

## *Chapter 1*

# **DISCOVERING, MODELING, AND RE-ENACTING OPEN SOURCE SOFTWARE DEVELOPMENT PROCESSES: A CASE STUDY**

Chris Jensen and Walt Scacchi

*Institute for Software Research*

*Donald Bren School of Information and Computer Science*

*University of California, Irvine*

*Irvine, CA USA 92697-3425*

*Email: {cjensen, wscacchi}@ics.uci.edu*

Software process discovery has historically been a labor and time intensive task, either done through exhaustive empirical studies or in an automated fashion using techniques such as logging and analysis of command shell operations. While empirical studies have been fruitful, data collection has proven to be tedious and time consuming. Existing automated approaches have very detailed, low level but not rich results. We are interested in process discovery in large, globally distributed organizations such as the NetBeans open source software development community, which currently engages over twenty thousand developers distributed over several continents working collaboratively, sometimes across several stages of the software lifecycle in parallel. This presents a challenge for those who want to join the community and participate in, as well as for those who want to understand these processes. This chapter discusses our efforts to discover selected open source processes in the NetBeans community. We employ a number of data gathering techniques ranging from ethnographic to semi-structured to formal, computational models, which were fed back to the community for further evaluation. Along the way, we discuss collecting, analyzing, and modeling the data, as well as lessons learned from our experiences.

## 1. Introduction

The Peopeware vision is an attempt to provide insight into the social qualities of project management that may lead to project success or failure. In a similar sense, open source software development (OSSD) has been effective in providing online social workscapes that have become the focus of attention in industry and research conferences alike. However, in order to understand and participate in these processes, people new to these processes must first discover what they are and how they operate. 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 OSSD projects, such as those associated with the Mozilla Web browser, Apache Web server<sup>1</sup>, and Java-based integrated development environments for creating Web applications like NetBeans<sup>2</sup> and Eclipse<sup>3</sup>. The challenge we face is similar to what prospective developers and corporate sponsors who want to join a given OSSD project face, and thus our efforts should yield practical results.

Process models are *prescriptive* if they state what activities should be done or *proscriptive* if they describe what activities could be done. With process discovery, our task is to create *descriptive* models by determining what activities have been done. OSSD projects, however, do not typically employ or provide explicit process model prescriptions, proscriptions, descriptions, 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 in projects like NetBeans by computer-assisted search and analysis of the project's content, structure, update and usage patterns associated with 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<sup>4, 5, 6</sup>. Similarly, events that denote updates to these sources are also publicly accessible,

and thus suitable for analysis. Though traditional ethnographic approaches to software process discovery<sup>7</sup> net a wealth of information with which to model, simulate, and analyze OSSD processes, they are time and labor-intensive. As a result, they do not scale well to the study of multiple OSSD development projects of diverse types in a timely manner. Subsequently, this suggests the need for a more automated approach that can facilitate process discovery.

In our approach, we examine three types of information in the course of discovering and modeling OSSD processes. First are the kinds of OSSD artifacts (source code files, messages posted on public discussion forums, Web pages, etc.). Second are the artifact update events (version release announcements, Web page updates, message postings, etc.). Third are work contexts (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 model that associates these data with software processes and process models<sup>8</sup>. Whereas the meta-model describes the attributes of process events and how they may be arranged (i.e. the language of the process), the reference model describes types and known instances of those attributes. 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

Process event notification systems have been used in many contexts, including process discovery and analysis<sup>9, 10</sup>. 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<sup>11</sup>. 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.

Recently, there have been a number of developments focused on mining software repositories<sup>12, 13</sup>. While these have yielded interesting insights into patterns of software development in OSSD communities, most of the work has focused on low-level social network analysis of artifacts and agents of software development rather than processes of software development.

Lastly, 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 process programming language<sup>14, 15</sup> and the PML process modeling and enactment language<sup>16</sup>, 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 kinds. Thus, we employ this practice here.

