Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair

Gabriel Ripoche [1,2]
gripoche@uiuc.edu

Les Gasser [1]
gasser@uiuc.edu

[1] Graduate School of Library and Information Science
University of Illinois at Urbana-Champaign
501 E. Daniel Street, Champaign, IL 61820, USA
Phone: +1 217 265 5021 / Fax: +1 217 244 3302

[2] LIMSI-CNRS / Université Paris XI
BP 133, 91403 Orsay Cedex, France
Phone: +33 1 69 85 81 01 / Fax: +33 1 69 85 80 88

Abstract

In this paper, we present the method we have developed to analyze socio-technical aspects of software problem management in F/OSS communities, based on large corpora of problem reports; and we report on early results we obtained using our framework. Given the amount of data available, computational techniques to scalably extract event data and model the collectives' behaviors are needed. We are using a variety of techniques that couple human-based qualitative analysis with computational extraction and modeling to generate models of the processes involved in software problem management.

Keywords

Process inference and modeling; corpus-based language processing; language-based pattern extraction; software problem management; distributed collective practices.

1 Introduction

The work of identifying and resolving errors has always played a large part in software production and maintenance. Buggy software is in the "best" case annoying to users, and in the worst case presents major threats to security, economic health, and even lives. As complex software artifacts proliferate and become more central to—even ubiquitous in—peoples' lives, the social cost of software errors and bugs may increase. Also, in the emergent paradigm of continuous design incarnated by the Free/Open Source Software (F/OSS) model, bug management plays a major role in the design of the software itself and in its evolution [6]. As such, bugs provide a window on the more general "organizational" puzzle of F/OSS.

Since errors and bugs reduce the effectiveness with which software systems are built, and as they play an increasingly important role in current software development processes, we are interested in understanding how awareness of bugs comes about, how bugs can best be managed, and how people who build and use advanced software systems can organize their work to prevent, overcome, accommodate, and even benefit from (e.g. learn from) problems.

Most accounts of software problems focus on flaws in technical design and usability. Surely better design, prototyping, and needs analyses can help, but there is clearly much more to the issue. Specifically, the reliability of a software artifact is related to the structure of the technical and organizational processes that produce it and to the technical and organizational infrastructures and constraints under which it is built and maintained [2]. Our research is probing these infrastructures and constraints by examining general issues such as:

– How do large software communities continuously (re)design and (re)develop software over long periods of time?
– How do they manage continuous streams of software errors and problems?
– How do they capture, represent, and use collective knowledge for software maintenance?

Because most interactions in F/OSS projects occur over the Internet (newsgroups, IRC, repositories, etc.), very large amounts of time- and project-specific data on the functioning of F/OSS projects and communities are readily available online. We are using these data corpuses as an empirical foundation to address the following issues that arise in large F/OSS communities and, to a certain extent, in all large-scale software development efforts:

– How do people collectively "translate" from bug reports to bugs to resolutions?
– With large numbers of simultaneously open problems, how does the community decide what to do next and how to do it?
– Why do some bugs persist for long periods of time while others get resolved quickly?

2 Data Used

With hundreds of thousands of problem reports from a variety of Free/Open Source projects, widely accessible bug repositories provide an extremely large and diverse dataset for analyzing issues like those above. In many repositories, data has been captured over several years, allowing for the analysis of processes and their evolution over time. Our current analyses focus on data from Bugzilla [11], the repository of the Mozilla development project [10].

The Mozilla project started in 1998—after Netscape released Communicator's source code—with the aim of developing an open-source web browser, designed for standards compliance, performance and portability. Over its five years of development, several thousands of programmers have contributed code, tens of thousands of users have given feedback, and millions of copies have been downloaded.

Our snapshot of the Bugzilla database, captured in March 2002, contains over 128,000 bug reports and over 1.2 million comments representing feedback from more than 45,000 registered users.

3 Method

Several other researchers have studied issues such as duration of problems in large software efforts using factor analyses of process data [8]. However, more thorough accounts are needed and these will depend on more detailed conceptualizations of bug-repair processes, which must be developed using qualitative coding and analysis. Given the amount of data in Bugzilla, these qualitative studies must go beyond conventional human-based methods, and employ computational techniques to scalably extract event data and model the collective's behaviors. To extract details of events and processes, we are using a variety of techniques that couple human-based qualitative analysis with computational discovery and analysis [4,5].

3.1 Process Modeling

Cook and Wolf [1] presented a well-known process modeling framework (see Figure 1) which we have built upon and extended. We have developed analysis tools that use these data to create generalized probabilistic network models in the form of Markov models and probabilistic finite state automata (FSA) [1].

