Experience in Discovering, Modeling, and Reenacting Open Source Software Development Processes

Chris Jensen[1] and Walt Scacchi[1]
[1]Institute for Software Research, University of California, Irvine
Irvine, CA USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

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

1 Introduction

The goal of our work is to develop new techniques for discovering, modeling, analyzing, and simulating software development processes based on information, artifacts, events, and contexts that can be observed through public information sources on the Web. Our problem domain examines processes in large, globally dispersed open source software development (OSSD) projects, such as those associated with the Apache Web server [16], Mozilla web browser [24], GNOME [9], and integrated development environments like NetBeans [18] and Eclipse [4]. The challenge we face is similar to what prospective developers or corporate sponsors who want to join a given OSSD project face in trying to understand how software development processes and activities are accomplished. As such, our efforts should yield practical results.

OSSD projects do not typically employ or provide explicit process models, prescriptions, or schemes other than what may be implicit in the use of certain OSSD tools for version control and source code compilation. In contrast, we seek to demonstrate the feasibility of automating the discovery of software process workflows via manual search and analysis methods in projects like NetBeans by analyzing the content, structure, update and usage patterns of their Web information spaces. These spaces include process enactment information such as informal task prescriptions, community and information structure and work roles, project and product development histories, electronic messages and communications patterns among project participants ([5], [26], [29]). Likewise, corresponding events that denote updates to these sources and other project repositories are also publicly accessible. Though such ethnographic discovery approaches net a wealth of information with which to model, simulate, and analyze OSSD processes, they are limited by a lack of scalability when applied to the study of multiple OSSD development projects (cf. [13]). Subsequently, it suggests the need for a more automated approach to that can facilitate process discovery.

In our approach, we identify the kinds of OSSD artifacts (e.g. source code files, messages posted on public discussion forums, Web pages, etc.), artifact update events (e.g. version release announcements, Web page updates, message postings, etc.), and work contexts (e.g. roadmap for upcoming software 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. That is the focus of our effort.

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 [15]. 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 text data mining and knowledge discovery, and previous results from process discovery studies. However, in this study, we focus on identifying the foundations for discovering, modeling, and re-enacting OSSD processes that can be found in a large, global OSSD project using a variety of techniques and tools.

2 Related Work

Event notification systems have been used in many contexts, including process discovery and analysis ([3], [30]). 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. This includes the need to install and integrate remote data collection mechanisms with local or project-specific software development tools, and it is unclear who would take on such effort within an existing OSSD project.

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 (object and tool invocations) [8]. They advise that rather than seeking to discover the entire development process from enactment instances, to instead focus on creating partial process specifications that may overlap with one another. This also reflects variability in software process enactment instantiation across iterations. This imparts additional inconvenience on project developers, and relies on her/his willingness to use the particular tools that monitor and analyze command shell events (which can become intractable when a developer uses tools or repository services from remote networked systems). 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 adoption of such event-capture technology.

Cook and Wolf [3] utilize algorithmic and statistical inference techniques to model processes where the goal was to create a single, monolithic finite state machine (FSM) representation of the process. However, it is not entirely clear that a single FSM is appropriate for modeling complex processes. Similarly, other FSM-related process representation schemes such as Petri-Net based FUNSOFT [6] offered a wide variety of activity and state-chart diagrams. It appears however that these representations may lack scalability when applied to a process situated within a global organizational context involving multiple tools, diverse artifact types, and multiple development roles across multiple networked sites of reasonable complexity, which is typical of large OSSD projects (cf. [9]).

Last, while process research has yielded many alternative views of software process models, none has proven decisive or clearly superior. Nonetheless, contemporary research in software process technology, such as Lil Jil [2], [21] and PML [19] 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 [28]. We employ this practice here.

3 Problem Domain

We are interested in discovering, modeling, simulating, and re-enacting software development processes in large, Web-based OSSD projects. Such projects are often globally distributed efforts sometimes involving tens, 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 ([26], [27]). 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, nor do they simply apply prior approaches. Instead, we capitalize on contextual aids offered by the domain. Some of these include:

- Web pages, including project status reports and task assignments, may be viewed and classified (informally) as object types.
- Asynchronous communications among project participants posted in threaded email discussion lists, which address process activities indicated by process identifier keywords (e.g., design, release, testing, etc.)
- Transcripts of synchronous communication via Internet chat (cf. [5]).