### **3. Problem Domain**

We are interested in discovering, modeling, and simulating re-enactment of 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<sup>5</sup>. 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 and captured in the process reference model. 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<sup>5, 6</sup>. 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 be 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 process discovery must be able to efficiently access, collect, and analyze the data across the project Web space. Such data includes 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<sup>5,6</sup>.

How the project organizes its information space may indicate what types of artifacts they generate. For example, a project Web page containing a public file directory named “x-test-results” can be examined to determine whether there is evidence that some sort of testing (including references to test cases and test results) has been conducted. Furthermore, timestamps associated with file, object, or Web page 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. Elsewhere, the types of artifacts available on the site can 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 a graphic image file (a Web-compatible image map) 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, as well as 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.

#### 4. Field Site and Process Description

To demonstrate the viability of our process discovery approach, we describe how we apply it through a case study. For this task, we examine a selected process in the NetBeans<sup>2</sup> OSSD project. The NetBeans project started in 1996 as a student project before being picked up and subsequently made an OSSD project by Sun Microsystems. The NetBeans project community is now an effort combining dozens of organizations (4 distributing releases, 42 writing extensions, and 21 building tools based on the NetBeans platform)<sup>1</sup> and boasts of over one hundred thousand developers around the globe<sup>2</sup>. The scale of the project thus necessitates developers to transparently coordinate their efforts and results in a manner that can be accessed and persist on the community Web site. As demonstrated in the previous section, this coordination evidence forms the basis from which processes may be identified and observed.

The *requirements assertion and release* process was chosen for study because its activities have short duration, are frequently enacted, and have a propensity for available evidence that could potentially be extracted using automated technologies. The process was discovered, modeled informally and formally, then prototyped for analysis and re-

---

<sup>1</sup> <http://www.netbeans.org/about/third-party.html>, as of October 2004

<sup>2</sup> <http://www.netbeans.org/community/news/index.html#494>, as of October 2004

enactment. The next two sections describe the methods we use to discover, model, and re-enact the requirements and release process found in the NetBeans OSSD project. Along the way, we present descriptions of the process under study using informal, semi-structured, and formal models, and the formal models are then further analyzed through a process enactment simulator we use for process re-enactment. Process re-enactment in turn allows us to further validate our models, as well as serve to refine and improve the discovered processes as feedback to the OSSD project in the study on possible process improvement opportunities.

## 5. Process Discovery and Modeling

The discovery and modeling approach used in this case study consisted of defining the process meta-model and reference model for the selected process, as described above. Next, we gathered data from the community Web, indexing it according to the reference model with an off-the-shelf search engine and correlating it based on its structure, content, and update context (usage data being unavailable for this project). Our experience has shown that it is best to view process discovery and modeling as a progressive activity. That is, we utilize several models at different levels of abstraction that become progressively more formal. The first of these depicts activity scenarios that reflect the correlation of tools, resources, and agents in the performance of development activities (i.e. instances of the attributes of the process meta-model). These are then refined into a flow graph illustrating more concretely, the order in which the activity scenarios are performed and lastly, a formal, computational process model. Moreover, progressive discovery can reduce collection of unrelated “noisy” process data by using coarsely grained data to direct the focus of discovery of more finely grained data.

As our results stem from empirical observations of publicly available artifacts of the NetBeans community Web, they face certain threats to validity. Cook *et al.*<sup>17</sup> demonstrated the validity of using the kinds of observations described above in terms of constructing process models, as well as showing internal and external consistency. Unlike Cook and Wolf, 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 locate and identify possible indicators that a given activity has occurred. This reference model was devised through a review of several open source communities (NetBeans, Mozilla, and Apache, among others). However, it must be updated to reflect the evolution of the types (and names) of tools, activities, roles, and resources in OSSD, in particular those of the particular community subject to process discovery to maintain the validity of our methodology. The full results of our case study may be found in<sup>18</sup>. Subsequent discovery of our study by the community and our discussions with prominent community members that followed verified our results and provided additional insight. This feedback allows both process modelers and process participants opportunities for mutual improvement of methods and the outputs they produce (i.e. process modeling techniques and process models, as well as software development processes and software products). The discussion of our process discovery and modeling methods and results follows next.

