Discovering, Modeling, and Reenacting Open Source Software Development Processes

Chris Jensen and Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

Abstract
Process discovery has been shown to be a challenging problem offering limited results. This paper describes a new approach to process discovery that examines the Internet information spaces of open source software development (OSSD) projects. In particular, we examine challenges, strengths, weaknesses and findings when seeking to discover, model, and re-enact processes associated with a large, global OSSD project like NetBeans.org. The goal of this approach is to determine the requirements and design of more fully automated process discovery and modeling mechanisms that can be applied to Web-based, open source software development projects.

Keywords: Process Discovery, Process Modeling and Simulation, Open Source Software Development

1. Introduction

The goal of our work is to develop new techniques for discovering, modeling, analyzing, and simulating software development processes based on information, artifacts, events, and contexts that can be observed through public information sources on the Web. Our problem domain examines processes in large, globally dispersed open source software development projects, such as those associated with the Apache Web server, Mozilla Web browser [Mockus, Fielding and Herbsleb 2002], and integrated development environments like NetBeans [2003] and Eclipse [2003]. The challenge we face is similar to what prospective developers or corporate sponsors who want to join a given OSSD project face, and thus our efforts should yield practical results.

OSSD projects do not typically employ or provide explicit process models, prescriptions, or schemes other than what may be implicit in the use of certain OSSD tools for version control and source code compilation. In contrast, we seek to demonstrate the feasibility of automating the discovery of software process workflows via manual search and analysis methods in projects like NetBeans by analyzing the content, structure, update and usage patterns of their Web information spaces. These spaces include process enactment information such as informal task prescriptions, community and information structure and work roles, project and product development histories, electronic messages and communications patterns among project participants [Elliott and Scacchi 2004, Scacchi 2002, Viller and Sommercille 2000]. Likewise, corresponding events that denote updates to these sources are also publicly accessible. Though such ethnographic discovery approaches net a wealth of information with which to model, simulate, and analyze OSSD processes, they are limited by a lack of scalability when applied to the study of multiple OSSD development projects. Subsequently, it suggests the need for a more automated approach to that can facilitate process discovery.

In our approach, we identify the kinds of OSSD artifacts (e.g. source code files, messages posted on public discussion forums, Web pages, etc.), artifact update events (e.g. version release announcements, Web page updates, message postings, etc.), and work contexts (e.g. roadmap for software version releases, Web site architecture, communications systems used for email, forums, instant messaging, etc.) that can be detected, observed, or extracted across the Web. Though such an approach clearly cannot observe the entire range of software development processes underway in an OSSD project (nor do we seek to observe or collect data on private communications), it does draw attention to what can be publicly observed, modeled, or re-enacted at a distance.

Our approach relies on use of a process meta-model to provide a reference framework that associates these data with software processes and process models [Mi and Scacchi 1996]. As such, we have been investigating what kinds of processing capabilities and tools can be applied to support the automated discovery and modeling of selected

software processes (e.g., for daily software build and periodic release) that are common among many OSSD projects. The capabilities and tools include those for Internet-based event notification, Web-based data mining and knowledge discovery, and previous results from process discovery studies. However, in this study, we focus on identifying the foundations for discovering, modeling, and re-enacting OSSD processes that can be found in a large, global OSSD project using a variety of techniques and tools.

2. Related Work

Event notification systems have been used in many contexts, including process discovery and analysis [Cook and Wolf 1998, Wolf and Rosenblum 1993]. However, of the systems promising automated event notification, many require process performers to obtain, install, and use event monitoring applications on their own machines to detect when events occur. While yielding mildly fruitful results, this approach is undesirable for several reasons, including the need to install and integrate remote data collection mechanisms with local software development tools.

Prior work in process event notification has also been focused on information collected from command shell histories, applying inference techniques to construct process model fragments from event patterns [Garg and Bhansali 1992]. They advise that rather than seeking to discover the entire development process, to instead focus on creating partial process specifications that may overlap with one another. This also reflects variability in software process enactment across iterations. This imparts additional inconvenience on the user and relies on her/his willingness to use the particular tools that monitor and analyze command shell events. By doing so, the number of process performers for whom data is collected may be reduced well below the number of participants in the project due to privacy concerns and the hassles of becoming involved. While closed source software engineering organizations may mediate this challenge by leveraging company policies, OSSD projects lack the ability to enforce or the interest to adopt such event-capture technology.

Cook and Wolf [1998] utilize algorithmic and statistical inference techniques to model processes where the goal was to create a single, monolithic finite state machine (FSM) representation of the process. However, it is not entirely clear that a single FSM is appropriate for modeling complex processes. Similarly, other FSM-related process representation schemes such as Petri-Net based FUNSOFT [Emmerich and Gruhn 1991] offered a wide variety of activity and state-chart diagrams. It appears however that these representations may lack scalability when applied to a process situated within a global organizational context involving multiple tools, diverse artifact types, and multiple development roles across multiple networked sites of reasonable complexity.

