Practice Evolution Explorer

Jialiang Xie*, Minghui Zhou*, Audris Mockus†, Xiujuan Ma* and Hong Mei*
*School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies, Ministry of Education
Beijing 100871, China
{xiejl11@sei.,zhmh@,maxj07@sei.,meih@}pku.edu.cn
† Avaya Labs Research
233 Mt Airy Rd, Basking Ridge, NJ
audris@avaya.com

***Abstract*—Reporting and resolving issues is an essential part of software development. This is accomplished primarily by volunteers in OSS projects and by service providers in commercial projects. Project environment often changes, e.g., the number of users may increase, and, to be successful, the projects develop new practices to cope with changes. We want to understand how the issue resolution practices evolve over time, how efficient and effective they are, and how they can be improved. We use ubiquitous records in the issue tracking systems to discover practice evolution and to quantify their impact. We built Practice Evolution Explorer (Pe^2) tool to visualize and understand issue tracking data via linked views/selectors representing properties of issues and issue transitions between states. We illustrate how to detect inadequate practices and how to quantify the impact of project decisions on service quality and efficiency. We plan to apply Pe^2 in both commercial and open sources projects to improve the quality of responses to user-reported issues while minimizing the effort needed to achieve that improvement. In particular, we would like to investigate how the commercial projects could achieve the rapid response times observed in OSS.**

***Keywords*-Practice evolution; issue resolution time; service quality; issue quality**

I. Introduction

Responsiveness to user issues is essential to software project's success. In open source projects input from users is critical for improving product quality, because many issues are found, reported, and sometimes fixed by early adopters [1]. It is, therefore, of interest to understand how projects engage users and volunteers and how such practices evolve in response to external changes. To achieve that we ask two research questions:

- Can we discover issue reporting and resolution practices and their evolution?
- Can we quantify the impact of these practices on service quality and help projects choose most suitable practices?

To answer these questions we first performed a qualitative study to understand the nature of these practices and their importance to the projects. In Gnome and Mozilla bug triaging (a common way to refer to practices used to report and resolve issues) was of great importance and participants primarily used issue tracking system both, to understand what is going on and to change some key practices. "Processes that limit the size or potential of our community limit Mozilla. Conversely, making it easier for people to cooperate, collaborate, experiment and play enhances the community's capacity"[1], according to a volunteer group developing community management metrics and tools for Mozilla. It is, therefore, critical to understand practices employed by a project and to address their weaknesses. Otherwise valuable contributors might leave. For example, one long-time contributor left "because of a general lack of interest in doing anything substantial to improve the Triage process"[2].

Issue-tracking systems are widely used in software projects and they record the way tasks are assigned, problems are discussed, and issues are resolved. Such data contains a detailed history of the project and might provide a way to find out decisions that were problematic or practices that proved beneficial. To cope with complexity of issue tracking data we developed Practice Evolution Explorer[3] (Pe^2) to spot anomalies and to quantify relevant measures of service quality (delay and effort). It visualizes transitions between states, time trends, and issue attributes and, based on these views, lets user select subsets of interest.

Using Gnome project we illustrate how we used the tool to detect several dramatic changes in the issue tracking practices and how the tool could be used to rapidly detect the impact of new technology and to find effective solution.

We make two contributions: First, Pe^2 helps to detect anomalies (and thus, evolution) in issue resolution practices. Second, it may help to design better practices and to avoid costly mistakes by quantifying the potential implications for quality and effort.

[1] http://eaves.ca/2011/04/07/developing-community-management-metrics-and-tools-for-mozilla/

[2] http://tylerdowner.wordpress.com/2011/08/27/some-clarification-and-musings/

[3] http://www.youtube.com/watch?v=y9O37OTecbE, and http://passion-lab.org/pee.html

We illustrate Pe^2 with two scenarios of practice changes discovered in Gnome in Section II, and describe design considerations and other details in Section III and Section IV. The related work is presented in Section V. Future work and summary are in the last section.

II. Practice Evolution

We illustrate Pe^2 on a large Bugzilla repository of Gnome software eco-system. Gnome implements user interface functionality, and has more than 10 years of history and more than 600K issues.

We use term "issue quality" to designate the fraction of issues in the sample that were resolved as fixed. For example, a high proportion of invalid or duplicate issue reports would waste time of project participants who need to ascertain the validity of such issues.

We use term "service quality" to refer to the time until 90% of the issues are resolved (average time is not a robust measure because of the statistical distribution of resolution times). A shorter resolution time implies rapid response to user concerns, thus representing good service quality.