- Software problem/bug and issue reports, which reveal information on software bug reporting and maintenance/repair processes
- Testing scripts and results, which highlight project-based software testing practices
- Community newsletters, which highlight project milestone events (e.g., system releases, turnover of core developers in the projects)
- Web accessible software product source code directories and repositories, which carry timestamps and other identifiers indicating when source code objects were checked in/out, and versioning information.
- Software system builds (executable binaries) and distribution packages, which are constructed and released on a periodic basis (daily, candidate (alpha, beta), and final release (distribution version)
- OSS development tools in use in an OSSD project (e.g., concurrent version system (CVS), GNU compiler collection (gcc), bug reporting (bugzilla) (cf. [10])

- OSS development resources, including other software development artifacts and process fragment descriptions (e.g., How-To guides, lists of frequently asked questions (FAQs), etc.) [26]

Each OSSD project has locally established methods of interaction, communication, leadership, and control [14], whether explicit or implicit ([26], [27]). These collaboration modes yield a high amount of empirically observable process evidence, as well as a large degree of unrelated data. However, information spaces are also dynamic. New artifacts are added, while existing ones are updated, removed, renamed and relocated, else left to become outdated. Artifact or object contents change, and project Web sites get restructured. In order to capture the history of process evolution, these changes need to be made persistent and shared with new OSSD project members. While code repositories and project email discussion archives have achieved widespread use, it is less common for other artifacts, such as instant messaging and chat transcripts, to be archived in a publicly available venue. Nonetheless, when discovering a process in progress, changes can de detected through comparison of artifacts at different time slices during the development lifecycle. At times, the detail of the changes is beneficial, and at other times, simply knowing what has changed and when is all that is important to determining the order (or control flow sequence) of process events or activity. To be successful, tools for automated process discovery must be able to efficiently access, collect, and analyze the data including areas of the project Web space such as public email/mailing list message boards, Web page updates, notifications of software builds/releases, and software bug archives in terms of changes to the OSS information space [26], [27].

To prove the viability of our process discovery approach, we demonstrate it with a case study. For this task, we examine a selected process in the NetBeans project, which is developing an open source IDE using Java technology[1]. The "requirements

[1] The NetBeans project was started in 1996, and SUN Microsystems began project sponsorship in 1998. At present, more than 60 companies are participating in the

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 be extracted using automated technologies. The process was discovered, modeled informally and formally, then prototyped for analysis and reenactment. The full results of our initial case study may be found elsewhere [22]. The discussion of our process discovery and modeling methods and results follows next.

4 Process Discovery and Modeling

Discovery of open source software processes relies on several data models. Firstly, we need to determine what aspects of the process we wish to discover, defined by our process meta-model. This meta-model is neither specific to our domain (OSSD processes), nor software processes. To capture OSSD software processes, we need a means of setting the problem domain within the terms of the meta-model. Such is the task of the reference model. Once this is done, we may begin looking for instances of processes within a corpus, in this case the OSSD project Web repository.

The meta-model we use is that of Mi and Scacchi [15]. It provides us with a vocabulary to describe the processes we examine. Our meta-model defines processes hierarchically: processes are composed of tasks (sets of related actions) and atomic actions. The hierarchy may be further divided (e.g. sub-processes, subtasks, and so forth) to achieve an arbitrary degree of decomposition. Each activity is defined in terms of process entities: agents that participate in the process activity, tools used by those agents in the performance of the activity, and resources that are the product of and are consumed by performance of the activity (see Figure 1). We find these to be a minimal set of entities necessary to describe a process. This meta-model is augmented with control-flow grammar in the process markup language (PML) [19] we use to formally represent software processes to show the order in which activities and instantiated.

Unlike Cook and Wolf's approach, we apply *a priori* knowledge of software development for discovering processes. Accordingly, we use a reference model [11] to help identify possible indicators (e.g., developer roles, probable tool types, input and output objects) that a given activity has occurred. We do this by creating a taxonomy of the process entities within the problem domain. Thus, we enumerate the types of tools (and resources, activities, and agents-roles) we expect to find

project through their developers. In 2004, the project passed the threshold of more than 100K developers contributing to the project.