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 Web. The information gathered here is used to configure the OSSD process reference model<sup>19</sup>. 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.

Guided by our framework, searching the “about” sub-section of the project Web site 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 structure 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<sup>3</sup>. 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 the unified modeling language (UML), use cases can serve as a notation to model scenarios of activities performed by actors in some role<sup>20, 7</sup>. 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<sup>21</sup> 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.

---

<sup>3</sup> <http://www.netbeans.org/community/contribute>, as of June 2004

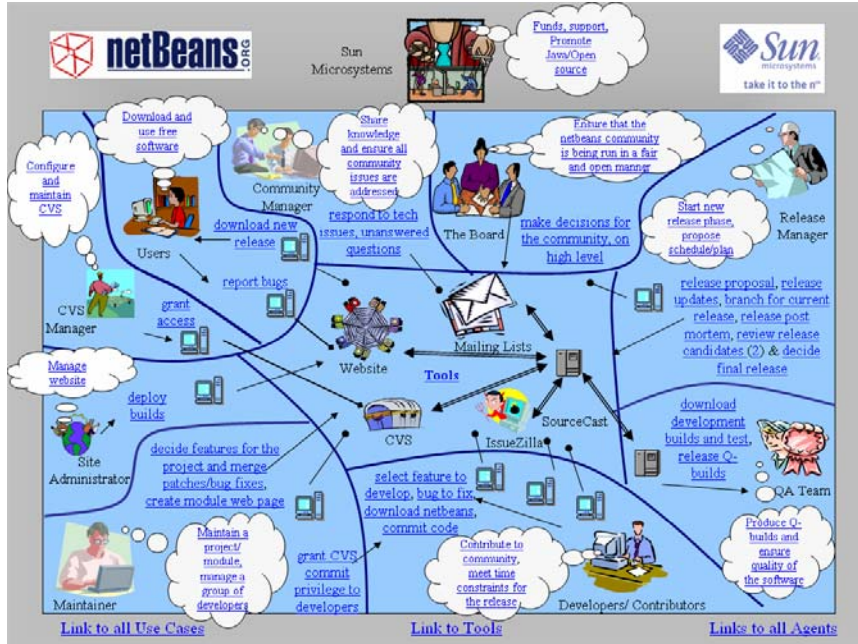


Figure 1. A hyperlinked rich hypermedia of the NetBeans requirements and release process<sup>18</sup>

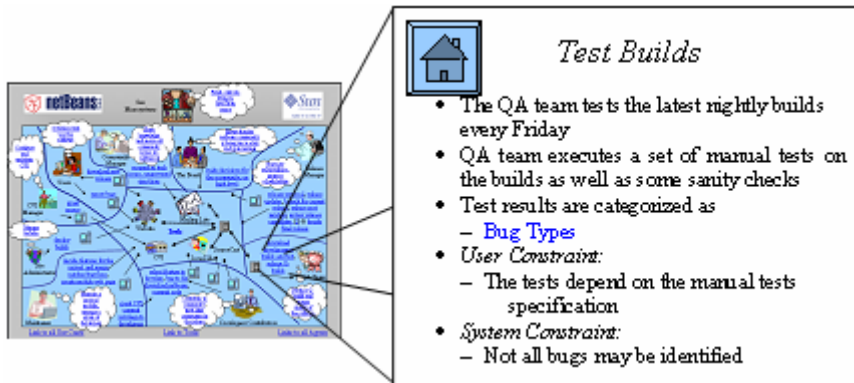


Figure 2. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case

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. The frequency of association and the relevance of artifacts carrying the association may strengthen the reliability of associations constructed in this fashion. If text extraction tools are used to discover elements of process fragments, they must also note the context in which they are located to determine this relevance. One way to do this is using the physical structure of the community Web site (i.e. its directory structure), as well as the logical structure of the referencing/referenced artifacts (the site's information architecture). In the NetBeans quality assurance (Q-Build) testing example, we can relate the "defects by priority" graph on the defect summary page<sup>4</sup> 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<sup>5</sup>. 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<sup>16</sup>. Constructing such a process model is facilitated and guided by use of an explicit process meta-model<sup>8</sup>. Using the PML grammar and software process meta-model, we created an ontology for process description with the Protégé-2000 modeling tool<sup>22</sup>.

