructed a Java tool to capture information from an ITS. The tool includes a generic data model that stores the needed data from different issue trackers in a unified fashion. The data model is optimized for post-mortem queries on large batches of issues.

After loading the issue data into our tool, we are able to generate three different views that enable assessment of the issue handling process. A high-level overview is provided by the *Issue Churn View*. When a problem area has been identified, the *Issue Risk Profiles* can give a more quantitative view on these areas, while the *Issue Lifecycle View* can be used to zoom in on a particular (sub)component to get a detailed view on the lifecycle of issues.

A. Issue Churn View

The high-level Issue Churn View (ICV) shows the issue handling on a monthly basis (see Figure 1). The X-axis represents time and the positive and negative values on the Y-axis represent, respectively, the number of submitted and resolved issues. For the submitted issues we use (1) dark red to indicate that an issue was opened and solved in the same month, (2) light red for issues that were opened but not closed, (3) dark grey for recent backlog (issues open $\leq$ 6 months) and (4) light grey for long-term backlog (issues open $>$ 6 months). For the resolved issues, we again distinguish between issues both submitted and closed in this month (dark green) and older issues that have been solved in this month (light green).

Important facts that we can derive from the ICV are: the number of incoming and outgoing issues, visible through the red and green colors, and the development of backlog. An increase of the backlog may indicate that issue solving capacity is below what is needed or that submitted bug reports are of low quality and require too much effort to reproduce or track down.

ICVs are best constructed separately for defects and other issues, since their numbers and priorities are very different.

B. Issue Risk Profiles

To quantify the speed of issue resolution, we aggregate resolution times of individual issues into so-called *risk profiles*, which are subsequently mapped to ratings.

We define the resolution time as the time an issue is in an open state. Thus, we look at the time an issue is marked as being in the *new* or *assigned* state but not in the *closed* or *resolved* state. If an issue has been closed and then reopened, all open intervals count towards the issue resolution time, but the intervals in which the issue was closed do not.

Individual issue resolution times do not follow a normal distribution, but rather a power-law-like distribution. As a result, a simple aggregation by taking the *mean* or *median* of the issue resolution times is not appropriate. Instead we construct risk profiles by assigning items to risk categories based on their metric values. For defect resolution time, we use the following risk categories:

Category	Thresholds	
Low	[0, 28]	days (4 weeks)
Moderate	(28, 70]	days (10 weeks)
High	(70, 182]	days (6 months)
Very high	(182, ∞)	days

For example, a defect with a resolution time of 42 days falls into the *moderate* risk category. The thresholds between these risk categories were chosen to coincide roughly with the 70th, 80th, and 90th percentile of defect resolution times in a set of about 100 releases of various open source software products, because at these percentiles the variability between releases was observed to be higher than at lower percentiles [2].

Based on this risk assignment, a risk profile is constructed by calculating the percentage of items in each category. For example, $\langle 70, 19, 11, 0\rangle$ is the risk profile of a product history where 70% of all defects were resolved within 4 weeks, 89% were solved within 10 weeks, and none took longer than 6 months to solve.

Risk profiles can be constructed for the entire history of a product but also for sub-groups of the defects, such as releases or sub-products. When grouping issues by product version, we take the issues that are resolved between that version and the next. For grouping by sub-product, we exploit issue categorization tags in the ITS.

Risk profiles can be mapped to ratings to enable straight-forward comparison. We rate on a unitless scale between 0.5 and 5.5 that can be rounded to an integral number of *stars*. By benchmarking against the defect sets of the same 100 systems, we calculated the following mapping [2]:

Rating	Moderate	High	Very High
*****	8.3%	1.0%	0.0%
****	14%	11%	2.2%
***	35%	19%	13%
**	77%	23%	34%

For example, a snapshot with risk profile $\langle 70, 19, 11, 0\rangle$ will be eligible for a ranking of 3 stars. By interpolation our ranking algorithm establishes an exact rating of 3.25.

Risk profiles and ratings for issue resolution time are useful for comparing (slices of) the history of software products in a quantitative manner. As such, systems or components who perform relatively worse can be identified and action can be undertaken. Table I shows examples.