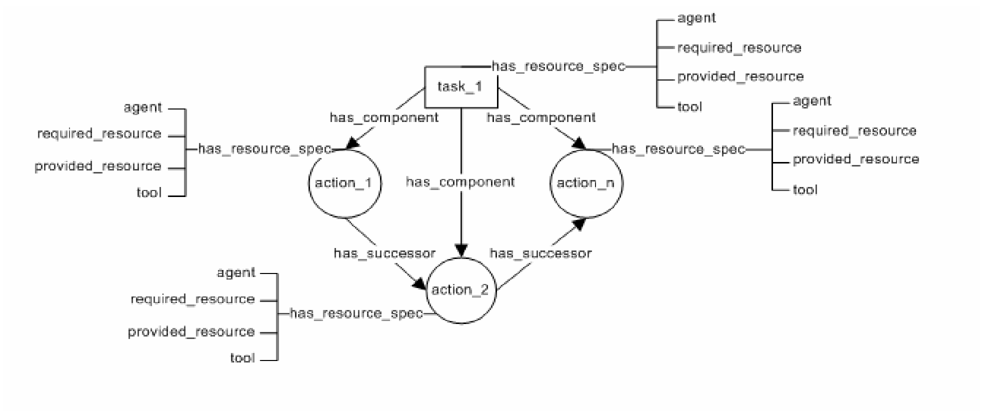


Figure 1. Software process meta-model (cf. [15]).

referenced in the project corpus (e.g. "email client") as well as instances of those tools (e.g. "Mozilla Thunderbird"). 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. The instances are necessary for discovering process entities used within the corpus while types and genericity/hierarchy aid us in abstracting the instance data into a more general process model spanning multiple enactments.

Although we would like to achieve some degree of automation in discovering open source software development processes, it is unreasonable to assume that a complete solution is possible due to the heterogeneity of data available within and across project information corpora. Instead, we seek to automate as much as possible in order to ease the effort inherent to the task. To this end, our methodology incorporates general information gathering independent of document type, augmented by some analysis techniques specific to the type and structure of the data available. Thus we automate the tasks that are easy to automate and provide value to make the effort worthwhile. And, we do manually tasks for which automation is either too difficult or does not provide payoff to validate effort required.

We use indexing at the core to do much of the legwork of general information gathering. We use this index to identify actions, tools, resources, and agents within artifacts in the corpus. These are correlated across artifacts according to usage and update information available. While we are able to tune the reference model to contain instance values of actions, resources, and tools (e.g. "submit defect report", "x-test-results", and "Issuezilla," respectively), identifying process agents by proper names a priori is not possible. Such identification requires document specific analysis techniques. These include parsers for extracting names, user handles, and email addresses from threaded mailing lists, chat logs, defect reports, and versioning repositories. Once extracted, they are looked up in the index and added to the action-tool-resource tuples already identified. Such document specific analysis may also be used to uncover heretofore-unknown instances of other process entities (i.e. actions, resources, and tools) in similar fashion and is essential to obtaining accurate timestamps in documents that aggregate multiple software artifacts (e.g. threaded mailing lists containing multiple messages within a single system file).

Document specific analysis provides rich results with a cost. There are many types of data and standards for document structure even for a single type of data and these vary highly across OSSD projects. As a result, a broad array of tools specifically tuned for each project corpus is required to obtain rich results. Such an array is painstaking to develop, although available off-the-shelf partial solutions ease this burden. Further, integrating large result sets from multiple data sources into a single process model of any degree of formality is a complex task in itself. Our reference model can suggest process entity tuples that are related and the temporal information we are able to extract provides a timeline of activities. However learning activity control flow and asserting an activity hierarchy remain somewhat an art as opposed to a science.

The discovery of processes within a specific OSSD project begins with a cursory examination of the project Web space in order to ascertain what types of information are available and where that information might be located within the project's Web site. Structure and content of the project Web space give us an idea of what happened in terms of process actions, agents, tools, and resources, whereas artifact usage and update patterns tell us when process activities happened as noted above.

To situate the process within its organizational context, we look for modes of contribution within the development process. The modes of contribution (development roles) can be used to construct an initial set of activity scenarios, which can be described as *use cases* for project or process participation.

