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

Abstract

Process discovery has been shown to be challenging 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 by having a human act as an "intelligent spider" searching the Web space for evidence of process activities and reconstructing process fragments based on the clues discovered. The purpose of such an approach is to help reveal the details of our manual process discovery approach. In turn, such knowledge can then be employed to determine the requirements and design of 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

Introduction

The goal of our work is to develop new techniques for discovering, modeling, analyzing, and simulating software development processes based on information, events, and contexts that can be observed through public information sources on the Web. Our problem domain examines processes in open source software development (OSSD) projects, such as those associated with the Apache Web server, Mozilla Web browser, and interactive development environments like NetBeans and Eclipse. We demonstrate the feasibility to automate the discovery of software process workflows via manual search and analysis methods in projects like NetBeans [24] by analyzing the content of their Web information spaces, including informal task prescriptions, community and information structure and work roles, project and product histories, and communications patterns among project participants. All of this information is publicly available on the Web, while corresponding events that denote updates to these sources is also publicly accessible. Though this 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 and 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 (source code files, messages posted on public discussion forums, Web pages, etc.), artifact update events (version release announcements, Web page updates, message postings, etc.) and work contexts (roadmap for software version releases, Web site architecture, communications systems in use (email, forums, instant messaging, etc.)) that can be observed, detected, or extracted through 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. A process meta-model then provides support for how to associate these data with software processes and process models. This is the same challenge that prospective developers or corporate sponsors who want to join a given OSSD project face. 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 found in many OSSD projects. The capabilities and tools include those for Internet-based event notification, Web-based data mining and knowledge discovery, and previous process discovery tools.

Related Work

Event notification systems have been used in many contexts [18, 28]. 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 her 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 [13, 14] 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 [25] with a wide variety of activity and state-chart diagrams in between, though it appears these representations still suffer from a lack of scalability when applied to a process of any reasonable complexity.

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 he disregarded the results as insignificant in comparison to those obtained without the extension, Cadez [9] points out that joint probabilities tend to be more informational.

Garg [16] 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) [20]. Cooley, *et al*. [15] are among those examining tools and methods for data mining on the Web and the unique challenges that apply to analysis done on remote objects on a server. This is the case with process discovery of open source projects. They propose an architecture for Web usage mining and a taxonomy of efforts 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 [20]. PRM give us a means of representing relationships between tasks, actors, tools, and resources that we may be able to easily characterize in an informal model of process and context such as a Rich Picture [21], but have been difficult to show in a singular view of a formal model [17] 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 [11] and PML [22] argues for graphical views or visual navigation 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 visual representations. Thus, we employ this practice here.

Problem Domain

Our work lies in the domain of OSSD projects. Specifically, we are interested in discovering, modeling, simulation, and enactment of software development processes in large scale online OSSD projects. Such projects are often globally distributed efforts sometimes involving hundreds of developers collaborating on large products without meeting face-to-face, and often without performing modern methods of software engineering [27]. Past approaches have shown process discovery to be difficult, yielding limited results, as already noted. 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 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 of project-related software development artifacts is organized
- Site contents -- what types of artifacts exist and what information they contain
- Usage patterns -- user interaction and content update patterns within the project Web.

How the project organizes its information may indicate what types of artifacts they generate. For example, a directory such as "x-test-results" can be examined to determine whether there is evidence that some sort of testing has been conducted, whereas timestamps associates with updates provide a sense of recent activity and information sharing. Similarly, when new branches in the site tree 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 names "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, at least 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.

Approach

As a prelude to developing an automated process discovery solution, we first have simulated the machine knowledge enabled approach with another type of intelligent agent, namely people trained in software processes and software process modeling techniques. 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. For this task, we examined a selected process in the NetBeans project developing an open source IDE using Java technology [1]. The "requirements and release" process was chosen because the activities it is composed of have a 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.

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 drew up a reference model 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 and the nature of its open source status. This project history is a basis for understanding current development practices. But also, it details ways of becoming involved in development and the community at large [4]. The modes of contribution can be used to construct an initial set of Use cases of project participation. The site map also shows a page dedicated to project governance buried 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 encodes 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 process model such as a Rich Picture [21]. The Rich Picture, shown in Figure 1, identifies developer roles, tools, and artifacts of development and their interaction, as well as hyperlinks (indicated as underlined phrases) to the corresponding Use cases. 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.