Last, while process research has yielded many alternative views of software process models, none has yet been proven decisive or clearly superior. Nonetheless, contemporary research in software process technology, such as Lil Jil [Cass, *et al*., 2000, Osterweil 2003] and PML [Noll and Scacchi 2001] argues for analytical, visual, navigational and enactable representations of software processes. Subsequently, we find it fruitful to convey our findings about software processes, and the contexts in which they occur, using a mix of both informal and formal representations of these kind. Thus, we employ this practice here.

3. Problem Domain

We are interested in discovering, modeling, simulating, and enacting software development processes in large, Web-based OSSD projects. Such projects are often globally distributed efforts sometimes involving hundreds or thousands of developers collaborating on products constituting thousands to millions of source lines of code without meeting face-to-face, and often without performing modern methods for software engineering [Scacchi 2002]. Past approaches have shown process discovery to be difficult, yielding limited results. However, the discovery methods we use are not random probes in the dark. Instead, we capitalize on contextual aids offered by the domain. Some of these include:

- Web pages, including project status reports and task assignments
- Asynchronous communications among project participants posted in threaded email discussion lists
- Transcripts of synchronous communication via Internet chat
- Software problem/bug and issue reports
- Testing scripts and results
- Community newsletters
- Web accessible software product source code directories

- Software system builds (executable binaries) and distribution packages
- OSS development tools in use in an OSSD project
- OSS development resources, including other software development artifacts

Each OSSD project has locally established methods of interaction and communication, whether explicit or implicit [Scacchi 2002, Scacchi 2004]. These collaboration modes yield a high amount of empirically observable process evidence, as well as a large degree of unrelated data. However, information spaces are also dynamic. New artifacts are added, while existing ones are updated, removed, renamed and relocated, else left to become outdated. Artifact or object contents change, and project Web sites get restructured. In order to capture the history of process evolution, these changes need to be made persistent and shared with new OSSD project members. While code repositories and project email discussion archives have achieved widespread use, it is less common for other artifacts, such as instant messaging and chat transcripts, to be archived in a publicly available venue. Nonetheless, when discovering a process in progress, changes can de detected through comparison of artifacts at different time slices during the development lifecycle. At times, the detail of the changes is beneficial, and at other times, simply knowing what has changed and when is all that is important to determining the order (or control flow sequence) of process events or activity. To be successful, tools for automated process discovery must be able to efficiently access, collect, and analyze the data including areas of the project Web space such as public email/mailing list message boards, Web page updates, notifications of software builds/releases, and software bug archives in terms of changes to the OSS information space [Scacchi 2002, Scacchi 2004].

How the project organizes its information space may indicate what types of artifacts they generate. For example, a public file directory named "x-test-results" can be examined to determine whether there is evidence that some sort of testing (including reference to test cases and test results) has been conducted, whereas timestamps associates with updates provide a sense of recent activity and information sharing. Similarly, when new branches in the Web site are added, we may be able to detect changes in the process or discover previously unknown activities. The types of artifacts available on the site may also provide insight into the project development process. Further investigation may excavate a file named "qa-functional-full" under the "x-test-results" directory, indicating that that functional testing has been performed on the entire system. Likewise, given an image file and its name or location within the site structure, we may be able to determine that an image named "roadmap2003" may show the progression that the project has made through the year of 2003 and future development milestones. This process "footprint" tells us that the some informal planning has been done. In some cases, artifacts containing explicit process fragments have been discovered, which may then be validated against the discovered process to determine whether the project is enacting the process as described. Whereas structure and content can tell us what types of activities have been performed, monitoring interaction patterns can tell us how often they are performed and what activities the project views as more essential to development and which are peripheral.

To prove the viability of our process discovery approach, we demonstrate it with a case study. For this task, we examine a selected process in the NetBeans [2003] project, which is developing an open source IDE using Java technology. The "requirements and release" process was chosen for study because it's activities have short duration, are frequently enacted, and have a propensity for available evidence that could be extracted using automated technologies. The process was discovered, modeled informally and formally, then prototyped for analysis and reenactment. The full results of our case study may be found in [Oza, *et al*. 2002]. The discussion of our process discovery and modeling methods and results follows next.

5. Process Discovery and Modeling

The discovery of processes within a specific OSSD project begins with a cursory examination of the project Web space in order to ascertain what types of information are available and where that information might be located within the project's Web site. The information gathered here is used to configure an OSSD process reference framework [Jensen and Scacchi 2003]. This framework provides a mapping between the tool, resource, activity, and role names discovered in the community Web with a classification scheme of known tools, resources, activities, and roles used in open source communities. This step is essential to permit association of terms such as "CVS" with source versioning systems, which have certain implications in the context of development processes. The project site map provided not only a breakdown of project Web pages within each section, but also a timestamp

of the latest update. This timestamp provides empirical evidence gathered from project content that reflects the process as it is currently enacted, rather than what it has evolved from.