The PML model builds from the use cases depicted in the rich hypermedia, then distills from 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.

---

<sup>4</sup> <http://qa.netbeans.org/bugzilla/graphs/summary.html> as of March 2004

<sup>5</sup> <http://www.netbeans.org/download/xtest-results/index.html> as of March 2004

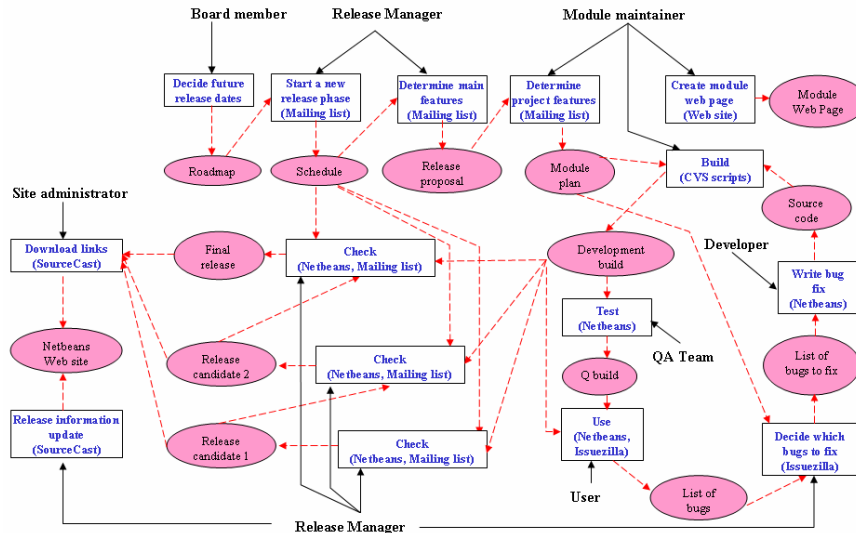


Figure 3. NetBeans Requirements and Release process flow graph<sup>18</sup>

## 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 offering prescriptions for community building<sup>5</sup>. 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 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<sup>23</sup>.

```

1. sequence Test {
2.   action Execute automatic test scripts {
3.     requires { Test scripts, release binaries }
4.     provides { Test results }
5.     tool { Automated test suite (xtest, others) }
6.     agent { Sun ONE Studio QA team }
7.   }
8.   action Execute manual test scripts {
9.     requires { Release binaries }
10.    provides { Test results }
11.    tool { NetBeans IDE }
12.    agent { users, developers, Sun ONE Studio QA team,
13.             Sun ONE Studio developers }
14.  }
15. iteration Update Issuezilla {
16.   action Report issues to Issuezilla {
17.     requires { Test results }
18.     provides { Issuezilla entry }
19.     tool { Web browser }
20.     agent { users, developers, Sun ONE Studio QA
21.              team, Sun ONE Studio developers }
22.   }
23.   action Update standing issue status {
24.     requires { Standing issue from Issuezilla, test
25.                results }
26.     provides { Updated Issuezilla issue repository }
27.     tool { Web browser }
28.     agent { users, developers, Sun ONE Studio QA
29.              team, Sun ONE Studio developers }
30.   }
31.   action Post bug stats {
32.     requires { Test results }
33.     provides { Bug status report, test result report }
34.     tool { Web editor, JFreeChart }
35.     agent { Release manager }
36.   }
37. }

```

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

As hinted earlier, because 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 advertised on the NetBeans.org Web site<sup>6</sup>.

