Simulating an Automated Approach to Discovery and Modeling of 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

August 2003
Previous Version: May 2003

Abstract

Process discovery has been shown to be a challenging problem offering limited results, however, most work has been conducted in closed source systems. This paper describes a new approach to process discovery that examines the Internet information spaces of open source software development projects. In searching for an automated solution to the process discovery problem, we first have simulated it manually by searching the Web space for evidence of process activities and reconstructing process fragments based on the clues discovered. The goal of this approach is to help reveal its challenges, strengths, weaknesses and findings such that they may be employed to determine the requirements and design of more fully automated process discovery and modeling mechanisms that can be applied to Web-based software development projects.

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

1. Introduction

The goal of our work is to develop new and automatable 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 (OSSD) projects, such as those associated with the Apache Web server [2003], Mozilla Web browser [2003], 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 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 [Scacchi 2002, Viller and Sommerville 2000]. Likewise, corresponding events that denote updates to these sources are also publicly accessible. Though this manual discovery approach netted a wealth of information with which to model, simulate, and analyze OSSD processes, it suffers from a lack of scalability when applied to the study of multiple OSSD development projects. Similarly, it suggests the need for an automated approach to that can more readily facilitate process discovery and modeling.

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 using automated tools operating 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 and modeled 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 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.

2. Related Work

Event notification systems have been used in many contexts [Cook and Wolf 1995, Hilbert and Redmiles 1997]. However, of the systems promising automated event notification, many require the process actant 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 collection from command shell histories, applying inference techniques to construct process model fragments from event patterns, which imparts additional inconvenience on the user and relies on his willingness to use the particular tools that observe events. By doing so, the number of process actants 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.

Cook [1995, 1996] utilized algorithmic and statistical inference techniques 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 Nets date back to the 1970's [Petersen 1977] with a wide variety of activity and state-chart diagrams in between. It appears however that these representations suffer from a lack of scalability when

applied to a process situated within an organizational context involving multiple tools, diverse artifact types, and multiple development roles across multiple networked sites.

While admitting that some massaging of the event-stream data was necessary to eliminate spurious events not pertaining to the process in focus, Cook notes the robustness of the Markov model in the face of noisy data. Realizing that the Markov model does not account for previous events in the determination of the next state, Cook experimented with Bayesian Markov transformations using conditional probabilities. Although they disregarded the results as insignificant in comparison to those obtained without the extension, Cadez, *et al.* [2000] points out that joint probabilities tend to be more informational than conditional probabilities, based on their studies that seek to discover user navigation processes on Web sites.

Garg [1992] looks at the problem of specifying an algorithm for software processes without knowing about them beforehand. He suggests process fragments are more easily described with hindsight than foresight. The implications of this work suggest that our approach to process discovery may also gain from such hindsight. Garg advises that rather than looking at the entire process, we instead focus on creating partial process specifications that may overlap with one another. This also reflects variability in software process enactment across iterations.

Elsewhere, new approaches which are as of yet untried in automating process discovery and modeling from empirical data sources include Web data mining and probabilistic relational modeling (PRM) [Getoor, *et al.* 2001, Koller 1999]. Cooley, *et al*. [1997] are among those examining tools and methods for mining usage patterns on the Web, as well as the challenges that apply to analysis done on objects accessed from a remote server. Process discovery in OSSD projects possesses similar challenges. Cooley, *et al*. propose an architecture for Web usage mining and a taxonomy of tasks in Web mining which provides a new consideration in how to automate the discovery and modeling of OSSD processes. PRM on the other hand seeks to overcome the shortfalls of Bayesian networks, which are unsuitable for representing complex structured, flexible domains [Koller 1999]. PRM give us a means of representing relationships between tasks, actors, tools, and resources that we may be able to easily characterize using an informal model of process and context, such as a rich picture [Monk and Howard 1998], but have been difficult to show in a single view of a formal model [Getoor, *et al*. 2001] and the uncertainty of these relationships resulting from the intrinsic dynamism of software processes.

Last, while process research has yielded a plethora of 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. We also found it fruitful to help convey our findings about software processes, and the contexts in which they occur, using both informal and formal representations of these kinds. 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 of software engineering [Scacchi 2002]. Past approaches have shown process discovery to be difficult, yielding limited results. However advances in the areas of Web data mining, knowledge discovery, and PRM provide us with new ways of viewing the problem space and tools for exploring it in working towards building a more automatable solution. Furthermore, our discovery efforts are not random probes in the dark. Instead, we capitalize on contextual aids offered by the domain. Such clues include:

- Project Web information structure -- how the Web site of project-related software development artifacts is organized
- Site contents -- what types of artifacts exist and what information attributes and content they contain
- Usage patterns -- user interaction and content update patterns within the project Web.

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.