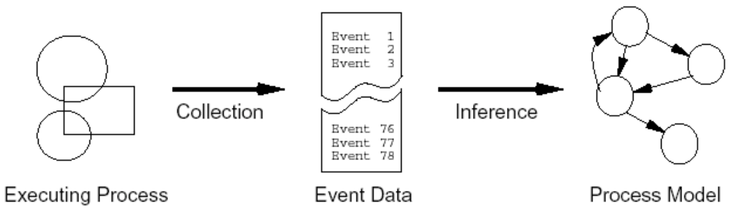

Figure 1 - Cook and Wolf's method of inferring models from executing processes.

For example, using this technique, we have modeled one type of formal process status change found in the Bugzilla repository. To do this, we automatically extracted the individual process trajectories of over 88,000 resolved problem reports, and coded them using sequences of letters (e.g. "R" for a status of "Resolved"). We also computed the frequencies of occurrence for each distinct trajectory (Figure 2 contains a sample of the "status strings" along with their frequency). Next, we combined these data in a probabilistic FSA, and obtained a model of the life-cycle of a bug report (see Figure 3). Currently, we are moving beyond the first-order model of Figure 3 to nth-order Markov models, to overcome the limitations of first order models (no effects of "history" in the processes).

```
# freq / process
 11037   ZZNZ-NZRD-RDVD
 10847   ZZUZ-UZRD-RDVD
 10510   ZZNZ-NZAZ-AZRF-RFVF
         [...]
    16   ZZNZ-NZAZ-AZNZ-NZAZ-AZRF-RFOF-OFOZ-OZRF-RFVF
         [...]
     2   ZZNZ-NZRF-RFOF-OFRF-RFOF-OFAF-AFRF-RFOF-OFOZ-OZRF-RFVF
         [...]
```

Figure 2 - Sample of input streams to our probabilitic models (here: bug report status changes).

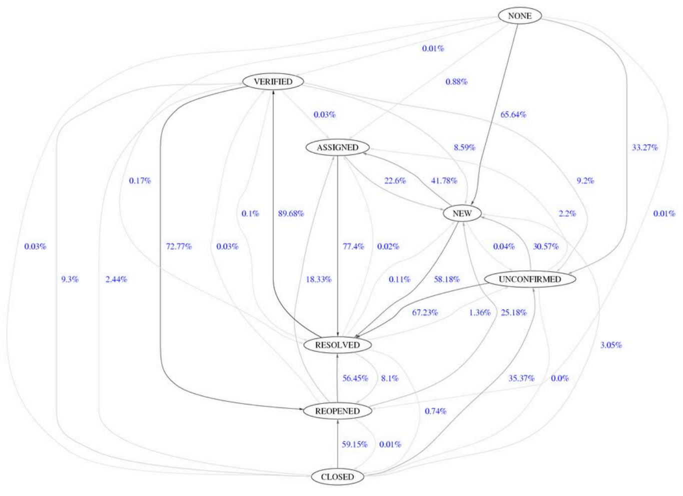

Figure 3 - 'Status'-only process model.

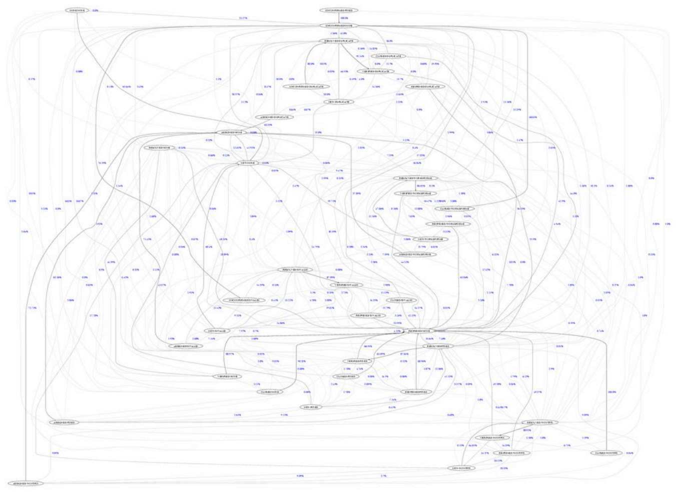

Figure 4 - A more complex process model (mapping two dimensions: 'Status' + 'Resolution Type').

The structure of these probabilistic models gives us a strong statistical and computational foundation for understanding bug repair processes, and for linking bug repair process decisions to process structures and outcomes. However, in the model of Figure 3, we used only one type, or "dimension", of process data—in this case, formal status changes. In most cases, such a simple model will not suffice to explain or predict process trajectories, so other aspects of (viewpoints on) processes will be needed.