A. Issue states

An issue is created by an issue reporter and handled (triaged) by project participants. In the course of the triage and resolution issues transfer through a predefined set of states. For example, Gnome defines standard steps of triaging [4]. An issue is reported (born) in an UNCONFIRMED state. When a triager confirms it as a valid issue, its state is changed to NEW. Alternatively, if it is, for example, a duplicate, it may be immediately resolved and its state changed to RESOLVED. When the report does not contain sufficient information to reproduce and fix, the state would be changed to NEEDINFO. Issues in state NEW are to be assigned and need to be resolved. The assignee may accept the issue (state ASSIGNED), or pass it to someone else (issue remains in the state NEW), or resolve it (state RESOLVED). However, in practice the transition sequence varies, as the scenarios below show.

Each "RESOLVED" issue has a resolution, e.g., FIXED, DUPLICATE, INCOMPLETE, or INVALID.

B. NEW vs UNCONFIRMED

The first simple scenario depicted in the brief video[3] illustrates the adjustment to the policy of reporting issues. The new policy restricted the population of participants who can report issues directly in state "NEW" instead of state "UNCONFIRMED". Issues in state "NEW" are considered to be valid issues while the validity still needs to be established for issues in state "UNCONFIRMED".

With Pe^2 it is easy to detect this change by selecting the issues that start with state "NEW" with the transition filter (see Section IV-D). The timeline view shows the dramatic

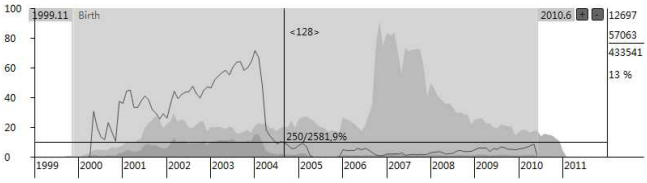

Figure 1. A change of "NEW" issues in the timeline view

a

Resolution	All	None
DUPLICATE	17%	1395
FIXED	65%	5203
INCOMPLETE	0%	63
INVALID	3%	262
NOTABUG	5%	465
NOTGNOME	1%	85
OBSOLETE	0%	0
WONTFIX	5%	427
WORKSFORME	0%	0
UNDEFINED	0%	0
Sum		7900/7901

b

Resolution	All	None
DUPLICATE	19%	2865
FIXED	60%	9065
INCOMPLETE	2%	373
INVALID	4%	740
NOTABUG	6%	965
NOTGNOME	2%	351
OBSOLETE	0%	14
WONTFIX	3%	572
WORKSFORME	0%	0
UNDEFINED	0%	0
Sum		14945/14946

c

Resolution	All	None
DUPLICATE	27%	811
FIXED	50%	1505
INCOMPLETE	3%	113
INVALID	8%	263
NOTABUG	6%	184
NOTGNOME	0%	4
OBSOLETE	1%	54
WONTFIX	2%	67
WORKSFORME	0%	0
UNDEFINED	0%	0
Sum		3001/3002

Figure 2. Quality of "NEW" issues in 2001, 2003, and 2004

rise from 40% of reported issues in state "NEW" in 2001 rising to 60% in 2003 before rapidly dropping to 10% after April of 2004 (see the black line in Figure 1) . Investigating what happened in 2001 (by selecting one year interval in the timeline view and observing the barchart of the distribution of resolutions) we found that 65% of these "NEW" issues were ultimately fixed, while in 2003 only 60% of them were fixed (see Figure 2a and 2b).

Clearly such drop suggests that the quality of "NEW" issues has gone down and that restricting the pool of participants with a privilege to report an issue in "NEW" state may improve the situation. The actions undertaken by the project lead to a much smaller fraction of "NEW" issues. However, the issue quality did not improve: only 50% of the issues reported as "NEW" were fixed in 2004 (Figure 2c) — an even smaller fraction than in 2003. Furthermore, the service quality also decreased: a calendar year prior to April 2004 it took 9 months to resolve 90% of issues while during the subsequent calendar year it took 9.7 months. It is, therefore, not clear if the intervention achieved its desired goals.

C. Usability of crash reporter

The second example of practice evolution in the video[3] is driven by the desire to let more Gnome users participate in issue reporting via crash reporting tool Bug-Buddy [5]. As Figure 3 shows, a dramatic peak with 11,600 new issues is visible in September, 2006, while during the prior month there were only 2,600 new issues. Of these September new issues, 82% were submitted via Bug-Buddy. While Bug-Buddy was introduced several years prior to that, the particular version 2.16 that became available in September has made it much easier for unsophisticated Gnome users to report issues. Earlier, users had to install and configure sendmail package or report an issue using Bugzilla web site. The innovation initially looked promising to project participants: "With the new Bug-Buddy, we're all receiving

[4]http://live.gnome.org/Bugsquad/TriageGuide

[5]When an application using the GNOME libraries crashes, Bug-Buddy generates a stack trace using gdb and invites the user to submit the report to the GNOME bugzilla.