4. Approach

As a prelude to developing an automated process discovery solution, we first have simulated a knowledge-enabled approach with another type of intelligent agent, namely people trained in software processes and software process modeling techniques [Oza, *et al*. 2002]. In doing so, we have taken on the role of such automated tools as we would like to assist us in mining information spaces for process evidence [cf. Viller and Sommerville 2000]. For this task, we examined a selected process in the NetBeans [2003] project 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 in ways that could be employed by our prospective technologies. The process was discovered, modeled informally and formally, then prototyped for simulated analysis and reenactment, as described next.

5. Results

The discovery process began with a cursory examination of the project Web space in order to ascertain what types of information were available to us and where that information might be located within the Web. The project site map provided not only a breakdown of 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's approach, we apply *a priori* knowledge of software development to discovering processes. Instead of randomly probing the information space, we construct a reference model from a process meta-model [Mi and Scacchi 1996] detailing possible indicators that a given activity has occurred. 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 the Unified Modeling Language (UML) [Fowler and Scott 2000], 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 [Phalp and Cox 2003]. 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 constitute additional use case information. 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 model can be especially 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 the process.

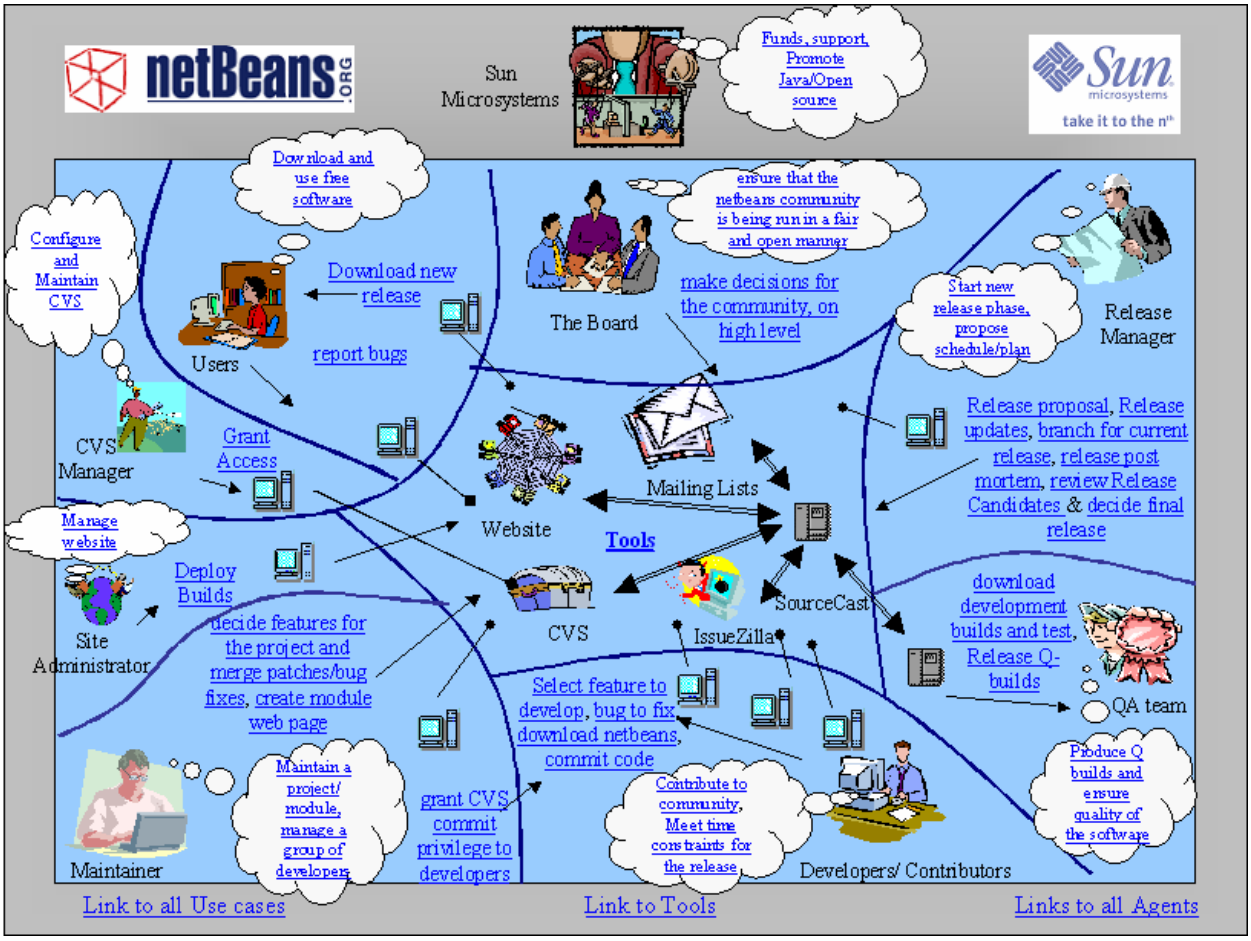

Figure 1. A hyperlinked rich hypermedia of the NetBeans Requirements and Release process [Oza, *et al.,* 2002]