C. Issue Lifecycle View

The Issue Lifecycle View (ILV), which is loosely based on the Change History View by Zaidman et al. [3], is a scatter plot of issues versus modification dates. An example ILV can be seen in Figure 3. The X-axis represents time, while the Y-axis is populated by issues, sorted by the date they first appeared in the tracker. An issue is represented by a horizontal line fragment, which indicates that the issue is marked open. Blue dots on this line signify that a comment was placed on the issue and yellow dots point at other events, i.e., any change in a property for that issue, except for opening, closing or commenting (e.g., reassignment, attaching a patch or a change in priority).

A number of interesting facts can be derived from the ILV: the number of horizontal lines directly above a point in time (X-axis) shows the number of open issues at that point. The length of the lines indicates the time an issue has been open and thus also shows potential backlog. By studying the length of lines we also see the age composition of open issues, which in turn helps to understand how the backlog is being handled. For example, a system with a constant (non-addressed) backlog has a set of very long lifecycle lines, the backlog, and a set of short lines, issues that are actually being solved. Vice versa, when the backlog is addressed, the line lengths would be more uniform, since the older issues are solved quicker, but the younger ones slower.

III. Input data

As input data for the study reported in this paper we have used and ITS dump of the GNOME project[1]. This data represents the period between 1999-01-01 and 2008-09-19. A few issues from before this period are in the dataset, but we removed those from our analysis. In the period under analysis, 431838 issues were recorded. Issues marked as *duplicate* (143568), *invalid* (114611, including *notgnome* etc.) or *wontfix* (18093) were not taken into consideration. Using the *severity* field of each issue, we discovered that 22120 of the remaining issues are in fact *enhancements* and the rest are true *defects* (133446). Of these defects, 106932 are closed and 26514 are still open.

IV. Results

Figure 1 shows an ICV for the defects in the GNOME issue tracker. A number of observations can be made about

[1]http://msr.uwaterloo.ca/msr2009/challenge/gnome_data/gnome_bugzilla.xml.bz2

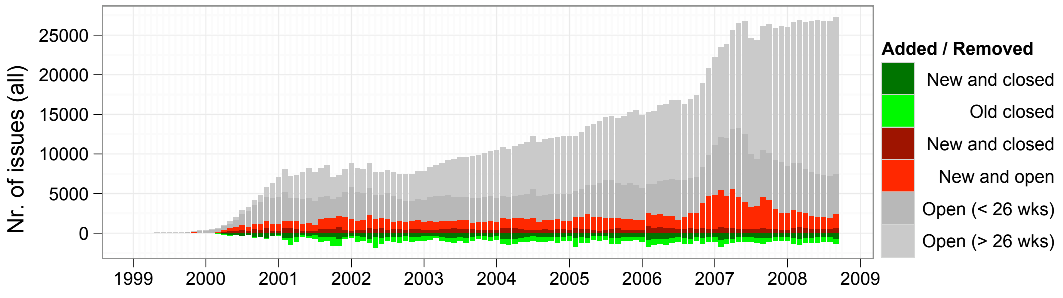


Figure 1. Issue Churn View for the issues of type *defect* of GNOME.