By text-searching the "defect" page [5], 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[6], 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 2. 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.

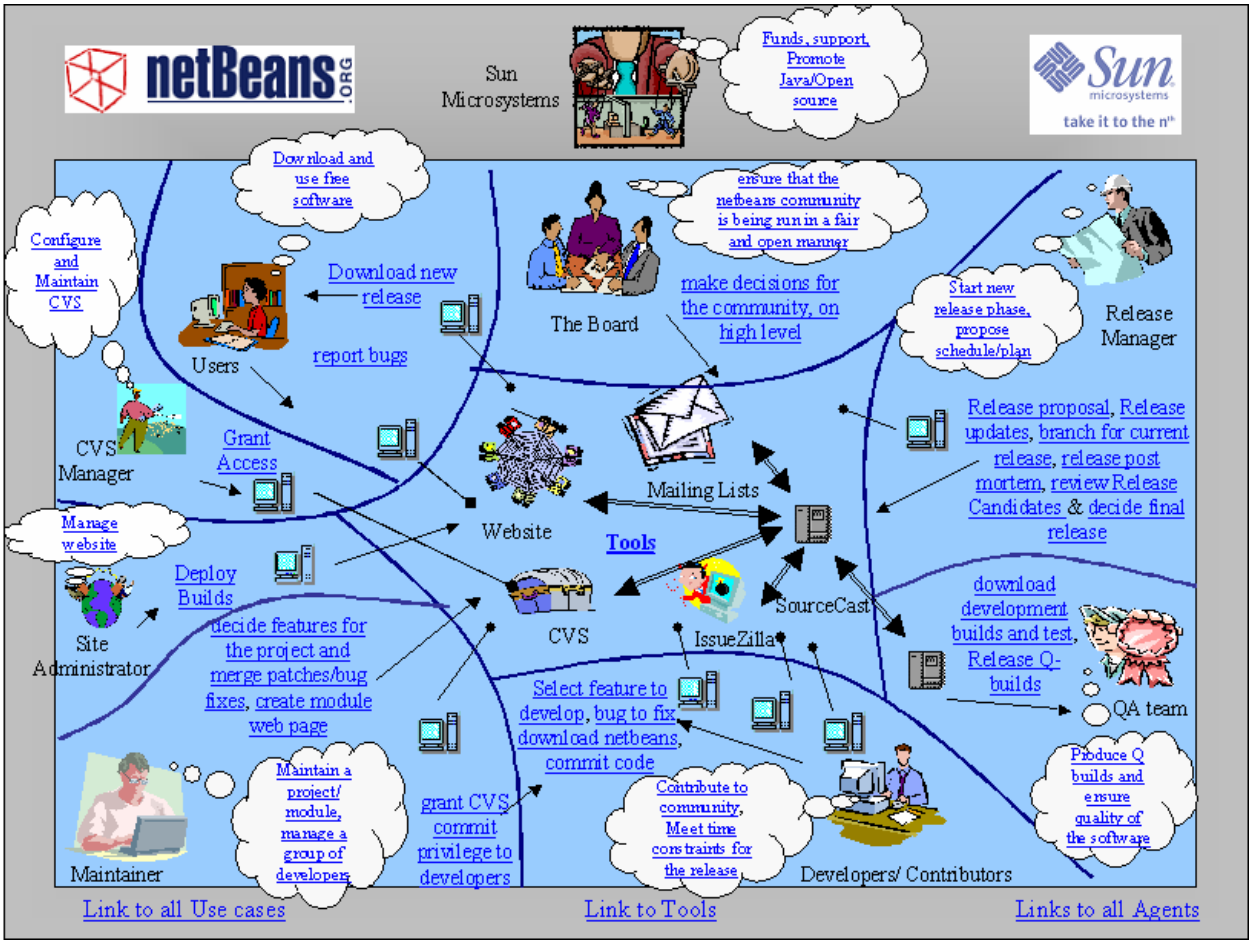

Figure 1. A hyperlinked Rich Picture of the NetBeans Requirements and Release process [24]