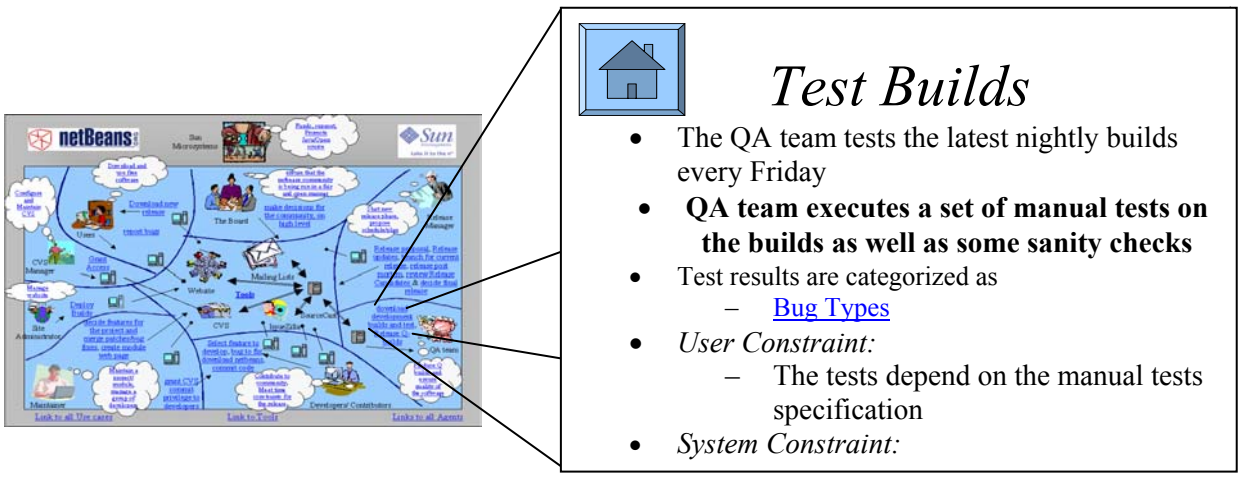

Figure 2. A hyperlink selection within a rich hypermedia presentation revealing details of a corresponding use case.

By text-searching the "defect" page [NetBeans Defect 2003], we can determine through the page location that quality assurance (QA) plays a role in the release process. We can enumerate some of the tools involved in the QA process (e.g. Bugzilla and JFreeChart). We can tell that there is some computation of total defects, their priority levels, and component location. Going a step further, we notice that these defect totals are tallied daily from test results listed on another page, which is neither linked to the charts on the defect page, nor located under the QA module directory. If we consider another clue, that the link to the test results indicates that the tests are automated using XTest, we can construct a sequence of events as follows. Every day, the build is tested against the XTest framework. Additional probing of the QA branch of the Web space turns up another set of tests: Q-Build verification. The Q-Build program is a hybrid manual and automated testing process conducted by the SUN Microsystems QA team to perform sanity checks on periodic builds. The Q-Build Verification page documents these results and the priority in which defects should be addressed. This information is fed in to JFreeChart along with the XTest results to generate the diagrams for the defect page, as shown in Figure 3. The products of these actions are then posted on their respective Web pages. In this way, we have been able to infer a sequence of events given such evidence in bottom up fashion. Further, we have been able to associate tools that correspond to each event and their relationship as determined by their input and output resources.