Unlike Cook and Wolf's approach, we apply *a priori* knowledge of software development to discovering processes. Instead of randomly probing the information space, we use the reference model to help identify possible indicators that a given activity has occurred. To situate the process within its organizational context, we began by looking for evidence of the project's evolution. Guided by our framework, the project history was found by searching the "about" sub-section of the project Web, which provided information on the NetBeans technologies under development, as well as the project structure (e.g., developer roles, key development tasks, designated file-sharing repositories, and file directories) and the nature of its open source status. This project history is a basis for understanding current development practices. However, it also details ways for outsiders to become involved in development and the community at large [NetBeans Contribute 2003]. The modes of contribution can be used to construct an initial set of activity scenarios, which can be described as *use cases* for project or process participation.

Though best known as a tenet of UML, use cases can serve as a notation to model scenarios of activities performed by actors in some role that use one or more tools to manipulate artifacts associated with an enterprise process or activity within it [Fowler and Scott 2000, Viller and Sommerville 2000]. The site map also shows a page dedicated to project governance hyperlinked three layers deep within the site. This page exposes the primary member types, their roles and responsibilities, which suggest additional use cases. Unlike those found through the modes of contribution, the project roles span the breadth of the process, though at a higher level of abstraction. Each use case can encode a process fragment. In collecting use cases, we can extract out concrete actions that can then be assembled into a process description to be modeled, simulated, and enacted.

When aggregated, these use cases can be coalesced into an informal model of a process and its context rendered as a *rich interactive hypermedia*, a semi-structured extension of Monk and Howard's [1998] rich picture modeling construct. The rich hypermedia shown in Figure 1 identifies developer roles, tools, concerns, and artifacts of development and their interaction, which are hyperlinked (indicated as underlined phrases) to corresponding use cases and object/role descriptions (see Figure 2). Such an informal computational model can be useful for newcomers to the community looking to become involved in development and offers an overview of the process and its context in the project, while abstracting away the detail of its activities. The use cases also help identify the requirements for enacting or re-enacting the process as a basis for validating, adapting, or improving the process.

A critical challenge in reconstructing process fragments from a process enactment instance is in knowing whether or not the evidence at hand is related, unrelated, or anomalous. Reliability of associations constructed in this fashion may be strengthened by the frequency of association and the relevance of artifacts carrying the association. If text extraction tools are used to discover elements of process fragments, they must also note the context in which are located in to determine this relevance. One way to do this is using the physical structure of the community Web (i.e. directory structure), as well as its logical structure (referencing/referenced artifacts). In the NetBeans quality -assurance (Q-Build) testing example, we can relate the "defects by priority" graph on the defect summary page[1] to the defect priority results from the Q-Build verification. Likewise, the defect tallies and locations correlate to the error summaries in the automated testing (XTest) results[2]. By looking at the filename and creation dates of the defect graphs, we know which sets of results are charted and how often they are generated. This, in turn identifies the length of the defect chart generation process, and how often it is executed. The granularity of process discovered can be tuned by adjusting the search depth and the degree of inference to apply to the data gathered. An informal visual representation of the artifacts that flow through the requirements and release process is shown in Figure 3.

These process fragments can now be assembled into a formal PML description of the selected processes [Noll and Scacchi 2001]. Constructing such a process model is facilitated and guided by use of an explicit process meta-model [Mi and Scacchi 1996]. Using the PML grammar and software process meta-model,

---

1 http://qa.netbeans.org/bugzilla/graphs/summary.html as of March 2004

2 http://www.netbeans.org/download/xtest-results/index.html as or March 2004

we created an ontology for process description with the Protégé-2000 modeling tool [Noy, *et al*. 2001].