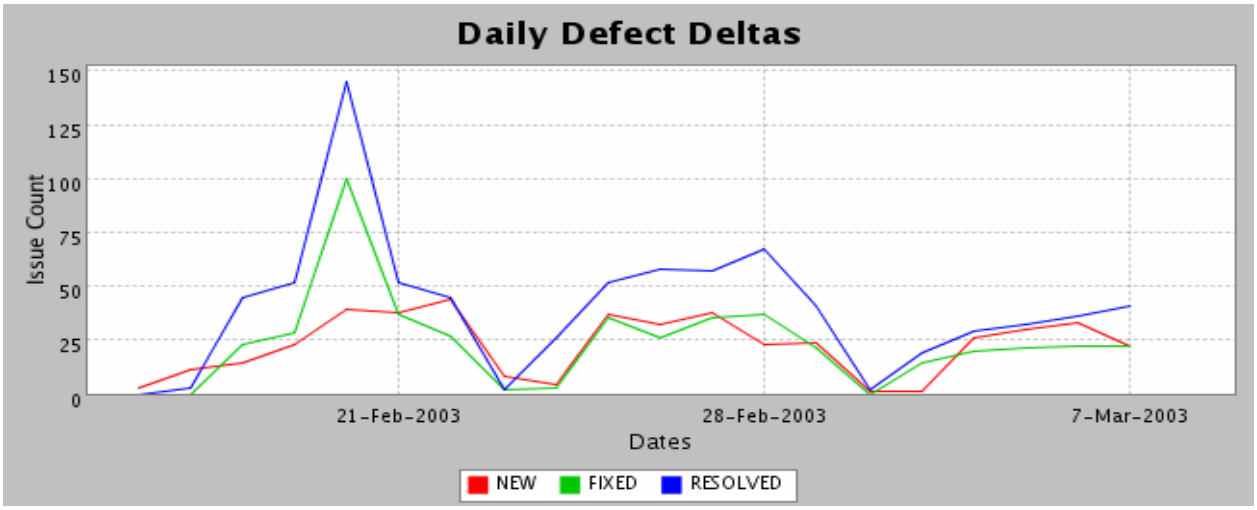

Figure 2. NetBeans Daily Defect Count Chart

The 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. 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 macro-view of the requirements and release process is shown in Figure 3.