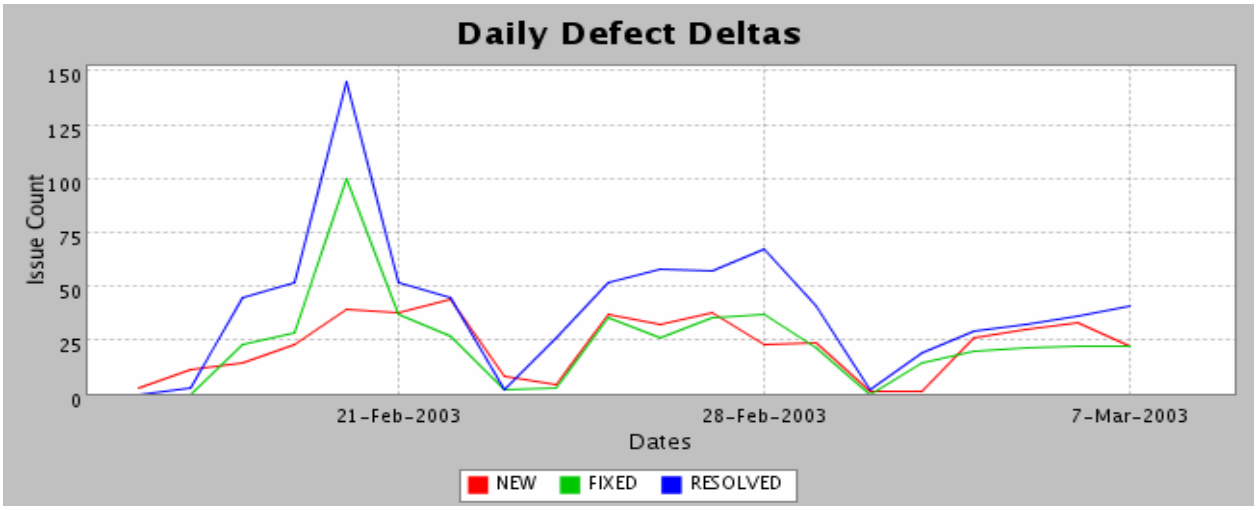

Figure 3. NetBeans Daily Defect Count Chart

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, assuming the use of an unbiased search engine, the rank of artifacts carrying the association. In the above example, we can relate the "defects by priority" graph on the defect page to the defect priority results from the Q-Build verification. Likewise, the defect tallies and locations correlate to the error summaries in the XTest results. By looking at the curvature and dates listed on 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 4.

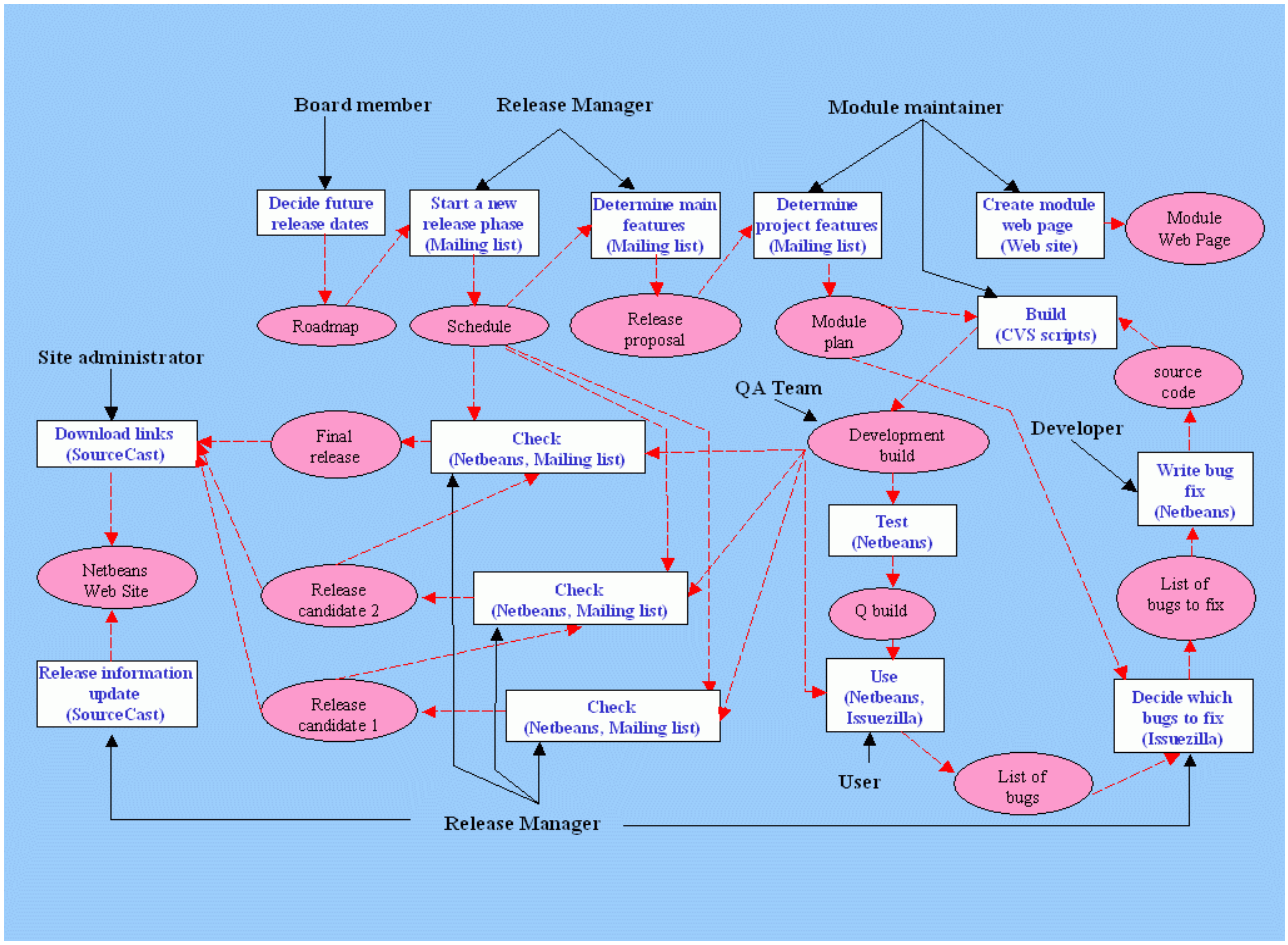

Figure 4. NetBeans Requirements and Release process flow graph [Oza, *et al*., 2002]