Though best known as a tenet of UML, use cases can serve as a notation to model scenarios of activities performed by actors in some role that use one or more tools to manipulate artifacts associated with an enterprise process or activity within it ([7], [29]). 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 hypermedia*, an interactive semi-structured

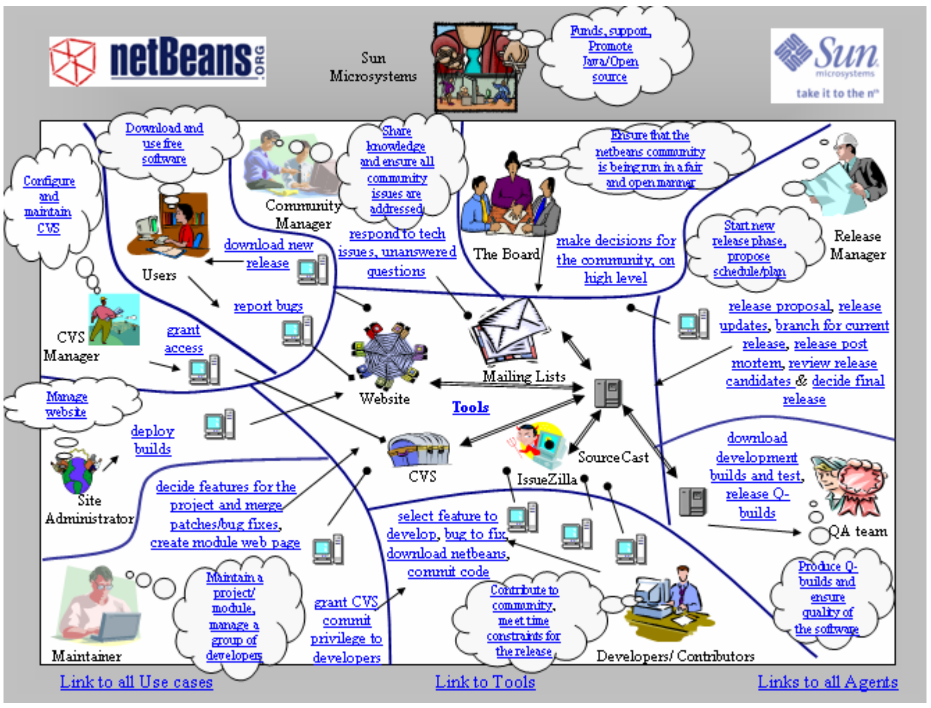
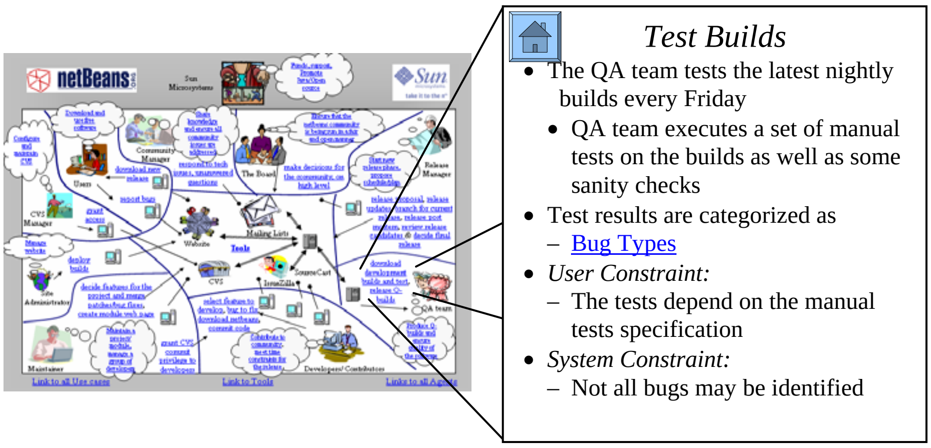

netBeans.org
Sun Microsystems
Funds, support, Promote Java/Open source
Sun microsystems take it to the n^th
Download and use free software
Configure and maintain CVS
Community Manager
Share knowledge and ensure all community issues are addressed
Ensure that the netbeans community is being run in a fair and open manner
Start new release phase, propose schedule/plan
Release Manager
Users
download new release
respond to tech issues, unanswered questions
The Board
make decisions for the community, on high level
report bugs
CVS Manager
grant access
Website
Mailing Lists
release proposal, release updates, branch for current release, release post mortem, review release candidates & decide final release
Manage website
Tools
deploy builds
CVS
IssueZilla
SourceCast
download development builds and test, release Q-builds
Site Administrator
decide features for the project and merge patches/bug fixes, create module web page
select feature to develop, bug to fix, download netbeans, commit code
QA team
Maintain a project/ module, manage a group of developers
grant CVS commit privilege to developers
Contribute to community, meet time constraints for the release
Produce Q-builds and ensure quality of the software
Maintainer
Developers/ Contributors
Link to all Use cases
Link to Tools
Links to all Agents