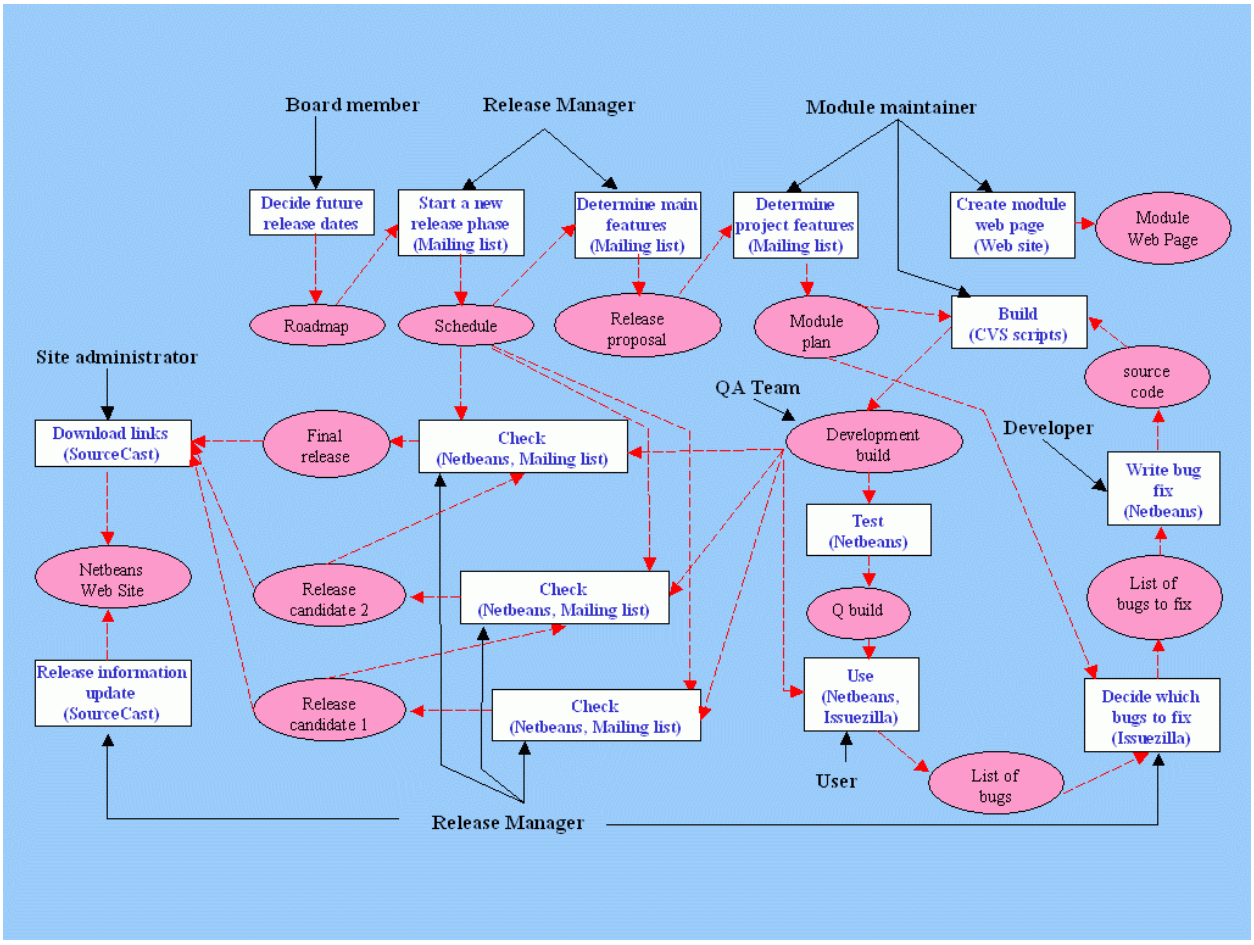

Figure 3. NetBeans Requirements and Release process diagram [24]

These process fragments can now be assembled into a formal PML description of the selected processes [22]. Using the PML grammar and meta-model as a foundation [22], we created an ontology for process description with the Protégé-2000 modeling tool [23]. The PML model builds from the Use cases depicted in the Rich Picture, 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.

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 [3] and Graphviz [7] ontology graph visualization tools. This visualization can be configured to incorporate relationships between process instance actions on varying depths and with different focal points (e.g. tools, roles, resources), as suggested in Figure 5.