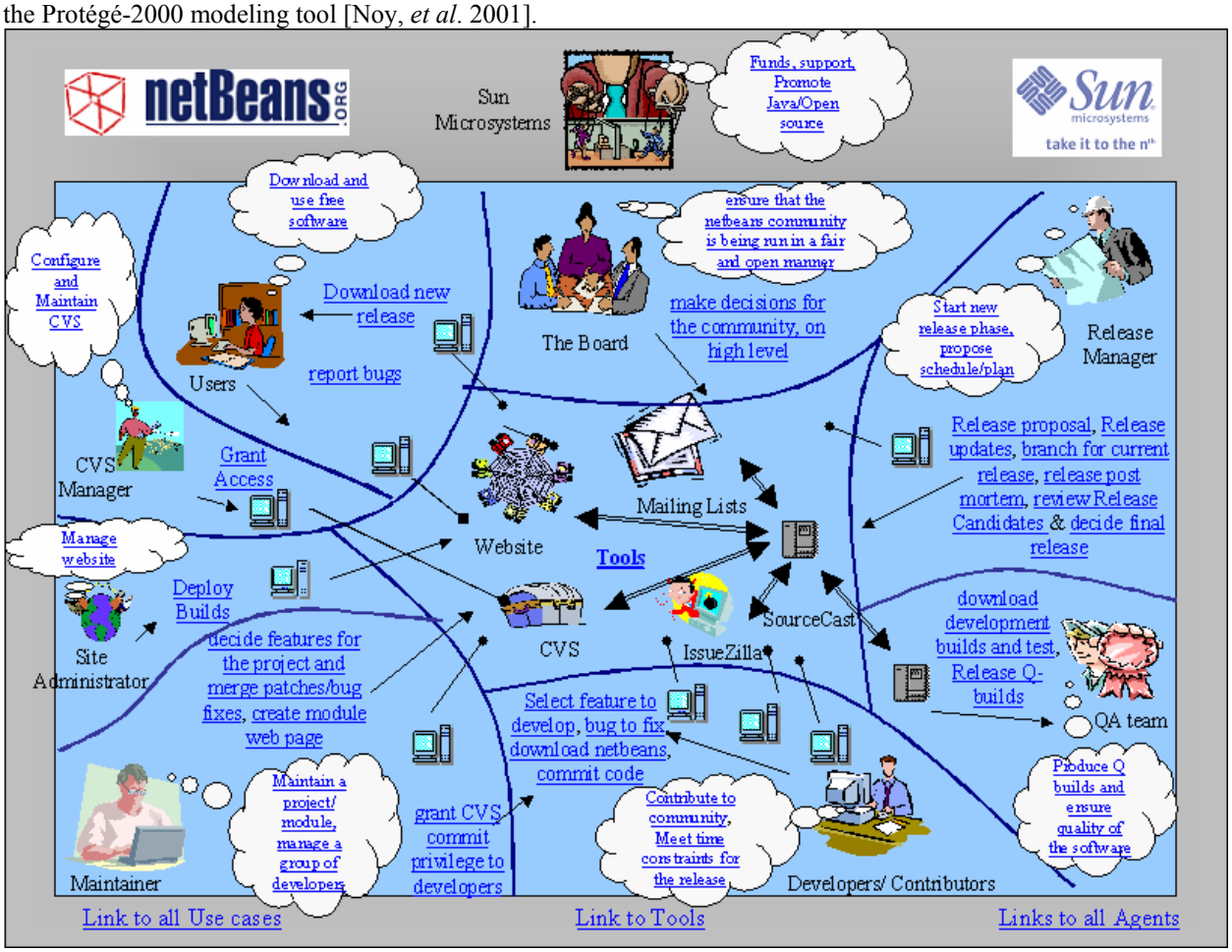

Figure 1. A hyperlinked rich hypermedia of the NetBeans requirements and release process [Oza, *et al.,* 2002]

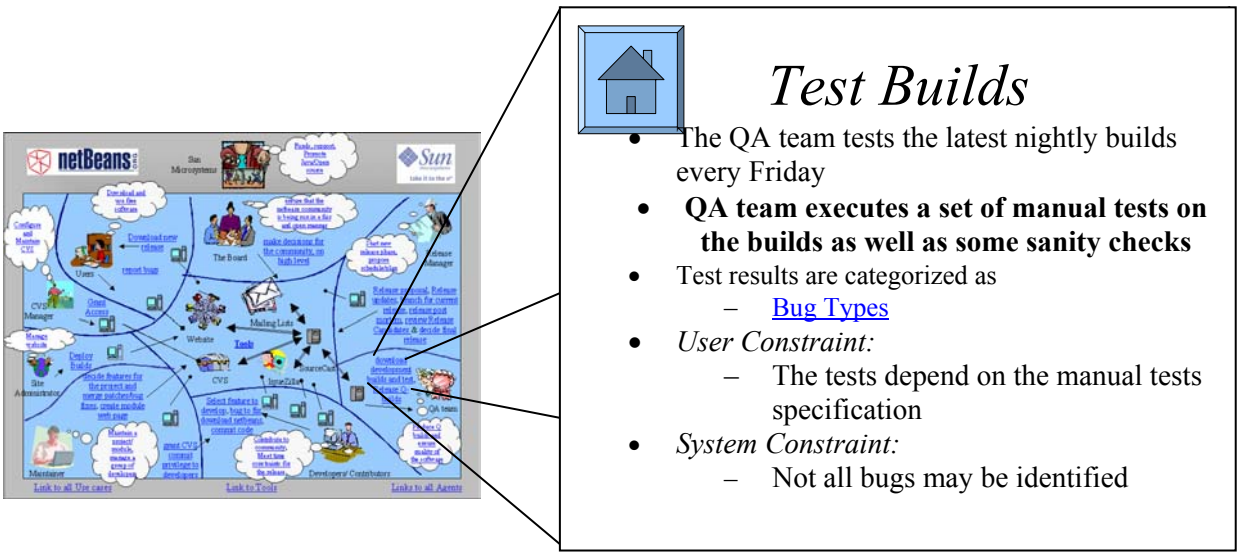

Figure 2. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case.