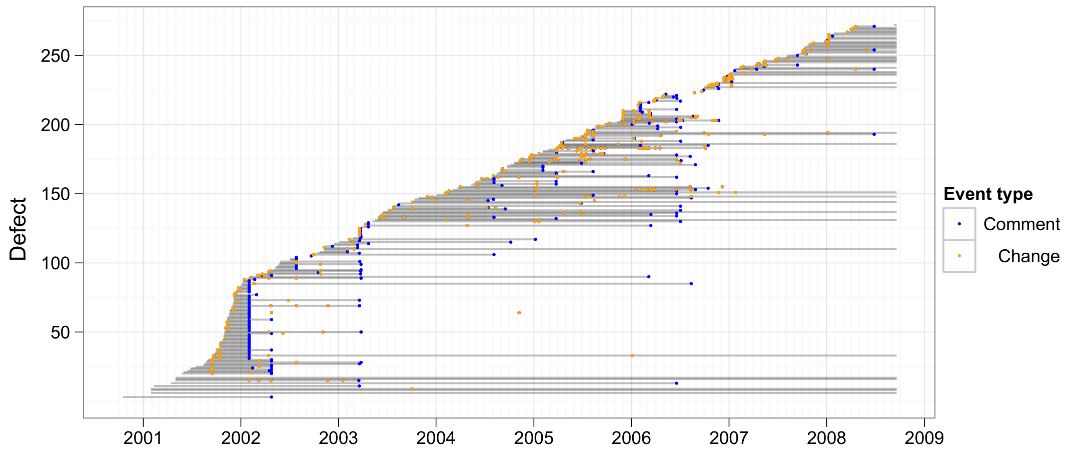


Figure 3. Issue Lifecycle View for defects in the GNOME-pilot subproject.

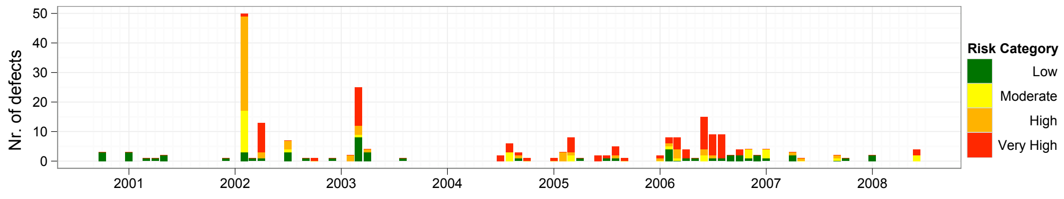


Figure 4. Quality profile evolution over time in the GNOME-pilot subproject.

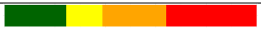
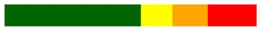
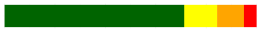
Product	Profile	Rating	
Gnome-pilot		*	(1.47)
Evolution		**	(2.18)
Beagle		***	(3.23)

Table I
EXAMPLE DEFECT RISK PROFILES.

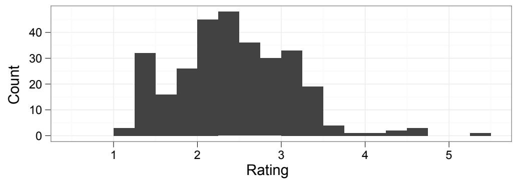


Figure 2. Histogram of defect resolution ratings for GNOME projects.

the GNOME defect solving history:

The use of the tracker picks up during the year 2000 and reaches a stable level from 2002 onwards.

From 2002 until November 2006, the number of incoming and outgoing defects per month is constant. The backlog is growing steadily, indicating that there are more defects being submitted than solved (i.e., a capacity deficit).

In November 2006, there is a sudden increase in defect submission activity, which lasts until the end of 2007 and then decreases again. In parallel, the backlog also strongly increases.

Because the backlog in the preceding graph seems very large, we are interested in knowing which parts of the GNOME project contain the longest to solve defects. We split up the dataset into separate products, using the *product name* recorded with each issue. We then assigned each product a *defect resolution rating*, based on the resolution times for the closed defects only.

Products that have less than 10 associated closed defects have been removed, because these contain too few defects to make the rating meaningful. Many of them have either the maximum or minimum possible rating, because all solved defects are in the highest or lowest risk category. The resulting dataset contains 300 products. Three example risk profiles are shown in Table I. For example, Evolution has the risk profile $\langle 54, 13, 14, 19\rangle$, giving it a rating of 2.18.

Figure 2 shows a histogram of all ratings for the 300 products. The ratings seem to be skewed to the left, indicating that in GNOME products it takes longer to solve defects than in the systems in our calibration set. Looking at the products in the lower end of this graph, we noticed a number of interesting candidates for further analysis. *Gnome-pilot*, *gnome-media* and *gnome-core* all have more than 200 defects and a rating below 1.6.

We picked *gnome-pilot*, a product for hand-held computers, for further investigation. It has 237 closed defects and rates 1.47, with over 35% of defects taking more than half a year to solve. To find out why this rating is so low, we use the ILV to investigate the defect lifecycle in more detail.