Process re-enactment allows us to simulate or prototype process enactments by navigationally traversing a semantic hypertext representation of the process<sup>16, 24</sup>. These re-enactment prototypes are automatically derived from a compilation of their corresponding PML process model<sup>16</sup>. One step in the process modeled for NetBeans appears in Figure 5.

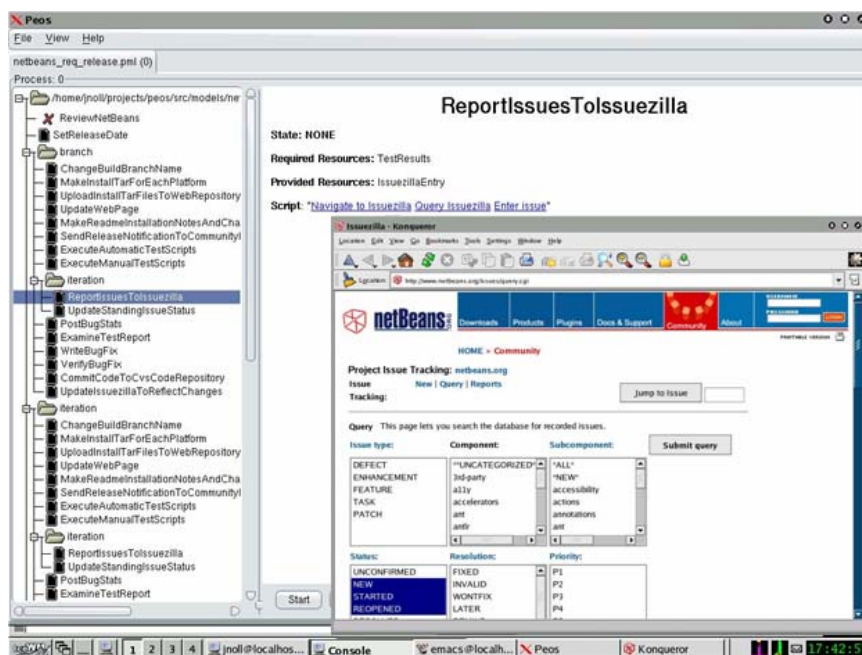


Figure 5. An action step in the re-enactment of the NetBeans requirements and release process

<sup>6</sup> See [http://www.netbeans.org/community/articles/UCI\\_papers.html](http://www.netbeans.org/community/articles/UCI_papers.html), as of October 2004

In exercising repeated process re-enactment walkthroughs, we have been able to detect process fragments that may be unduly lengthy, which may serve as good candidates for streamlining and process redesign<sup>24</sup>. Process re-enactment also allows us, as well as participants in the global NetBeans project, to better see the effects of their 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. In the case of developers and contributors, they will not merely submit a bug report or unsuccessful testing result to the IssueZilla issue tracking system,<sup>7</sup> 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<sup>24</sup>. Process re-enactments can also be treated as process prototypes in order to interactively analyze whether or how altering a process may lead to potential pitfalls that can be discovered before being deployed. Over the course of constructing and executing our prototype we discovered some concrete reasons for why there are few volunteers for the release manager position. The role has an exceptional amount of tedious administrative tasks. However, as these tasks are critical to the success of the project it might be more effective to distribute these tasks to others.

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 in use. If any member does not want

---

<sup>7</sup> See <http://www.netbeans.org/kb/articles/issuezilla.html>, as of March 2004

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.

## **7. Conclusion**

Our desire is to obtain and model 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 and abstract process instances into PML<sup>16</sup>, which provides us with a formal description of an enactable, low-fidelity model of the process in question. Such a formal process model 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 acting as a reference model, can serve as a suitable framework for process discovery, modeling and re-enactment. As demonstrated in the testing activity 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 that 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<sup>25</sup>.