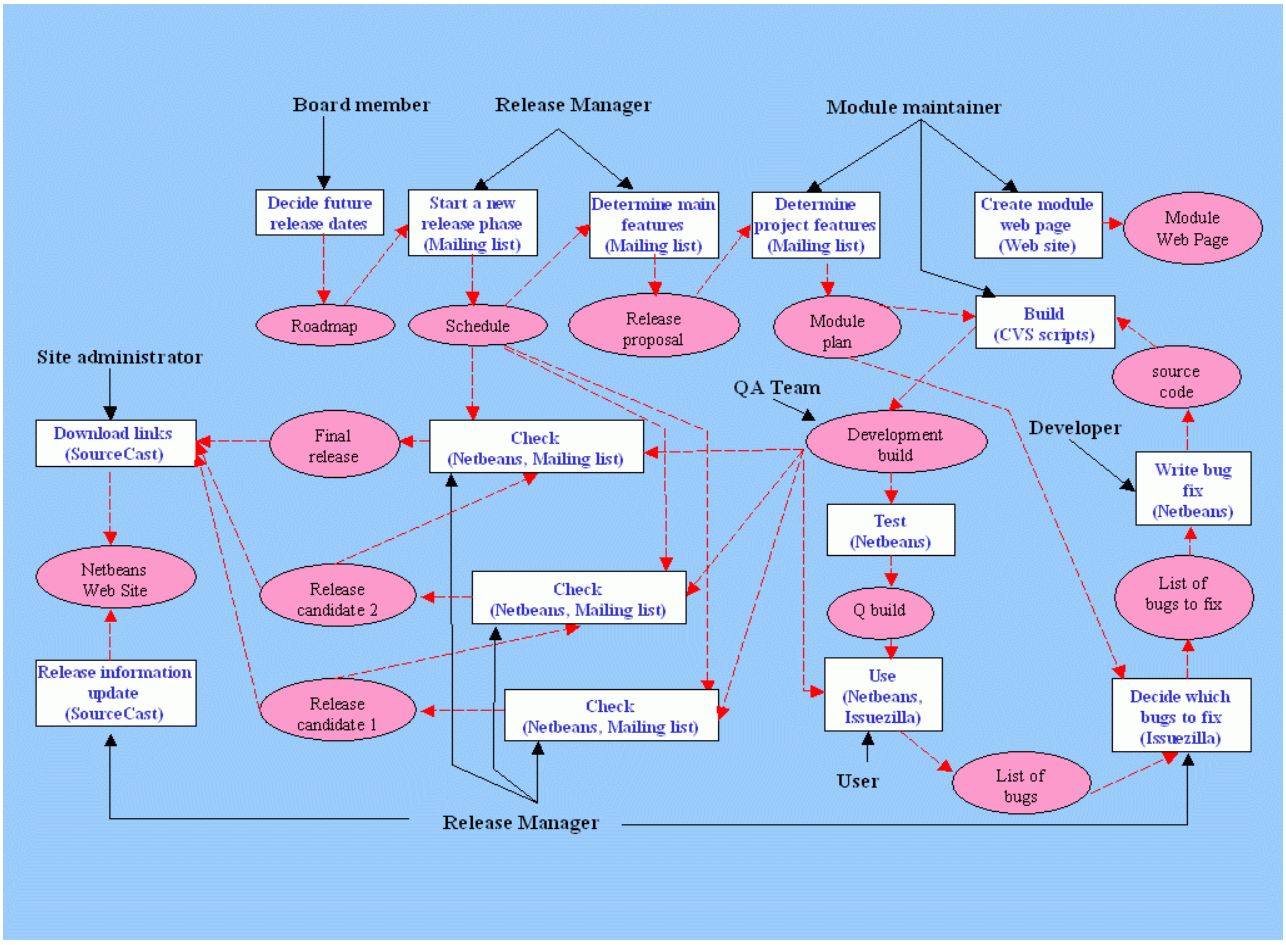

Figure 3. NetBeans Requirements and Release process flow graph [Oza, *et al*., 2002]

```
sequence Test {
   action Execute automatic test scripts {
         requires { Test scripts, release binaries }
         provides { Test results }
         tool { Automated test suite (xtest, others) }
         agent { Sun ONE Studio QA team }
         script { }
   }
   action Execute manual test scripts {
         requires { Release binaries }
         provides { Test results }
         tool { NetBeans IDE }
         agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio
         developers }
         script { }
   }
   iteration Update Issuezilla {
         action Report issues to Issuezilla {
         requires { Test results }
         provides { Issuezilla entry }
         tool { Web browser }
         agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio
         developers }
```

```
        script {
                Navigate to Issuezilla.
                Select issue number/component to update.
        }
}
action Update standing issue status {
        requires { Standing issue from Issuezilla, test results }
        provides { Updated Issuezilla issue repository }
        tool { Web browser }
        agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio
        developers }
        script { }
}
action Post bug stats {
        requires { Test results }
        provides { Bug status report, test result report }
        tool { Web editor, JFreeChart }
        agent { Release manager }
        script { }
```