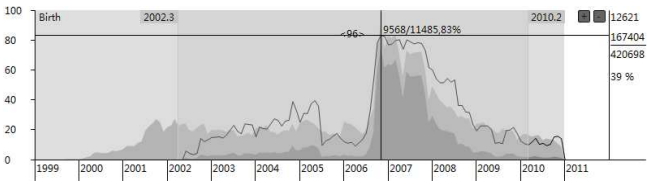

Figure 3. A peak of new-born issues in the timeline view

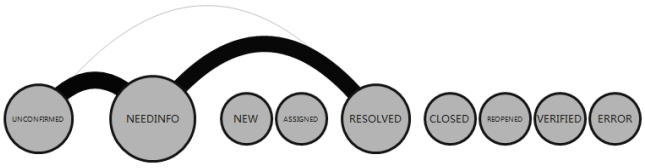

Figure 4. Workflow with NEEDINFO in the transition view

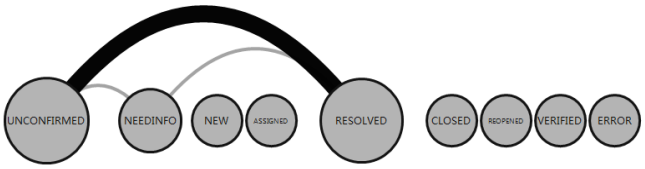

Figure 5. Workflow without NEEDINFO in the transition view

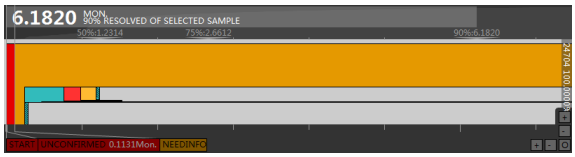

Figure 6. Issue resolution time in the process view

tons of new bugs. It's good, since we now know about some crashers we didn't know before."

The volume of new issue reports, however, was overwhelming and the quality was quite low: only 7% of the new issues had stack traces with debugging information. Simply having a stack trace is not as useful as having actual lines of code causing the crash. Users who could now easily report crashes, did not have enough motivation or skill to install debugging libraries which would provide debug symbols, thus improving the quality of the issue reports. Furthermore, 95% of the issues that needed additional information to be reproduced were closed with the resolution of INCOMPLETE because the reporters did not respond to requests for additional information. As one developer put it: "The NEEDINFO status is nearly killed by these incomplete reports."

To address these problems, the project introduced new technology and evolved practices. To address the issue of missing line numbers Gnome introduced Google Airbag tool in Bug-Buddy v2.19. Airbag annotates certain crash reports with compiler-provided debugging information. As a result, the fraction of invalid issues dropped down to 55% for Bug-buddy v2.19. From practice's perspective, Gnome community streamlined the transition UNCONFIRMED $\Longrightarrow$ NEEDINFO $\Longrightarrow$ RESOLVED (Figure 4) to UNCONFIRMED $\Longrightarrow$ RESOLVED (Figure 5) in May, 2007. Before the change, 90% reported issues were resolved within 6.18 months (as shown in Figure 6). The change resulted in an improvement of service quality by reducing the delay to 1.14 months.

III. Approach

Based on the qualitative study we designed Pe^2 to address two issues that vexed project participants the most: detecting anomalies in the issue resolution practices and quantifying the impact of specific practices.

To accomplish that Pe^2 visualizes and compares various properties of the subsets of issues that a user can interactively select using a variety of visual and textual (regular expressions) options. An overview of Pe^2 is given in Figure 7. The basic paradigm is that of linked views, where the same set (or sets) are displayed in a variety of ways to allow:

1) display of trends and corresponding anomalies,
2) visual selection of subsets of interest,
3) quantification and comparison of the issue and service quality for the selected subsets.

For example, a user can select one year before April, 2004 by brushing the mouse over relevant period in the timeline view. After saving the state (shown in the history panel at top-right), user can select one year after April, 2004. By toggling between these two saved states a user can clearly see what changed. In another scenario, a user may select issues that were resolved and then reopened using a simple regular expression "S*[UE]" where S is an abbreviation for resolved, U for unconfirmed, and E for NEW.

IV. Views and Selectors

Each view of Pe^2 is designed to present a particular set of anomalies or to quantify service and issue quality and also serve as an interactive filter that allows user to select the subsets of interest for comparison and to quantify issue and service quality for these subsets.