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 established software process meta-model [Mi and Scacchi 1996]. Using the PML grammar and software process meta-model, we created an ontology for process description with the Protégé-2000 modeling tool [Noy, *et al*. 2001]. A fragment of this software process ontology is shown in Figure 5. 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 6. Using this Protégé-based PML modeling environment, we construct a formal representation of the modeled process with two external instance representations. The first is an XML description according to the ontology while the second is a visual depiction generated by the Ontoviz [2003] and Graphviz [2003] ontology graph visualization tools. This visualization can be configured to highlight or hide relationships

between process instance actions on varying depths and with different focal points (e.g. tools, roles, resources), as suggested in Figure 7.

```
(defclass Action "A primitive step in a process."
   (is-a USER)
   (role concrete)
   (single-slot name_
;+       (comment "The name of the action.")
         (type STRING)
;+       (cardinality 1 1)
         (create-accessor read-write))
   (single-slot url
;+       (comment "A URL that optionally links to additional documentation
available remotely.")
         (type STRING)
;+       (cardinality 0 1)
         (create-accessor read-write))
   (single-slot next^action
;+       (comment "The next action to be performed.  This could be set to
nothing if, for example, an action is the last in a sequence.")
         (type INSTANCE)
;+       (allowed-classes Action)
;+       (cardinality 0 1)
         (create-accessor read-write))
   (multislot provides
;+       (comment "Resources that the action provides.")
         (type INSTANCE)
;+       (allowed-classes Resource)
         (create-accessor read-write))
   (single-slot script
;+       (comment "An automated script.")
         (type INSTANCE)
;+       (allowed-classes Script)
;+       (cardinality 0 1)
         (create-accessor read-write))
   (single-slot type
;+       (comment "The type of the action; either \"manual\" or
\"executable.\"")
         (type STRING)
```