Figure 2. A hyperlinked rich hypermedia of the NetBeans requirements and release process [22]



Test Builds

- The QA team tests the latest nightly builds every Friday
  - QA team executes a set of manual tests on the builds as well as some sanity checks
- Test results are categorized as
  - Bug Types
- *User Constraint:*
  - The tests depend on the manual tests specification
- *System Constraint:*
  - Not all bugs may be identified

Figure 3. A hyperlink selection within a rich hypermedia presentation that reveals a corresponding use case

extension of Monk and Howard's [17] rich picture modeling construct. The rich hypermedia shown in Figure 2 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 3). Such an informal computational model can be useful for newcomers to the community looking to become involved in development and offers an overview of the process and its context in the project, while abstracting away the detail of its activities. The use cases also help identify the requirements for enacting or re-enacting the process as a basis for validating, adapting, or improving the process.

A critical challenge in reconstructing process fragments from a process enactment instance is in knowing whether or not the evidence at hand is related, unrelated, or anomalous. Reliability of associations constructed in this fashion may be strengthened by the frequency of association and the relevance of artifacts carrying the association. If text extraction tools are used to discover elements of process fragments, they must also note the context in which are located in to determine this relevance. One way to do this is using the physical structure of the project's Web site (i.e. directory structure), as well as its logical structure (referencing/referenced artifacts). In the NetBeans quality-assurance (Q-Build) testing example, we can relate the "defects by priority" graph on the defect summary page[2] 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[3]. 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 artifacts flowing through the requirements and release process appears in Figure 4.

These process fragments can now be assembled into a formal process modeling language description of the selected processes. Using the PML grammar and process meta-model, we created an ontology for process description with the Protégé-2000 modeling tool [20]. 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 PML description that results appears in Figure 5.

5 Process Reenactment for Deployment, Validation, and Improvement

Since their success relies heavily on broad, open-ended participation, OSSD projects often have informal descriptions of ways members can participate, as well as offer prescriptions for community building [26]. 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

[2] http://qa.netbeans.org/bugzilla/graphs/summary.html as of March 2004
[3] http://www.netbeans.org/download/xtest-results/index.html as of March 2004

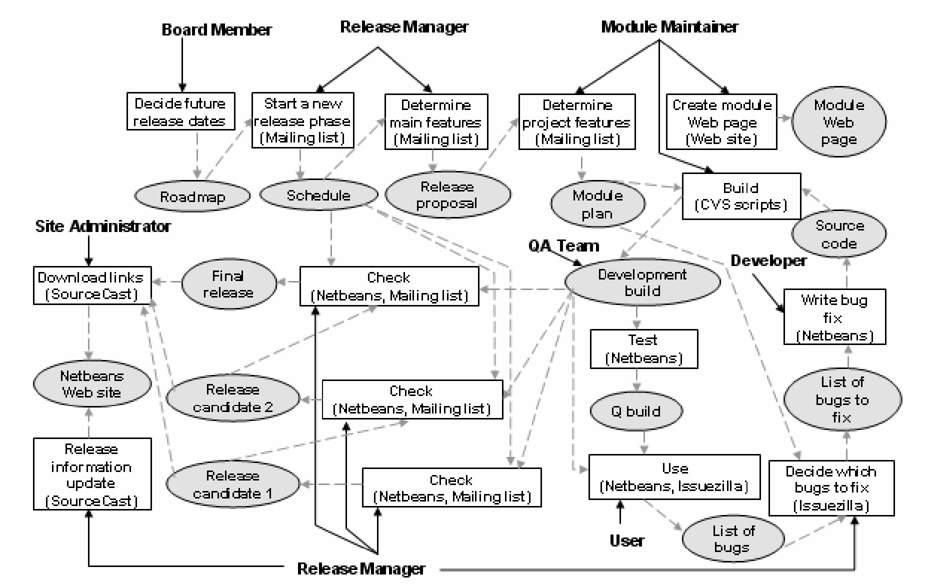

Board Member
Release Manager
Module Maintainer
Decide future release dates
Start a new release phase (Mailing list)
Determine main features (Mailing list)
Determine project features (Mailing list)
Create module Web page (Web site)
Module Web page
Roadmap
Schedule
Release proposal
Module plan
Build (CVS scripts)
Source code
Site Administrator
QA Team
Developer
Download links (SourceCast)
Final release
Check (Netbeans, Mailing list)
Development build
Write bug fix (Netbeans)
Test (Netbeans)
Netbeans Web site
Release candidate 2
Check (Netbeans, Mailing list)
Q build
List of bugs to fix
Release information update (SourceCast)
Release candidate 1
Check (Netbeans, Mailing list)
Use (Netbeans, Issuezilla)
Decide which bugs to fix (Issuezilla)
User
List of bugs
Release Manager