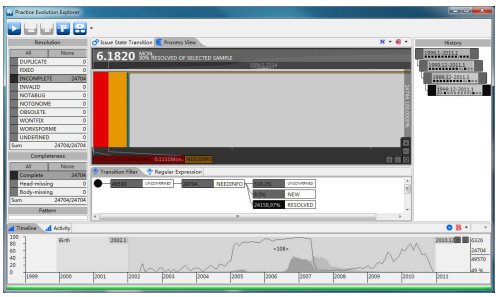

Figure 7. Overview of Pe^2

A. The timeline view

The timeline view shows trends and serves as date filter. It is represented by an area chart with date on the horizonal axis and chosen statistics on the vertical axis. Statistics include Birth Rate (the number of issues reported during one month), Expiration Rate (the number of issues resolved during the month), and Cumulative Issues (open, but not yet resolved). The timeline view shows two subsets of the selected issue population. The part shown in darker color represents the entire selection while the lighter color shows one part of the selection, for example, issues that are reported as "NEW". In addition, the fraction of issues representing the lighter color is drawn as a black line.

B. The transition view

The transition view shows frequencies and delays between the states the issues pass through. Circles show states and arcs transitions, with the thickness of the arc indicating one of the following statistics for the selected set: the number of issues having that transitions, the total delay incurred for that transition, and the average delay incurred for that transition. The arcs above the circles go left to right while the ones below circles go right to left.

As noted above, the transition view is linked with the timeline (and other) view(s). In particular, moving the time range in the timeline view shows the animation of the evolution of the transitions among states.

C. The process view

The process view is designed to quantify service quality. It provides details of the delay for each transition. The horizontal axis shows delay and the vertical axis shows the numbers of issues. Each state is drawn in single color with the width representing average time and the high the number of issues. The area of each state shows the total time spent transiting between two states in all selected issues. Time zero represents the time an issue is created and the time at which next colored region starts indicates delay between the time the issue was created and the next state.

D. The selectors

The transition filter provides visual and textual methods to select subsets of issues that went through chosen state transitions.

The resolution and completeness filters (shown in the top-left of Figure 7) display the number (and fraction) of issues with each resolution and level of completeness in the the current subset. A user may also expand or narrow the current subset of issues by adding (removing) resolutions or levels of completeness to (from) the current subset.

V. Related Work

There has been substantial amount of work on developing tools to investigate software repositories in order to improve software production. For example, Expertise Browser [2] presents the relationships between developers and source code they change to help determine experts for the code and develop expertise profiles, Hipikat [3] provides access to the group memory that is implicitly formed by all of the artifacts produced during the development for a software project, and Ariadne [4] as a plug-in for Eclipse offers developers visualizations of the source code authorship and the potential presence of coordination problems. More recently, Codebook [5] tries to discover transitive relationships between people, code, bugs, test cases, specifications, and related artifacts by mining all kinds of software repositories, and Lungu et al. [6] visualize the evolution of software ecosystems.

However, the evolution of project practices through issue tracking have been neither investigated, nor quantified. In this study, we visualize the anomalies of issue tracking practices and quantify the relevant effects. We hope to help developers understand the impact of their practices and to design practices.

VI. Conclusion

We introduced a visualization tool to detect anomalies in the issue resolution process and to quantify the service quality and issue quality. Pe^2makes it easy to quantify the impact of various practices and, thus, may help reduce time wasted by developers on invalid issues and improve service quality by responding to user issues more rapidly.

We are working on applying Pe^2in both commercial and open source projects to help discover and remove inefficiencies in issue resolution practices.

References

[1] A. Mockus, R. T. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 1–38, July 2002.

[2] A. Mockus and J. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *ICSE 2002*. Orlando, Florida: ACM Press, May 19-25 2002, pp. 503–512.

[3] D. Cubranic and G. Murphy, "Hipikat: A project memory for software development," *TSE*, vol. 31, no. 6, 2005.

[4] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles, "Supporting collaborative software development through the visualization of socio-technical dependencies," in *GROUP 2007*. New York, NY, USA: ACM, 2007, pp. 147–156.

[5] A. Begel, K. Y. Phang, and T. Zimmermann, "Codebook: Discovering and exploiting relationships in software repositories," in *ICSE 2010*. New York, NY, USA: ACM, 2010, pp. 125–134.

[6] M. Lungu, M. Lanza, T. Gîrba, and R. Robbes. "The Small Project Observatory: Visualizing Software Ecosystems." in *Science of Computer Programming*, Elsevier 75(4) p. 264275, April 2010.