Figure 5. A Protégé-2000 ontology definition for a PML action

```
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 6. A PML description of the testing sequence of the NetBeans release process

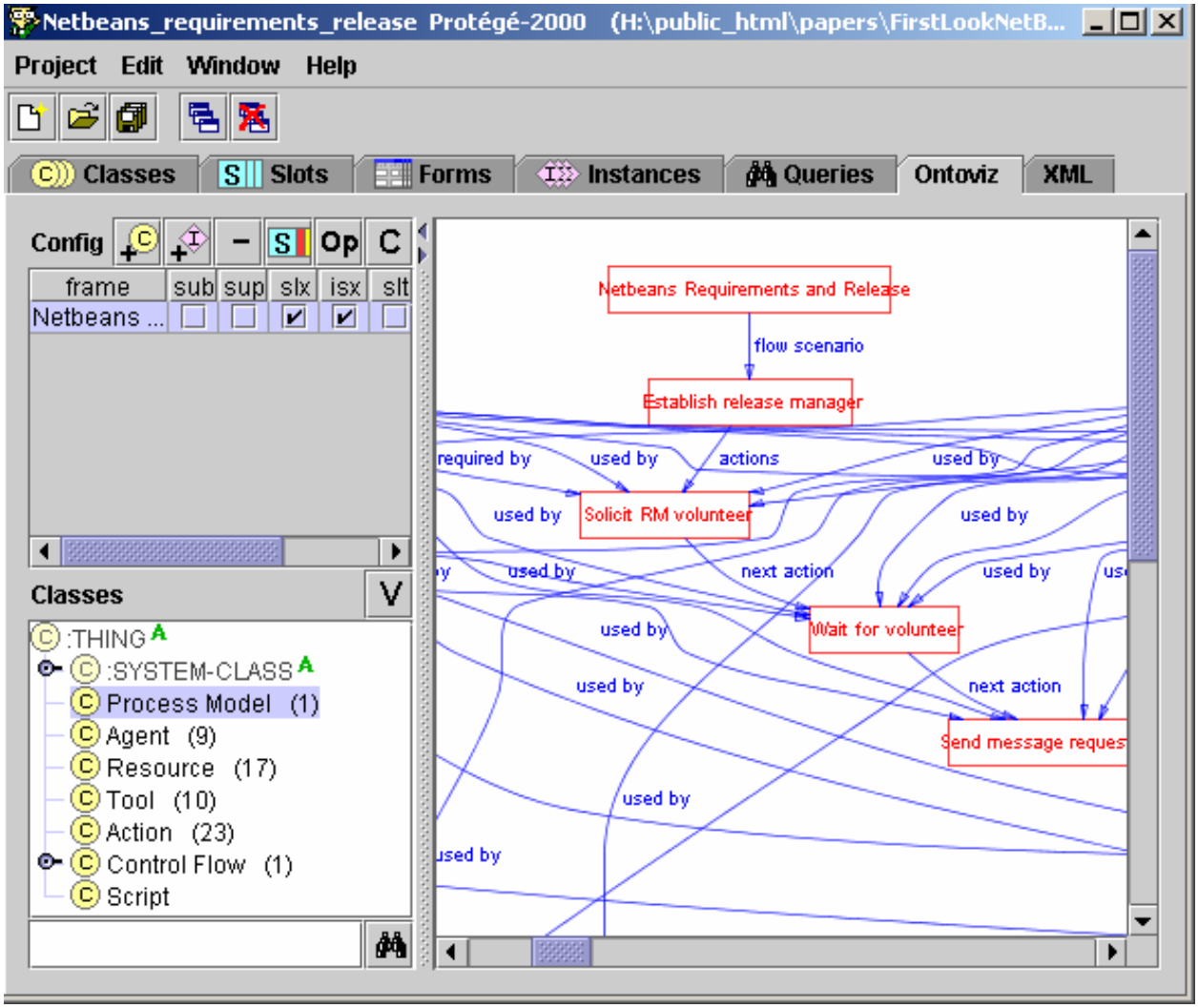

Figure 7. The Protégé-based PML modeling environment displaying a (partial) view of the NetBeans Requirements and Release process [Oza, *et al*. 2002].

Prototyping the process allows us to simulate process enactment instances by navigationally traversing a semantic hypertext representation of the process [Noll and Scacchi 2001]. One such hypertext page from this prototype appears in Figure 8. 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 reorganization and process redesign [Scacchi 2000]. Process prototyping also allows us 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 IssueZilla, but may also take responsibility for resolving it.

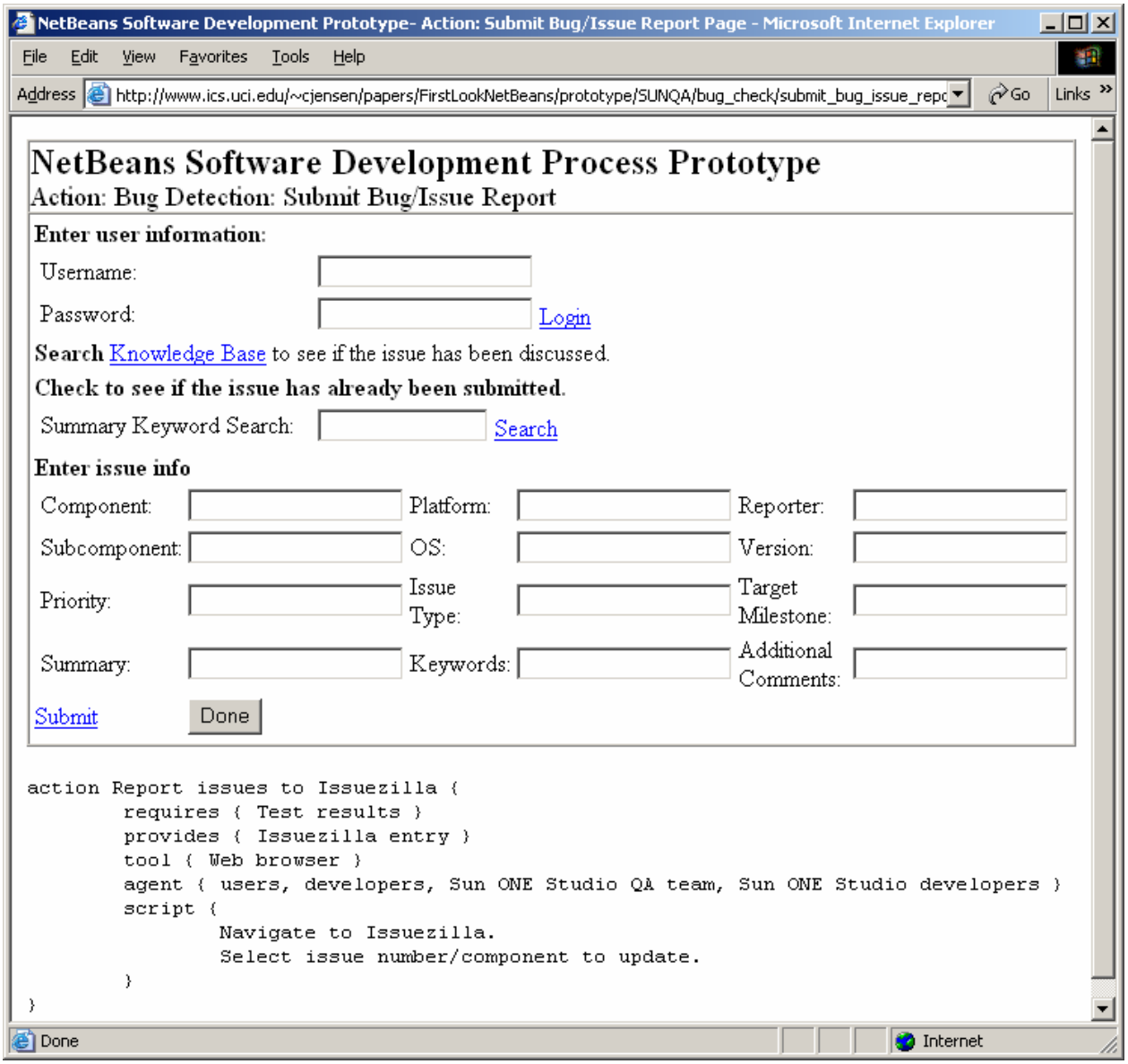

Figure 8. The NetBeans Requirements and Release Process Prototype

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. Prototypes are especially useful when implementing a process or altering a process in that these potential pitfalls may 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. This is a good indication that downstream activities may also uncover a way to better distribute the tasks and lighten 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 his peers in the matter, which is rarely the case in more 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. Discussion

Our experience with the NetBeans project has led us to several valuable lessons for determining the requirements for prototyping the design of an automated approach to process discovery and modeling in terms of information availability, granularity control, process validation, and accounting for evolution both of the process, itself, and the reference model used for its discovery.

6.1 Information Availability

First, there must be enough empirical evidence about the process in order to draw plausible conclusions about software process activities and behavior within an OSSD project. This information can be gathered from a number of sources. Combined with related studies of the Mozilla SQA and Release process [2003], and the release process of the Apache Software Foundation [2003], we have been able to identify several Web-based OSSD artifacts that encode process activities. Some of these are:

- 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
- Build 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 established preferred methods of interaction and communication, whether explicitly or implicitly. 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 a process enactment history, these changes need to be made persistent. 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].

6.2 Granularity Control

Wolf and Rosenblum [1993] lament that it is not possible to capture all process events through automated discovery. Indeed, in communities where members operate with a high degree of autonomy, there may be a lesser degree of communication of the details of those independently enacted processes, resulting in fewer information artifacts on the community Web. Such hidden processes may be difficult to detect. Yet, such a high fidelity process description may even be undesirable if the enactment of such independent processes proves highly variable across instances. Thus, low-fidelity process models that abstract away highly variable process enactment details may be a more fruitful direction for further study.

Given a sufficiently rich information space, the data collection stage of process discovery could be endless. As such, it is necessary to define the level of granularity desired. This can be controlled in several ways. Limiting the search depth and breadth of the Web space reduces the number of process artifacts that will be uncovered through mining to avoid over-refinement. Additionally, limiting the conditions for a match filters out peripheral and unrelated material from the results. At the same time, regulating the inferences applied to the results can eliminate the generality. This is an essential component of process discovery and requires some human sanity checking to ensure verify that the inferences made are rational and valid and not loosely grounded assertions. This can be monitored by adjusting the relationship search depth of the inference engine.