Figure 4. A PML description of the testing sequence of the NetBeans release process

The PML model builds from the use cases depicted in the rich hypermedia, then distills them a set of actions or sub-processes that comprise the process with its corresponding actor roles, tools, and resources and the flow sequence in which they occur. A sample result of this appears in Figure 4.

6. Process Re-enactment for Deployment, Validation, and Improvement

Since their success relies heavily on broad, open-ended participation, OSSD projects often have informal descriptions of ways members can participate, as well as offer prescriptions for community building [Scacchi 2002]. Although automatically recognizing and modeling process enactment guidelines or policies from such prescriptions may seem a holy grail of sorts for process discovery, there is no assurance that they accurately reflect the process as it is enacted. However, taken with the discovered process, such prescriptions begin to make it is possible to perform basic process validation and conformance analysis by reconciling developer roles, affected artifacts, and tools being used within and across modeled processes or process fragments [cf. Podorozhny, Perry and Osterweil 2003].

As OSSD projects are open to contributions from afar, it also becomes possible to contribute explicit models of discovered processes back to the project under study so that project participants can openly review, independently validate, refine, adapt or otherwise improve their own software processes. Accordingly, we have contributed our process models and analyses of the NetBeans requirements and release process in the form of a public report hosted (and advertised) on the NetBeans.org Web site[3].

Process re-enactment allows us to simulate or prototype process enactments by navigationally traversing a semantic hypertext representation of the process [Noll and Scacchi 2001, Scacchi 2000]. These re-enactment prototypes are automatically derived from a compilation of their corresponding PML process model [Noll and Scacchi 2001]. One step in the process modeled for NetBeans appears in Figure 5. In exercising repeated simulated process enactment walkthroughs, we have been able to detect process fragments that may be unduly lengthy, which may serve as good candidates for downstream process engineering activities such as streamlining and process redesign [Scacchi 2000]. Process re-enactment also allows us, as well as participants in the global NetBeans project, to better see the effects of duplicated work. As an example, we have four agent types that test code. Users may carry out beta testing from a black box perspective, whereas developers, contributors, and SUN Microsystems QA experts may perform more in-depth white-box testing and analysis, and, in the case of developers and contributors, they will not merely submit a report to

[3] See http://www.netbeans.org/community/articles/index.html, as of May 2003.

the IssueZilla issue tracking system,[4] but may also take responsibility for resolving it.

However, is it really necessary to have so many people doing such similar work? While, in this case, the benefits of having more eyes on the problem may justify the costs of involvement (which is voluntary, anyway), in other cases, it may be less clear.

We are also able to detect where cycles or particular activities may be problematic for participants, and thus where process redesign may be of practical value [Scacchi 2000]. Process re-enactment prototypes are a useful means to interactively analyze whether or how altering a process may lead to potential pitfalls that can be discovered before they lead to project failure. Over the course of constructing and executing the prototype we discovered some of the more concrete reasons that there are few volunteers for the release manager position. The role has an exceptional amount of tedious administrative tasks that are critical to the success of the project.

Between scheduling the release, coordinating module stabilization, and carrying out the build process, the release manager has a hand in almost every part of the requirements and release process. This is a good indication that downstream activities may also uncover a way to better distribute the tasks and lighten her/his load.

The self-selective nature of OSSD project participation has many impacts on the development process they use. If any member wishes not to follow a given process, the enforcement of the process is contingent on the tolerance of her/his peers in the matter, which is rarely the case in corporate development processes. If the project proves intolerant of the alternative process, developers are free to simply not participate in the project's development efforts and perform an independent software release build.

6. Conclusion

Our goal is to obtain process execution data and event streams by monitoring the Web information spaces of open source software development projects. By examining changes to the information space and artifacts within it, we can observe, derive, or otherwise discover process activities. In turn, we reconstitute process instances using PML [Noll and Scacchi 2001], which provides us with a formal description of an enactable, low-fidelity model of the process in question, that can be analyzed, simulated, redesigned, and refined for reuse and redistribution. But this progress still begs the question of how to more fully automate the discovery and modeling of processes found in large, global scale OSSD projects.