The ILV for *gnome-pilot*, Figure 3, shows us that there are many defects that are open for multiple years. The graph shows a number of clearly visible vertical stripes, where a large number of defects was solved simultaneously. Looking at the comments associated to these issues, we found that many (but not all) had been solved some time previously, but were left open in the issue tracker. At the dates where the stripes show, these were closed as a clean-up.

We suspected these large simultaneous actions to be the cause of the low overall rating. To investigate this, we constructed an issue resolution quality profile per month for *gnome-pilot*. The result is displayed in Figure 4. In this graph the height of a bar indicates how many defects were solved. The peaks coincide with the large actions in the ILV, and are primarily composed of (very) high risk defects. The large simultaneous actions do indeed seem to form the bulk of the high-risk defects. Nonetheless, since not all defects with long resolution times are closed as part of clean up actions, the rating of *gnome-pilot* remains low.

V. Related work

Kim and Whitehead investigated the bug-fixing time of each file in a software system. They found that the files with the longest bug-fixing time also contain the most bugs [4]. Similar to our study, they also try to identify problem areas.

Ihara et al. propose a method to analyze the bug modification process. They calculate the time required to transit between states in the bug modification process [5]. Similar to our own study, their aim is to identify bottlenecks in the bug modification process. One of the bottlenecks they have identified is the verification of resolved bugs, which was the most time-consuming step in the bug resolution process in both Apache and Firefox.

VI. Concluding remarks

In this paper we have presented three views and an approach that allow software engineers to retrospectively assess the issue handling process on the basis of recorded issue data. We have applied these to the GNOME project. We can now answer the research questions of Section I:

RQ1: Can the efficiency of issue handling be assessed in an objective way?: Yes. We can perform a high-level assessment of the issue handling process using the *Issue Churn View*. Using *Issue Risk Profiles* and ratings derived from them, we can zoom in and assess issue handling during particular periods and/or for particular system components. Finally, the *Issue Lifecycle View* allows detailed assessment on the level of individual issues.

RQ2: Are there significant fluctuations in bug solving efficiency throughout time?: Yes. In the case of GNOME the *Issue Churn View* revealed certain periods (e.g., November 2006 – end of 2007) with an increase in the number of defects reported and a subsequent sharp increase in the defect backlog. The monthly *Issue Risk Profiles* showed a simultaneous increase in high risk defects (resolution times over 6 months) and a drop in the defect resolution *rating*.

RQ3: Are there significant differences in bug solving efficiency between software components or packages?: Yes. The *Issue Risk Profiles* offer a quick way of comparing the defect solving efficiency, by assigning a rating to each component. In the case of GNOME we have seen that there is a large difference in the time it takes to solve defects between subprojects, with some packages where most defects are solved within 28 days, while for some other packages many defects take 182 days or more to solve.

Our case study has shown that a number of straightforward instruments for visualisation and quantification of issue handling can be used in concert to assess the efficiency of issue handling both at a high abstraction level and in detail.

In future work we expect to refine these instruments. In particular, we want to take into account the difficult to solve a bug (e.g., blocker bugs). Furthermore, we aim to apply our approach to commercial software projects.

References

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Coping with an open bug repository," in *Proc. 2005 OOPSLA workshop on Eclipse technology eXchange (eclipse'05)*. ACM, 2005, pp. 35–39.

[2] B. Luijten, "The influence of software maintainability on issue handling," Master's thesis, Delft University of Technology, 2010.

[3] A. Zaidman, B. Van Rompaey, S. Demeyer, and A. van Deursen, "Mining software repositories to study co-evolution of production and test code," in *Proceedings of the International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2008, pp. 220–229.

[4] S. Kim and E. J. Whitehead, Jr., "How long did it take to fix bugs?" in *Proc. Int'l workshop on Mining software repositories (MSR)*. ACM, 2006, pp. 173–174.

[5] A. Ihara, M. Ohira, and K.-i. Matsumoto, "An analysis method for improving a bug modification process in open source software development," in *Proc. joint int'l workshops on Principles of software evolution (IWPSE) and software evolution (Evol)*. ACM, 2009, pp. 135–144.

TUD-SERG-2010-004
ISSN 1872-5392