3.2 Explaining Probabilistic Choices

Taking the risk of oversimplifying, let us consider the activity of managing bugs as a (likely very complex) type of heuristic action. Each process at play is composed of a sequence of steps, as illustrated in Figure 2 and Figure 3. Each state of a process has some history (trajectories in), and some set of future states (trajectories out) which are represented in the data. Figure 5 illustrates this concept. The question then becomes: what biases exist in any specific process enactment that will cause a specific output transition to be taken when the process is in a given state?

To reveal these biases, it is necessary to exploit many more types or "dimensions" of information than simply the formal report status used above. For example, Figure 4 shows the process model that results from combining an additional dimension, "Resolution Type", which is also formally represented in the Bugzilla repository. This new model has many more states and much finer distinctions are made in state-to-state transitions. Other potentially explanatory dimensions we are currently examining (which we hope will also add detail and increase discrimination) include: elapsed time, bug type, existence/absence of certain basic social processes (see below), and presence/absence of specific information types.

The previous point can be illustrated as follows: if for example a process reaches a formal "Resolved" state, we want to know whether the presence of certain types of basic social processes or information are reliably associated with transitions to a "Reopened" state, and hence may eventually provide predictions or explanations for generic process types.

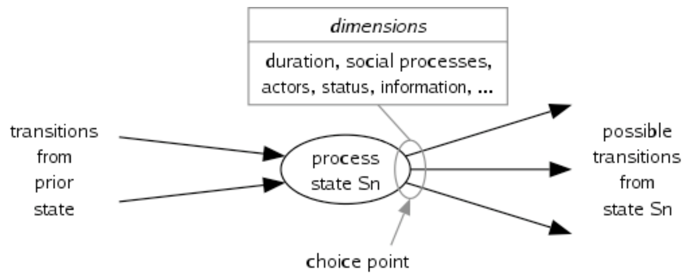
dimensions
duration, social processes, actors, status, information, ...
transitions from prior state
process state Sn
possible transitions from state Sn
choice point

Figure 5 - Choice point decision.

3.3 Computational Amplification

Unfortunately, many of the dimensions of interest are not represented and coded directly in the corpus, like "Status" and "Resolution Type" are. Instead, many of the interactions taking place in Bugzilla—and more generally in F/OSS communities—are represented in the form of natural language textual exchanges (what we will call informal data). These exchanges capture key data on many of the dimensions of interest, but only implicitly through language. In order to access information from these sources, we are developing language-processing tools to automatically annotate and extract relevant data.

Currently, we are implementing techniques based on semi-supervised generation of extraction patterns [7,12,13,14,15] and on discourse analysis [9] that allow us to identify specific instances of process events in many dimensions including information needed/used and basic social processes. For example, the top-level of our taxonomy of BSPs—elaborated using traditional ethnographic methods and human qualitative analysis—is shown in Figure 6.

Ambiguity	Description	Rationale
Articulation	Evaluation	Statement
Conflict	Negotiation	Uncertainty

Figure 6 - First level of our taxonomy of "basic social processes".

Using such taxonomy and a series of simple linguistic signatures gathered from a manually annotated sample of bug reports, we have been able to automatically extract and code candidate episode sets from the entire data corpus (or from specific targeted sub-samples). The automatically generated results—machine-proposed instances of BSPs in the data—were then analyzed for accuracy by the annotators, and the extraction patterns were selected and/or modified accordingly. Multiple iterations of this process allow for the selection of highly discriminative extraction patterns.

Our current human evaluations of extraction accuracy for the “conflict” and “evaluation” BSPs are on the order of 80% recall[1]. These tools have successfully enabled us to automatically filter large bodies of data for candidate episodes of a number of concepts, and to develop statistical distributions and models of social processes.

3.4 Looking at Bug Persistence

We have successfully applied this framework to the question of bug persistence and obtained promising preliminary results [3]. Prior work concerning the management of software errors showed that misalignment in distributed activity networks increased the difficulty of resolution [2]. In the context of Bugzilla, one way to grasp this insight is to consider that misalignments will lead to longer resolution time.

We coded an initial comparative sample of 159 bug reports containing 59 “long duration” reports (with a time to resolution above 1000 days) and 100 “short duration” reports (with a time to resolution of 30 days). A human-coded qualitative analysis of this sample indicated that long duration reports tended to contain more episodes of conflict, ambiguity and uncertainty. This led us to formulate the hypothesis that the duration of a bug depends (at least partly) on “time to consensus” on several key aspects of the nature of a bug, or in other words, on the time to alignment of these various aspects.

Of course, other constraints such as allocation of resources or priority of a bug have an impact on the time to resolution. However, this does not contradict our hypothesis, and our model allows for the future addition of these dimensions in order to further refine our comprehension of this process.