Our experience with process discovery in the NetBeans project, and its requirements and release process, suggests that a bottom-up strategy for process discovery, together with a top-down process meta-model, can serve as a suitable framework for process discovery, modeling and re-enactment. As demonstrated in the testing example, action sequences are constructed much like a jigsaw puzzle. We compile pieces of evidence to find ways to fit them together in order to make claims about process enactment events, artifacts, or circumstances which may not be obvious from the individual pieces. We find that these pieces may be unearthed in ways that can be executed by software tools that are guided by human assistance [Jensen and Scacchi 2004].

Our approach to discovery, modeling, and reenactment relies on both informal and formal process representations. We constructed use cases, rich pictures, flow graphs as informal but semi-structured process representations which we transformed into a formal process representation language guided by a process meta-model and support tools. These informal representations together with a process meta-model then provide a scheme for constructing formal process descriptions. Thus demonstration of a more automated process discovery, modeling, and re-enactment environment that integrates these capabilities and mechanisms is the next step in this research.

Finally, it is important to recognize that large OSSD projects are diverse in the form and practice of their software development processes. Our long-term goal in this research is to determine how to best support a more fully automated approach to process discovery, modeling and re-enactment. Our study provides a case study of a real-world process in a complex global OSSD project to demonstrate the feasibility of such an approach. Subsequently, questions remain as to which OSSD processes are most amenable to such an approach, which are likely to be of high value to the host project or other similar projects, and whether all or some OSSD projects are more/less amenable to such discovery and modeling given the richness/paucity of their project information space

[4] See http://www.netbeans.org/kb/articles/issuezilla.html, as of March 2004.

and diversity of artifacts. As government agencies, academic institutions and industrial firms all begin to consider or invest resources into the development of large OSS systems, then they will seek to find what the best OSSD processes are, or what OSSD practices to follow. Thus discovery and explicit modeling of OSSD processes in forms that can be