6.3 Process Validation

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]. Furthermore, 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, and refine 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[1].

6.4 Accounting for Evolution

Software processes are constantly evolving and adopting to changing circumstances, whether consciously prescribed by the project or through more subtle changes in daily work patterns. However, without *a priori* knowledge of process revisions, there is no way to determine whether the footprint was made before or after a change. Since the last probe for process data collection, the process may have evolved and taken a new direction. To discover a process, it is necessary that process footprints be placed at their correct location along the path of evolution in order to determine whether they are a part of the process as currently enacted, or as it was enacted at some point in the past. This means that projects interested in software process discovery must maintain the Web space and show proof of its maintenance.

6.5 Accuracy of the Reference Model

Since process discovery hinges heavily on an accurate model of what process clues may be found where and in which artifacts, it is essential that this reference model be maintained rigorously. Should the community adopt new means of collaboration or unconventional data representations, an automated process discovery mechanism would need to be given a means of locating and translating the data into the corresponding process fragment. Without reference model maintenance, the discovered processes will be incomplete at best, or outdated and misleading at worst.

7. 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. Work such as WebQuilt [Hong, *et al.* 2001] demonstrates that such information may be gathered in ways that are unobtrusive to process participants and that minimize the invasiveness found with many autonomous agent based event notification systems. These capabilities enable the capture of event streams from multiple iterations of a process or task through data mining and knowledge discovery techniques, so that we can identify and extract process fragments. In turn, we reconstitute process instances using xPADL, a process architecture description language based on XML and PML [Choi and Scacchi 2001, Noll and Scacchi 2001]. xPADL provides us with a formal description of each process instance which can be transformed (generalized) to realize 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.

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

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 automated process discovery and modeling. As demonstrated in the testing example, action sequences are constructed much like a jigsaw puzzle. We compile pieces of evidence 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. These pieces may be unearthed in ways that can be executed by software tools with little or no human assistance.

Our approach to discovery and model building 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 then transformed into a formal process representation language guided by a process meta-model ontology and support tools. We investigated techniques like Web site data mining and probabilistic relational modeling to craft the informal process representations, and it appears that in other domains of application, these discovery techniques have been successfully automated. Whether they can be used to automatically or semi-automatically generate use cases remains an open challenge, while construction of visual process representations like rich pictures and flow graphs appears manageable, once event flow and artifact dependencies can be recognized from process fragments. These informal representations together with a process meta-model then provide a scheme for constructing formal process descriptions, and this too appears to be a manageable task. Thus demonstration of a prototype implementation that integrates these capabilities and automated 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 goal in this research is to determine how to better support a more fully automated approach to process discovery and modeling. Our study provides a single example 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 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 are the best processes or development practices to follow. Thus discovery and explicit modeling of OSSD processes in forms that can be shared, reviewed, modified, and redistributed appears to be an important topic for further investigation, and this study represents an initial step in this direction.