```
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 [24]

Prototyping the process has allowed us to simulate an activity instance of it without actually enacting it. In doing so, we have been able to detect processes that may be unduly lengthy, which may serve as good candidates for downstream activities such as reorganization and process redesign [26]. It 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 an issue and resolve it.

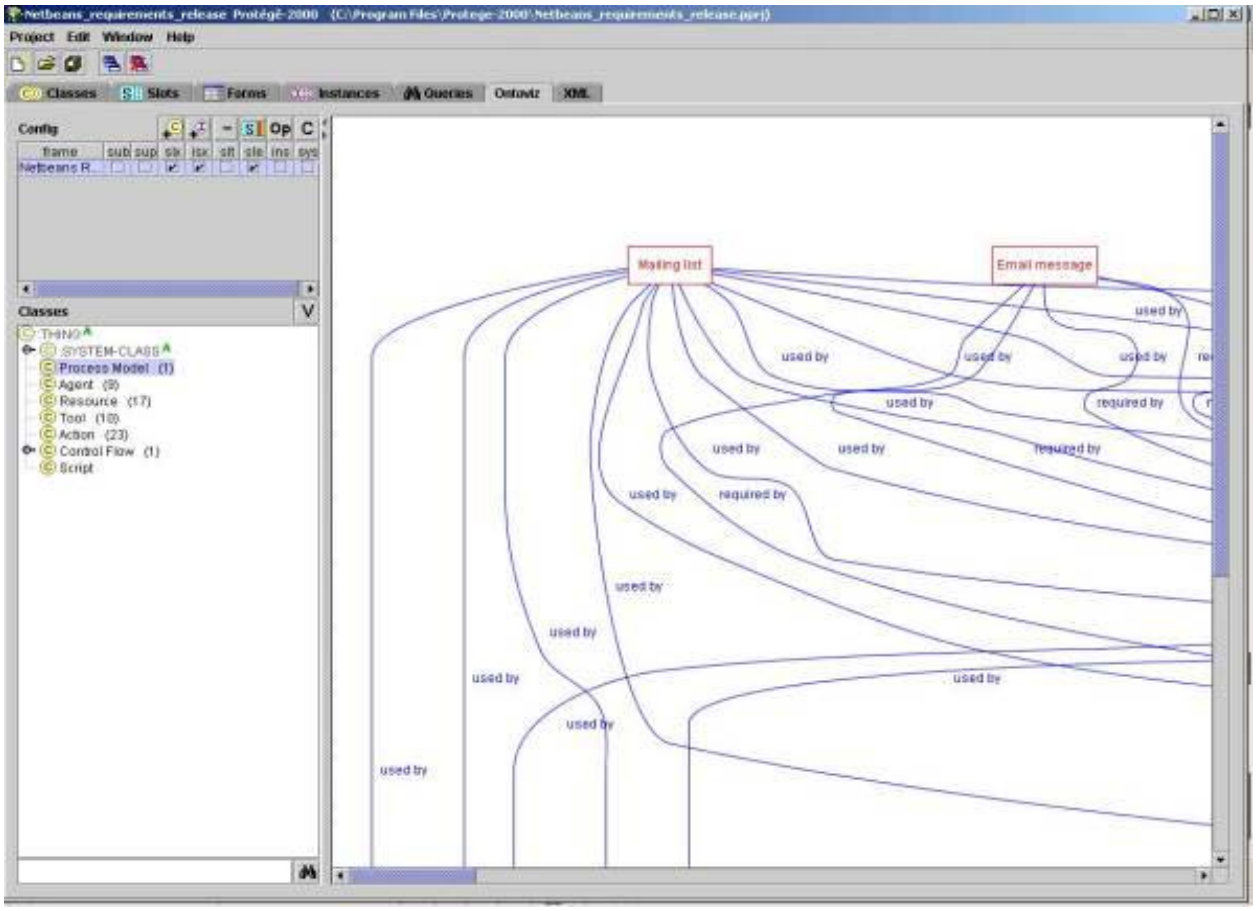

Figure 5. The Protégé-based PML modeling environment displaying a (partial) view of the NetBeans Requirements and Release process [24].

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 her load.