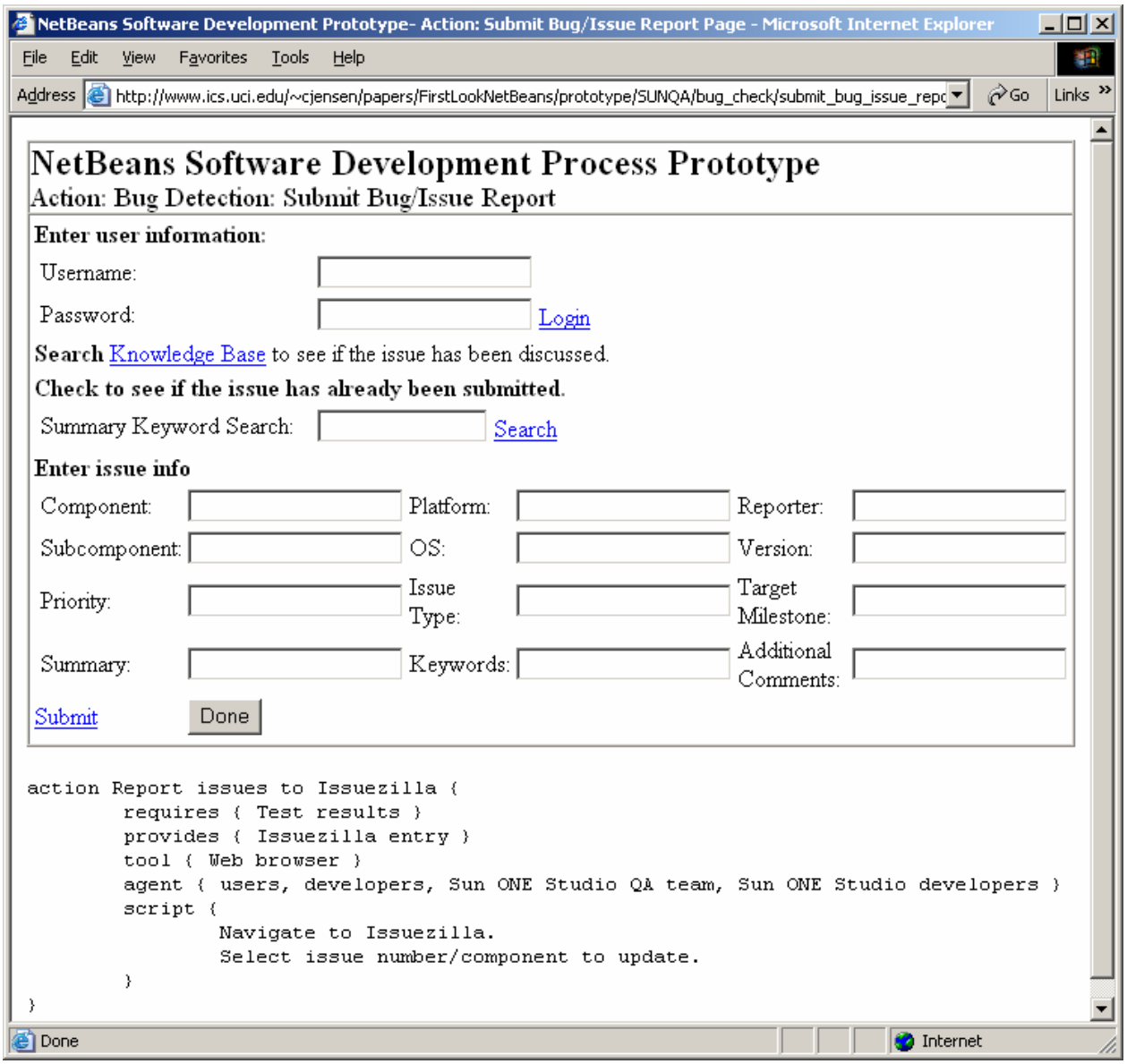

Figure 5. An action step in the re-enactment of the NetBeans requirements and release process.

shared, reviewed, modified, re-enacted, and redistributed appears to be an important topic for further investigation, and this study represents a step in this direction.

8. Acknowledgements

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075, #ITR-0205679 and #ITR-0205724. No endorsement implied. Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; and Margaret Elliott at the UCI Institute for Software Research are collaborators on the research described in this paper.

9. References

Cass, A.G., Lerner, B., McCall, E., Osterweil, L. and Wise, A. 2000. Little JIL/Juliette: A process definition language and interpreter. *Proc. 22nd*

Intern. Conf. Software Engineering, 754-757, Limerick, Ireland, June.

Cook, J. and Wolf, A.L. 1998. Discovering Models of Software Processes from Event-Based Data, *ACM Trans. Software Engineering and Methodology*, 7(3), 215-249.

Eclipse Web Site, 2003. http://www.eclipse.org

Elliott, M. and Scacchi, W., Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Publishing, 2004.

Emmerich, W. and Gruhn, V., FUNSOFT Nets: a Petri-Net based Software Process Modeling Language, *Proc. 6th ACM/IEEE Int. Workshop on Software Specification and Design*, Como, Italy, IEEE Computer Society Press, 175-184, 1991.

Fowler, M. and Scott, K. 2000. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Second Ed. Addison Wesley: Reading, MA.

Garg, P.K. and Bhansali, S. 1992. Process programming by hindsight. *Proc. 14th Intern. Conf. Software Engineering*, 280-293.

Jensen, C. and Scacchi, W. Applying a Reference Framework to Open Source Software Process Discovery, in *Proc. 1st Workshop on Open Source in an Industrial Context,* OOPSLA-OSIC03, Anaheim, CA October 2003.

Jensen, C. and Scacchi, W. 2004. Data Mining for Software Process Discovery in Open Source Software Development Communities, submitted for publication.

Mi, P. and Scacchi, W. 1996. A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330.

Mockus, A., Fielding, R., and Herbsleb, J., Two Case Studies in Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Engineering and Methodology*, 11(3), 309-346, 2002.

Monk, A. and Howard, S. 1998. The Rich Picture: A Tool for Reasoning about Work Context. *Interactions,* 21-30, March-April.

NetBeans Open Source Project, 2003. http://www.netbeans.org

Noll, J. and Scacchi, W. 2001. Specifying Process Oriented Hypertext for Organizational Computing. *Journal of Network and Computer Applications* 24 39-61,

Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W. and Musen, M.A. 2001. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems,* 16(2), 60-71.

Osterweil, L. 2003. Modeling Processes to Effectively Reason about their Properties, *Proc. ProSim'03 Workshop,* Portland, OR, May 2003.

Oza, M., Nistor, E., Hu, S. Jensen, C., and Scacchi, W. 2002. *A First Look at the Netbeans Requirements and Release Process*, http://www.ics.uci.edu/cjensen/papers/FirstLookNetBeans/

Podorozhny, R.M., Perry, D.E., and Osterweil, L. 2003, Artifact-based Functional Comparison of Software Processes, *Proc. ProSim'03 Workshop,* Portland, OR, May 2003.

Scacchi, W., 2000. Understanding Software Process Redesign using Modeling, Analysis, and Simulation, *Software Process—Improvement and Practice*, 5(2/3), 183-195.

Scacchi, W., 2002. Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings—Software*, 149(1), 25-39.

Scacchi, W., 2004, Free/Open Source Software Development Practices in the Game Community, *IEEE Software*, 21(1), 59-67, Jan-Feb. 2004.

Viller, S., and Sommerville, I., 2000. Ethnographically Informed Analysis for Software Engineers, *Intern. J. Human-Computer Interaction*, 53, 169-196.

Wolf, A.L. and Rosenblum, D.S. 1993. A Study in Software Process Data Capture and Analysis. *Proc. Second Intern. Conf. on the Software Process*,115-124, IEEE Computer Society.