8. Acknowledgements

The research described in this report is supported by grants from the National Science Foundation #IIS-0083075, #ITR-0205679 and #ITR-0205724. No endorsement implied. Contributors to work described in this paper include John Georgas, who tailored the Protégé-2000 tool for use in software process modeling. 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 also collaborators on the research project described in this paper.

9. References

Apache Software Foundation Web Site, 2003. http://www.apache.org

Ata, C., Gasca, V., Georgas, J., Lam, K., and Rousseau, M., 2002. *The Release Process of the Apache Software Foundation*, 2002. http://www.ics.uci.edu/~michele/SP/index.html

Cadez, I.V., Heckerman, D., Meek, C., Smyth, P., and White, S. 2000. Visualization of Navigation Patterns on a Web Site Using Model Based Clustering. *Proc. Knowledge Discovery and Data Mining,* 280-284.

Carder, B., Le, B., and Chen, Z. 2002. *Mozilla SQA and Release Process*,. http://www.ics.uci.edu/~acarder/225/index.html

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 Engiineering*, 754-757, Limerick, Ireland, June.

Choi, J.S. and Scacchi, W. 2001. Modeling and Simulating Software Acquisition Process Architectures. *Journal of Systems and Software,* 59(3) 343-354, 15 December 2001.

Cook, J. and Wolf, A.L. 1995. Automating Process Discovery through Event-Data Analysis. *Proc. 17th Intern. Conf. Software Engineering*, 373-386, Seattle, Washington, USA, April.

Cook, J. 1996. *Process Discovery and Validation through Event-Data Analysis.* Ph.D. dissertation. Computer Science Dept., University of Colorado.

Cooley, R., Mobasher, B., and Srivastava, J. 1997. Web Mining: Information and Pattern Discovery on the World Wide Web. *Proc. Ninth IEEE Intern. Conf. on Tools with Artificial Intelligence,* 558-567, November.

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

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.

Getoor, L., Friedman, N., Dzeroski, S., and Lavrac, N. 2001, Learning Probabilistic Models. *Relational Data Mining*. Springer-Verlag.

Graphviz, AT&T-Labs-Research. 2003. *Graphviz Open Source Graph Drawing Software*, http://www.research.att.com/sw/tools/graphviz/

Hilbert, D.M. and Redmiles, D.F. 1997, *An Approach to Large-Scale Collection of Application Usage Data Over the Internet.* University of California Irvine, Report UCI-ICS-97-40, Sept.

Hong, J.I., Heer, J., Waterson, S., and Landay, J.A. 2001, WebQuilt: A Proxy-based Approach to Remote Web Usability Testing*, ACM Trans. Information Systems,* 19(3), 263-285,

Koller, D. Probablistic Relational Models, 1999, ILP'99 Invited Talk, http://www.cs.bris.ac.uk/~ilp99/Invited/kollerHTML/

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

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

Mozilla Web Site, 2003. www.mozilla.org

NetBeans Contribute to The Community Web Page, 2003. http://www.netbeans.org/about/community/contribute.htm

NetBeans Defects in Graphics Page, 2003. http://qa.netbeans.org/bugzzilla/graphs/summary.html

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

NetBeans XTest Results, 2003. http://www.netbeans.org/download/xtest-results/index.html

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.

Ontoviz Web Site, 2003. http://protege.stanford.edu/plugins/ontoviz/ontoviz.html

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/

Protégé Web site, 2003. http://protege.stanford.edu/

Phalp, K. and Cox, K. 2003. Using Enactable Models to Enhance Use Case Descriptions, *Proc. ProSim'03 Workshop,* Portland, OR, May 2003.

Petersen, J.L. 1997. Petri Nets*, ACM Computing Surveys,* 9(3), September.

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.

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.