```
(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)
;+          (cardinality 1 1)
            (create-accessor read-write)))
```

Figure 6. A Protégé-2000 ontology definition for a PML action [24]

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 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 her own software release build.

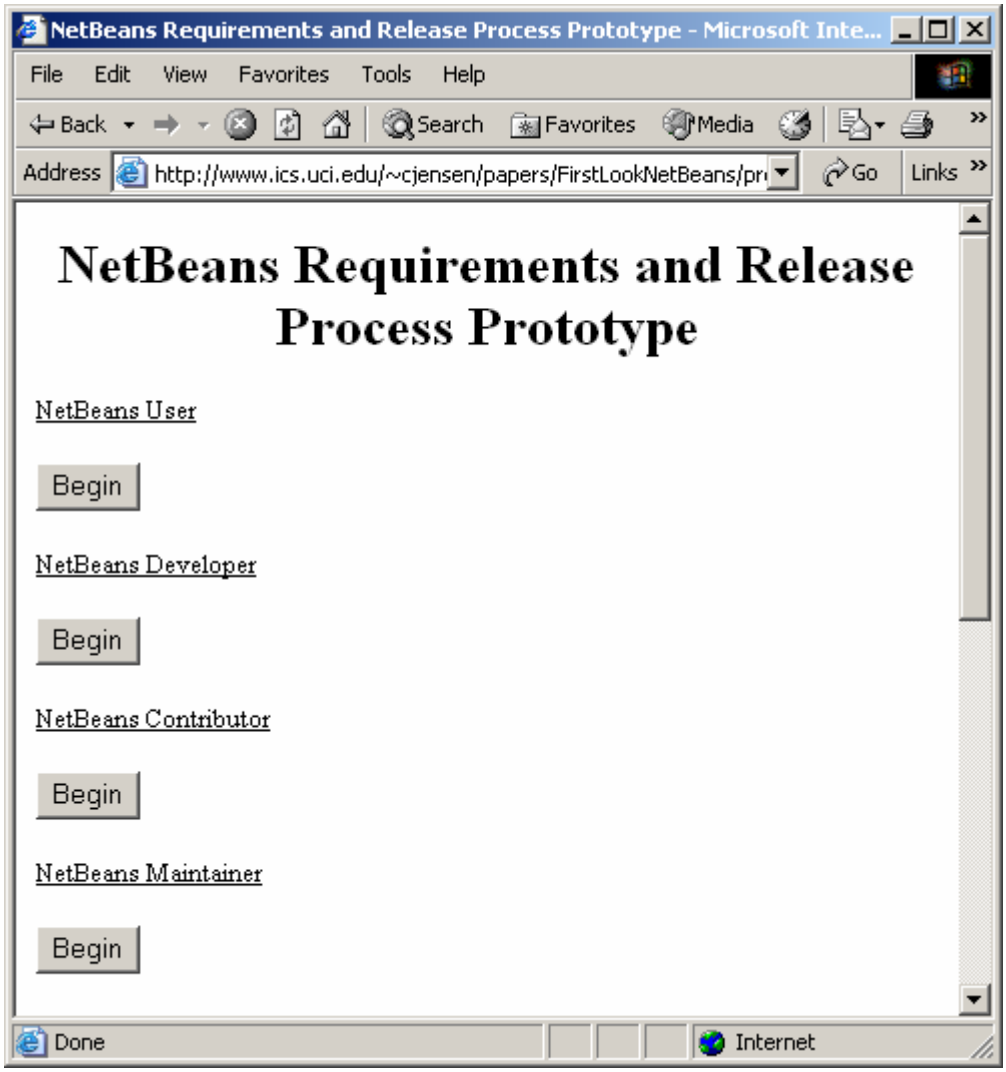

Figure 7. The NetBeans Requirements and Release Process Prototype [cf. 12].

Discussion

Our experience with the NetBeans project has led us to several lessons for prototyping an automated approach to process discovery and modeling in terms of information availability, granularity control, process validation, and accounting for evolution.

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 independent studies of the Mozilla SQA and Release process [10] and the release process of the Apache Software

Foundation [8], we have been able to identify several process encoding Web artifacts. Some of these are:

- Web Pages
- Asynchronous communications via threaded email discussion lists
- Software product source code directories as Web pages
- OSS development tools
- OSS development resources, including other software development artifacts
- OSS extension mechanisms (e.g., scripting language usages)

Each project has established preferred methods of interaction and communication, whether explicitly or implicitly. These collaboration modes yield a high amount of process evidence, as well as a large degree of unrelated data. 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.

Granularity Control

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.

Process Validation

Since their success relies heavily on public participation, OSSD projects often have informal descriptions of ways members can participate and prescriptions of the community process. Although 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 make it is possible to perform validation analysis.

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.

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, we may be able to observe, derive, or otherwise discover process activities. Work such as WebQuilt [19] 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 [19]. Using data mining and knowledge discovery techniques, we can identify and extract process fragments. In turn, we reconstitute process instances using xPADL, a process architecture [12] description language based on XML and PML [22]. xPADL provides us with a formal description of each process instance which can be transformed (generalized) to realize an enactable model of the process in question, that can then be employed with a process simulator [12, 22].

Our experience with 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 events of the larger picture, 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. Once assembled the process may then be verified, modeled, prototyped or simulated, and then enacted to reveal locations where the process has broken down, and how the process may recover from failure, and redesigned to achieve a more optimal use of human and computational resources. Our future work aims to realize a more fully automated approach to process discovery and tools and techniques for its recovery and redesign.

Our purpose in applying probabilistic techniques to modeling process enactment instances is an acknowledgement that there is often variation within the instantiation of recurring processes activities or sub-processes, whether due to constraints of a given project or changing preference of a selection of tools. Both such actions may impact not only the way an action is performed, but the types of actions that are performed. These dependencies and their consequences are taken care of by accounting for, not only the probability of enacting an activity given the action completed immediately prior, but also then entire chain of actions leading up to it. Eliminating chains with an execution rate below a given threshold gives us a basis for simulating action sequences that reflect this variability.

By leveraging existing OSS object modeling tools such as Protégé-2000 [2, 23], we were able to extend PML using an XML framework to create an extensible process architecture language (xPADL) and develop a modeling and prototyping environment for xPADL that can produce xPADL, XML or SQL as its outputs. We then use these tools to model software

processes observed in a sample of OSSD projects, including the NetBeans IDE project [24]. Subsequently, outputs such as these can be fed into existing process simulator tools we have previously investigated [12, 22].