Figure 4. NetBeans Requirements and Release process flow graph [22]

```
sequence Test {
  action Execute automatic test scripts {
  requires { Test scripts, release binaries }
  provides { Test results }
  tool { Automated test suite (xtest, others) }
  agent { Sun Java Studio QA team }
  script { /* Executed off-site */ } }
action Execute manual test scripts {
  requires { Release binaries }
  provides { Test results }
  tool { NetBeans IDE }
  agent {users, developers, Sun Java Studio developers, QA team}
  script { /* Executed off-site */ } }
iteration Update Issuezilla {
  action Report issues to Issuezilla {
    requires { Test results }
    provides { Issuezilla entry }
    tool { Web browser }
    agent{users, developers, Sun Java Studio developers, QA
team}
    script {
      <br><a href="http://www.netbeans.org/issues/">Navigate to
Issuezilla </a>
      <br><a href=http://www.netbeans.org/issues/query.cgi>
Query Issuezilla </a>
      <br><ahref=http://www.netbeans.org/issues/enter_bug.cgi>
Enter issue </a> } }
```

Figure 5. A partial PML description of the testing sequence of the NetBeans release process

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 in modeled processes or process fragments (cf. [1], [23]).

As OSSD projects are open to contributions from afar, it also becomes possible to contribute explicit models of discovered processes back to the project under study so that project participants can openly review, independently validate, refine, adapt or otherwise improve their own software processes. Accordingly, we have contributed our process models and analyses of the NetBeans requirements and release process in the form of a public report hosted (and advertised) on the NetBeans.org Web site[4].

Process re-enactment allows us to recreate, simulate, or prototype process enactments by navigationally traversing a semantic hypertext (i.e., PML) representation of the process [19], [25]. These re-enactment prototypes are automatically derived from a compilation of their corresponding PML process model, and the instantiation of the complied result in a Web-based run-time (enactment) environment [19]. One step in the process modeled for NetBeans appears in Figure 6, drawn from the excerpt shown in Figure 5. In exercising repeated simulated process enactment walkthroughs, we have been able to detect process fragments that may be unduly lengthy, which may serve as good candidates for downstream process engineering activities such as streamlining and process redesign [25]. Process re-enactment also allows us, as well as participants in the global NetBeans project, to better see the effects of duplicated work. As an example, we have four agent types that test code. Users may carry out beta testing from a black box perspective, whereas developers, contributors, and SUN Microsystems QA experts may perform more in-depth white-box testing and analysis, and, in the case of developers and contributors, not merely submit a report to the IssueZilla issue tracking system,[5] but may also take responsibility for resolving it.

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 [25]. Process re-enactment prototypes are a useful means to interactively analyze whether or how altering a process may lead to potential pitfalls that can be discovered before they lead to project failure. Over the course of constructing and executing the prototype we discovered some of the more concrete reasons that there are few volunteers for the release manager position. The role has an exceptional amount of tedious administrative tasks that are critical to the success of the project.

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

[4] See http://www.netbeans.org/community/articles/index.html, as of May 2003.
[5] See http://www.netbeans.org/kb/articles/issuezilla.html, as of March 2004.