4 Future Work

In the previous section we gave a broad overview of our method for large-scale analysis of problem management. Our current research focuses on improving the tools used in the various steps of this method and on applying them to a broader number of dimensions and contexts.

One of the issues we briefly mentioned above concerns the need for annotated corpora in order to bootstrap our extraction mechanisms. While it is not possible to completely bypass this requirement, we are working on more autonomous extraction mechanisms, largely borrowing from the current trends in Information Extraction [14,15,12].

We are also developing better process models in order to account for more parameters and increase the descriptive power of the models. For example, current models based on finite state machines do not take history of a process into account. Recent experiments include the use of nth-order Markov models [1], and we are investigating other stochastic methods as well.

Because Bugzilla is such a central coordination tool in the Mozilla community, the data captured in the repository accounts for a large part of the activity related to problem management. However, future research includes looking at other archives of project activity (such as mailing-lists, project minutes, and CVS check-ins) both to complete our view of problem management in Mozilla and to broaden our investigations to the larger issue of continuous design in F/OSS communities.

Finally, while most of this paper focused on scalable, automated techniques, we are not ignoring the necessity of conducting human-based qualitative studies on samples of the data in order to precisely characterize the dimensions we integrate in our models. More such in-depth studies will be needed to improve our analyses.

5 Conclusions

We have developed a generic model for process-based explanation and have applied it to analyze the question of bug persistence. The scalability, the various techniques for event data collection and the modeling of multiple dimensions make our method well adapted to the complex analyses of large distributed software processes. We are currently working on refining the various extraction mechanisms and developing better process models in order to address in more detail the socio-technical mechanisms of F/OSS communities.

Acknowledgements

We gratefully acknowledge the contributions made by Jean-Paul Sansonnet and Bob Sandusky. Bryan Penne contributed to the data coding efforts. This material is based upon work supported by the National Science Foundation under Grant No. 0205346. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

[1] Because manual annotation of sample corpora is such a time consuming effort, we have not yet coded another test corpora that would allow us to measure the number of “misses” using our extraction patterns.

References

[1] Cook, J. E., Wolf, A., Automating Process Discovery through Event Data Analysis. In Proceedings of the 17th International Conference on Software Engineering, 1995.

[2] Gasser, L., The Social Organization of Errors in Computing Work. Working Paper LG-2000-13, Graduate School of Library and Information Science, University of Illinois, 2000.

[3] Gasser, L., Dubin, D., Penne, B., Ripoche, G., Sandusky, R., A Comparative Study of Resolution Times and Processes for Mozilla/Bugzilla Bug Reports. SQA Project Memo UIUC-2003-06. 22 February 2003.

[4] Gasser, L., Penne, B., Ripoche, G., and Sandusky, R., Computational Amplification of Qualitative Analysis: Mining Social Processes with Language Models. SQA Project Memo UIUC-2003-14. 25 April 2003.

[5] Gasser, L., Ripoche G., Building and Analyzing PFSAs as Process Representations, SQA Project Memo UIUC-2003-17. 20 May 2003.

[6] Gasser, L., Scacchi, W., Penne, B., Ripoche, G., Understanding Continuous Design in F/OSS. To appear in Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications (ICSSEA-03). 2003.

[7] Grishman, R., Adaptive Information Extraction and Sublanguage Analysis. In Proceedings of the Workshop on Adaptive Text Extraction and Mining at the 17th International Joint Conference on Artificial Intelligence. 2001.

[8] Herbsleb, J. D., Mockus, A., An Empirical Study of Speed and Communication in Globally-Distributed Software Development. IEEE Transactions on Software Engineering, 29:6. June, 2003.

[9] Herring, S., Computer-Mediated Discourse Analysis: An Approach to Researching Online Behavior. In Designing for Virtual Communities in the Service of Learning, Barab S. A., Kling R., and Gray J. H. (Eds.). New York: Cambridge University Press. 2003.

[10] The Mozilla Project website. `http://www.mozilla.org/` (last visited: 10/13/2003).

[11] The Mozilla Project's Bugzilla repository. `http://bugzilla.mozilla.org/` (last visited: 10/13/2003).

[12] Poibeau, T., Dutoit, D., Generating Extraction Patterns from a Large Semantic Network and an Untagged Corpus. SemaNet Workshop at COLING'02. 2002

[13] Riloff, E., Automatically Generating Extraction Patterns from Untagged Text. In Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96), 1996.

[14] Riloff, E., Jones, R., Learning Dictionaries for Information Extraction by Multi-Level Bootstrapping. In proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), 1999.

[15] Yangarber, R., Grishman, R., Tapanainen, P., Huttunen, S., Unsupervised Discovery of Scenario-Level Patterns for Information Extraction, In Proceedings of the 6th Conference on Applied Natural Language Processing, 2000.