The approach to discovery, modeling, and re-enactment described in this chapter relies on a variety of informal and formal process representations. We constructed use cases and rich hypermedia pictures as informal process descriptions, 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 basis for constructing formal process descriptions. Thus demonstration of a more automated process discovery, modeling, and re-enactment environment that integrates these capabilities and mechanisms into a more streamlined and more automated environment is the next step in this research. We anticipate that such an environment will yield additional venues for tool assistance in process data collection and analysis.

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, and which are likely to be of high value to the host project or other similar projects. Furthermore, we need to establish whether all or only some OSSD projects are more/less amenable to such discovery and modeling given the richness/paucity of their project information space 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 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.

### **Acknowledgements**

The research described in this report is supported by grants from the National Science Foundation #0083075, #0205679, #0205724, and #0350754. 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; Margaret Elliott and

others at the UCI Institute for Software Research are collaborators on the research described in this paper.

## References

1. Mockus, A., Fielding, R. and Herbsleb, J. 2002. Two Case Studies in Open Source Software Development: Apache and Mozilla, *ACM Trans. Software Engineering and Methodology*, 11(3), 309-346.
2. *NetBeans Open Source Project*, 2003. <http://www.netbeans.org>
3. *Eclipse Web Site*, 2003. <http://www.eclipse.org>
4. Elliott, M. and Scacchi, W. 2004. Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development*, Idea Publishing.
5. Scacchi, W. 2002. Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings—Software*, 149(1), 25-39.
6. Scacchi, W. 2004. Free/Open Source Software Development Practices in the Game Community, *IEEE Software*, 21(1), 59-67, Jan-Feb. 2004.
7. Viller, S. and Sommerville, I. 2000. Ethnographically Informed Analysis for Software Engineers, *Intern. J. Human-Computer Interaction*, 53, 169-196.
8. Mi, P. and Scacchi, W. 1996. A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330.
9. 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.
10. 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.
11. Garg, P.K. and Bhansali, S. 1992. Process programming by hindsight, *Proc. 14<sup>th</sup> Intern. Conf. Software Engineering*, 280-293.
12. Sandusky, R., Gasser, L., and Ripoche, G. 2004. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community, *Proc. MSR'04 Workshop*, Edinburgh, Scotland, May 2004.
13. Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J. 2004. Applying Social Network Analysis to the Information in CVS Repositories, *Proc. MSR'04 Workshop*, Edinburgh, Scotland, May 2004.
14. Cass, A.G., Lerner, B., McCall, E., Osterweil, L. and Wise, A. 2000. Little JIL/Juliette: A process definition language and interpreter, *Proc. 22<sup>nd</sup> Intern. Conf. Software Engineering*, 754-757, Limerick, Ireland, June.
15. Osterweil, L. 2003. Modeling Processes to Effectively Reason about their Properties, *Proc. ProSim'03 Workshop*, Portland, OR, May 2003.

16. Noll, J. and Scacchi, W. 2001. Specifying Process Oriented Hypertext for Organizational Computing, *Journal of Network and Computer Applications*, 24 39-61.
17. Cook, J., Votta, L. and Wolf, A.L. 1998. Cost-Effective Analysis of In-Place Software Processes, *IEEE Transactions on Software Engineering*, 24(8), 650-663.
18. 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/>
19. Jensen, C. and Scacchi, W. 2003. Applying a Reference Framework to Open Source Software Process Discovery, *Proc. 1<sup>st</sup> Workshop on Open Source in an Industrial Context*, OOPSLA-OSIC03, Anaheim, CA, October 2003.
20. Fowler, M. and Scott, K. 2000. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Second Ed. Addison Wesley: Reading, MA.
21. Monk, A. and Howard, S. 1998. The Rich Picture: A Tool for Reasoning about Work Context. *Interactions*, 21-30, March-April 1998.
22. 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.
23. 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.
24. Scacchi, W. 2000. Understanding Software Process Redesign using Modeling, Analysis, and Simulation, *Software Process—Improvement and Practice*, 5(2/3), 183-195.
25. Jensen, C. and Scacchi, W. 2004. Data Mining for Software Process Discovery in Open Source Software Development Communities, submitted for publication.