At this stage, we are now investigating how best to apply probabilistic data mining and knowledge discovery techniques to automatically construct software process models from process enactment fragments. We also seek to automatically detect events or updates that may reveal where the current process enactment has failed, or where the current process model could be improved or redesigned [26]. Furthermore, we seek to identify which OSSD processes are amenable to automated discovery, modeling, simulation, and improvement, and which OSSD process are not so amenable. We are also examining how XML data binding tools like Protégé-2000 can provide a straightforward segue from our xPADL process descriptions to existing or new process simulation capabilities. Finally, our goal is to develop and refine our process discovery, modeling, and simulation tools and techniques as open source software that can be shared, employed, reused, modified and redistributed to other OSSD projects, in a manner consistent with the emerging vision for free/open source software projects, such as NetBeans.

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 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; Margaret Elliott, Mark Bergman, and Xiaobin Li at the UCI Institute for Software Research; and Julia Watson at The Ohio State University are also collaborators on the research project described in this paper.

References

[1] *NetBeans Open Source Project*. <http://www.netbeans.org>.

[2] *Protégé Web site*. <http://protege.stanford.edu/>.

[3] *Ontoviz Web Site*. <http://protege.stanford.edu/plugins/ontoviz/ontoviz.html>.

[4] *NetBeans Contribute to The Community Web Page*. <http://www.netbeans.org/about/community/contribute.htm>.

[5] *NetBeans Defects in Graphics Page*. <http://qa.netbeans.org/bugzzilla/graphs/summary.html>.

[6] *NetBeans XTest Results*. <http://www.netbeans.org/download/xtest-results/index.html>.

[7] AT&T-Labs-Research. *Graphviz Open Source Graph Drawing Software*. <http://www.research.att.com/sw/tools/graphviz/>.

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

[9] Cadez, I.V., Heckerman, D., Meek, C., Smyth, P., and White, S. Visualization of Navigation Patterns on a Web Site Using Model Based Clustering. In *Proc KDD.* 280-284, 2000.

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

[11] Cass, A.G., Lerner, B.S., McCall, E.K., Osterweil, L.J., Sutton, J., S.M., and A., W. Little JIL/Juliette: A process definition language and interpreter. In *Proc. 22nd International Conference on Software Engiineering.* 754-757, Limerick, Ireland, June, 2000.

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

[13] Cook and Wolf, A.L. Automating Process Discovery through Event-Data Analysis. In *Proceedings of the 17th International Conference on Software Engineering.* 373-386, Seattle, Washington, USA, April, 1995.

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

[15] Cooley, R., Mobasher, B., and Srivastava, J. Web Mining: Information and Pattern Discovery on the World Wide Web. In *Proceedings of the Ninth IEEE International Conference on Tools with Artificial Intelligence.* 558-567, November, 1997.

[16] Garg, P.K. and Bhansali, S. Process programming by hindsight. In *Proc. 14th International Conference on Software Engineering.* 280-293, 1992. <http://doi.acm.org/10.1145/143062.143128>.

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

[18] Hilbert, D.M. and Redmiles, D.F. *An Approach to Large-Scale Collection of Application Usage Data Over the Internet.* Department of Information and Computer Science, University of California, Irvine, Report UCI-ICS-97-40, September, 1997.

[19] Hong, J.I., Heer, J., Waterson, S., and Landay, J.A. WebQuilt: A Proxy-based Approach to Remote Web Usability Testing. *To appear in ACM Transactions on Information Systems.*

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

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

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

[23] Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R.W., and Musen, M.A. Creating Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems.* March/April, 2001.

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

[25] Peterson, J.L. Petri Nets. *ACM Computing Surveys (CSUR).* 9(3), September, 1997.

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

[27] Scacchi, W. Software Development Practices in Open Software Development Projects: A Comparative Study. In *Proceedings of the First Workshop on Open Source Software Engineering.* May, 2001. <http://opensource.ucc.ie/icse2001/scacchi.pdf>.

[28] Wolf, A.L. and Rosenblum, D.S. A Study in Software Process Data Capture and Analysis. In *Proceedings of the Second International Conference on the Software Process.* 115-124, IEEE Computer Society. February, 1993.