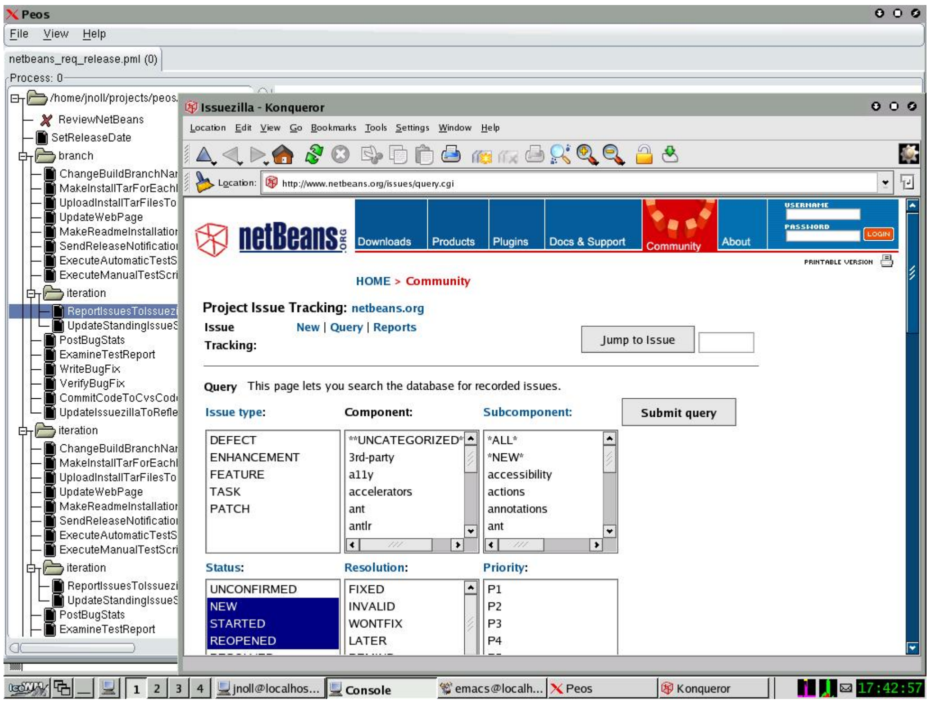


Figure 6. An action step in a re-enactment of the NetBeans requirements and release process, specified in Figure 5

their development process. If any member wishes not to follow a given process, the process enforcement is contingent on the tolerance of her/his peers in the matter, which is rarely the case in corporate development processes. If the project proves intolerant of the alternative process, developers are free to simply not participate in the project's development efforts and perform an independent software release build.

6 Conclusion

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

Our experience with process discovery in the NetBeans project, and its requirements and release process, and our case studies discovering, modeling, and reenacting processes used in the Mozilla and Apache HTTPD projects suggest that a bottom-up strategy for process discovery, together with a top-down process meta-model, can serve as a suitable framework for process discovery, modeling and re-enactment. As demonstrated in the testing example, action sequences are constructed much like a jigsaw puzzle. We compile pieces of evidence to find ways to fit them together in order to make claims about process enactment events, artifacts, or circumstances 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 [12].

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

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

7 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; and Margaret Elliott at the UCI Institute for Software Research are collaborators on the research described in this paper.

References

1. Atkinson, D.C. and Noll, J. 2003.. Automated Validation and Verification of Process Models, *Proc. 7th Intern. IASTED Conf. Software Engineering and Applications*, November.

2. Cass, A.G., Lerner, B., McCall, E., Osterweil, L. and Wise, A. 2000. Little JIL/Juliette: A process definition language and interpreter. *Proc. 22nd Intern. Conf. Software Engineering*, 754-757, Limerick, Ireland, June.

3. 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.

4. Eclipse Web Site, 2005. http://www.eclipse.org

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

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

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

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

9. German, D., 2003. The GNOME project: A case study of open source, global software development, *Software Process—Improvement and Practice*, 8(4), 201-215.

10. Halloran, T., and Scherlis, W. 2002. High Quality and Open Source Software Practices, *Proc. 2nd Workshop on Open Source Software Engineering*, Orlando, FL, May.

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

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

13. Jensen, C. and Scacchi, W. 2005, Process Modeling across the Web Information Infrastructure, *Software Process—Improvement and Practice*, (to appear).

14. Jensen, C. and Scacchi, W. 2005b, Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Open Source Software Development Community, *Proc. 38th Hawaii Intern. Conf. Systems Sciences*, Kona, HI.

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

16. 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.

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

18. NetBeans Web Site, 2005. http://www.netbeans.org

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

20. 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.

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

22. 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/

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. Reis C.R. and Fortes, R.P.M. 2002. An Overview of the Software Engineering Process and Tools in the Mozilla Project, *Proc. Workshop on Open Source Software Development*, Newcastle, UK, February

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

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

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

28. Scacchi, W., Jensen, C., Noll, J. and Elliott, M., 2005, Multi-Modal Modeling, Analysis, and Validation of Open Source Software Development Processes, *Proc. 1st Open Source Software Conference*, Genova, IT (to appear).

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

30. 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.