Process Modeling Across the Web Information Infrastructure

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

Abstract

Web-based open source software development (OSSD) communities provide interesting and unique opportunities for software process modeling and simulation. Whereas most studies focus on analyzing processes in a single organization, we focus on modeling software development processes both within and across three distinct but related communities: Mozilla*, a Web information artifact consumer; the* Apache HTTP server *that handles the transactions of Web information artifacts to consumers such as the Mozilla browser; and* NetBeans*, an integrated development environment (IDE) for creating Web information artifacts. In this paper, we look at the process relationships between these communities as components of a Web information infrastructure. We look at expressive and comparative techniques for modeling such processes that facilitate and enhance understanding of the software development techniques utilized by their respective communities and the collective infrastructure in creating them.*

Keywords

Process Modeling, Open Source Software Development, Apache, Mozilla, NetBeans

1. Introduction

Large-scale geographically distributed software development projects present challenging process problems. The Apache, Mozilla, and NetBeans open source software development communities collectively have millions of estimated users and tens of thousands of community members contributing in one fashion or another. Such magnitudes would be difficult for most closed source organizations to manage. Yet these three communities have proven extremely successful at it. Further, they have done so in a delicate ecosystem of evolving Web standards and tools. These standard technologies and tools compose framework for integrating each community's tools together. Therefore, as Web standards evolve, each community must negotiate its position within the process space or suffer its collapse.

At ProSim 2003, we discussed discovery and modeling of a single community development process [Jensen and Scacchi 2003]. Here, we look at processes within and across three related open source software development communities [cf. Scacchi 2002].

In our efforts to model software development processes on both community and infrastructure levels, we have used a variety of techniques. These include a detailed narrative model of the process, a semi-structured hyperlinked model, a formal computational process model, and a reenactment simulator, all of which serve as input for other process engineering activities [Scacchi and Mi 1997]. Further, all of our models are hypermedia artifacts that may be produced and consumed by the software products of the processes they describe. Our belief is that the richness provided by these modeling techniques will prove scalable from the simplest to most complex processes as well as facilitate, enhance, and expedite their understanding and analysis in comparison with static linear models.

We will set the stage with a discussion of each process modeled independently before taking the infrastructure together, as a whole. Finally, we look at the modeling techniques themselves, how they may be used to guide developers, and how they can serve as a basis for process simulation and other process activities.

2. Process Modeling Techniques

As previously introduced [Jensen and Scacchi 2003], we address three process modeling techniques here as a sampling of those we have applied in our study. These are the rich hypermedia, process flow graphs, and formal modeling. Formal modeling in turn supports tools for simulated reenactment of software processes, which is used to preview, interactively walkthrough, validate, and support process training on demand [Scacchi and Mi 1997]

2.1. Rich Hypermedia

Based on the rich picture concept described by Monk and Howard [1998], we created a rich hypermedia variant as a semi-structured model of software development processes in each of Mozilla, Apache, and NetBeans projects, showing the relationships between tools, agents, their development concerns, and activities that compose the process. Whereas Monk and Howard propose a static model, our hypermedia is interactive and navigational [Noll 2001], including process fragments captured and hyperlinked as use cases. Use cases are a known technique compatible with the Unified Modeling Language (UML) for representing (user) process enactment scenarios [Fowler 2000]. The hypermedia artifacts are also annotated with detailed descriptions of each tool, agent, and concern. Each of these process objects is hyperlinked to its description. Descriptions can, in turn, be linked to other data or hypermedia resources. In this way, the modeler can define the scope of the rich hypermedia to include as little or as much information as the need requires. The rich hypermedia provides a quickly discernable intuition of the process without the burden of formalization. Figure 1 displays an example of a rich hypermedia model as an image map for the Mozilla Quality Assurance process.

2.2. Process Flow Graph

The process flow graph illustrates the flow of development artifacts through a path of interaction with process agents and activities. This workflow diagram provides some sequential ordering of the process fragments and allows us to tease out dependencies between artifacts and activities seen in the rich hypermedia. It also offers an idea of which artifacts and activities are most vital to development, by measuring the fan-in and fan-out of each.

These artifacts are the most likely to be the cause of bottlenecks in the development process when they are found to be inadequate, incomplete, or faulty results of prior development activities. Borrowing from Web modeling terminology, an artifact that is a hub or nexus for several activities will hold up development until it is completed or found satisfactory. Likewise, an artifact that is a product of several inputs inhibits activities that require it until it is ready for further processing. Additionally, we can also detect cycles of development, such as in the stabilization process and refining the software build release plan.

While these insights can be captured in other means, this diagram, like the rich hypermedia, provides an overall representation of the context for process activities without the weight of the details of more formal models. Process entities shown in the flow graph may also be hyperlinked to resources in the community Web to provide interactive richness, as well as to enable process inspection activities. Figure 2 shows a process flow graph for the Apache HTTPD Server project’s release process, where the boxes denote process activities and ellipses denote the resources or artifacts flowing through the process. Further, software developer roles are associated with each process activity.

2.3. Formal Modeling

We developed formal models of software processes following an ontology [Mi and Scacchi 1996] based on PML described in [Noll and Scacchi 2001] using Protege-2000 [Georgas 2002, Noy, *et al.*, 2001]. The work done here is identifying instances for all the PML process meta-model components: agents, resources, tools, actions and activity control flows, which we represent using the Protege-2000 tool. Once a process instance is input, it may be exported to an XML format and also a graphical representation using the Ontoviz tool. Protege-2000's facilities for scoping of process entities (i.e. tools, actions, agents, and resources) in graphical rendering, allowing process experts to focus analysis on certain process entity relationships, abstracting unrelated information. Such portable formats are important given the complexity of the processes rendered.

While the graphical process rendering can be more intuitive than a coded textual format, the textual representation can be used as input to other process lifecycle activities such as enactment and prototyping. Figure 3 shows a graphic representation of an underlying PML model of the NetBeans Requirements and Release process that has been interpreted for visual rendering and layout of its relational interdependencies.

2.4. Reenactment Simulator

Process analysis seeks to identify potential pitfalls that can be discovered prior to their deployment or adoption in a project. Process simulators that can enact or reenact processes are especially useful when validating, modifying, or redesigning a process, as well as for providing on-demand training [Scacchi and Mi 1997].

Our process enactment simulator [Choi and Scacchi 2001, Noll and Scacchi 2001] interactively serves a series of Web pages according to the control flow expressed in the PML model. This simulator allows process performers and other community members to simulate enacting the process through a step-by-step interactive walkthrough. With such an reenactment simulator, developers within a project may be able to exercise, critique, and identify improvement opportunities within processes that can be observed at a distance. It also provides the potential to easily transition from the simulator to live process enactment transactions on the

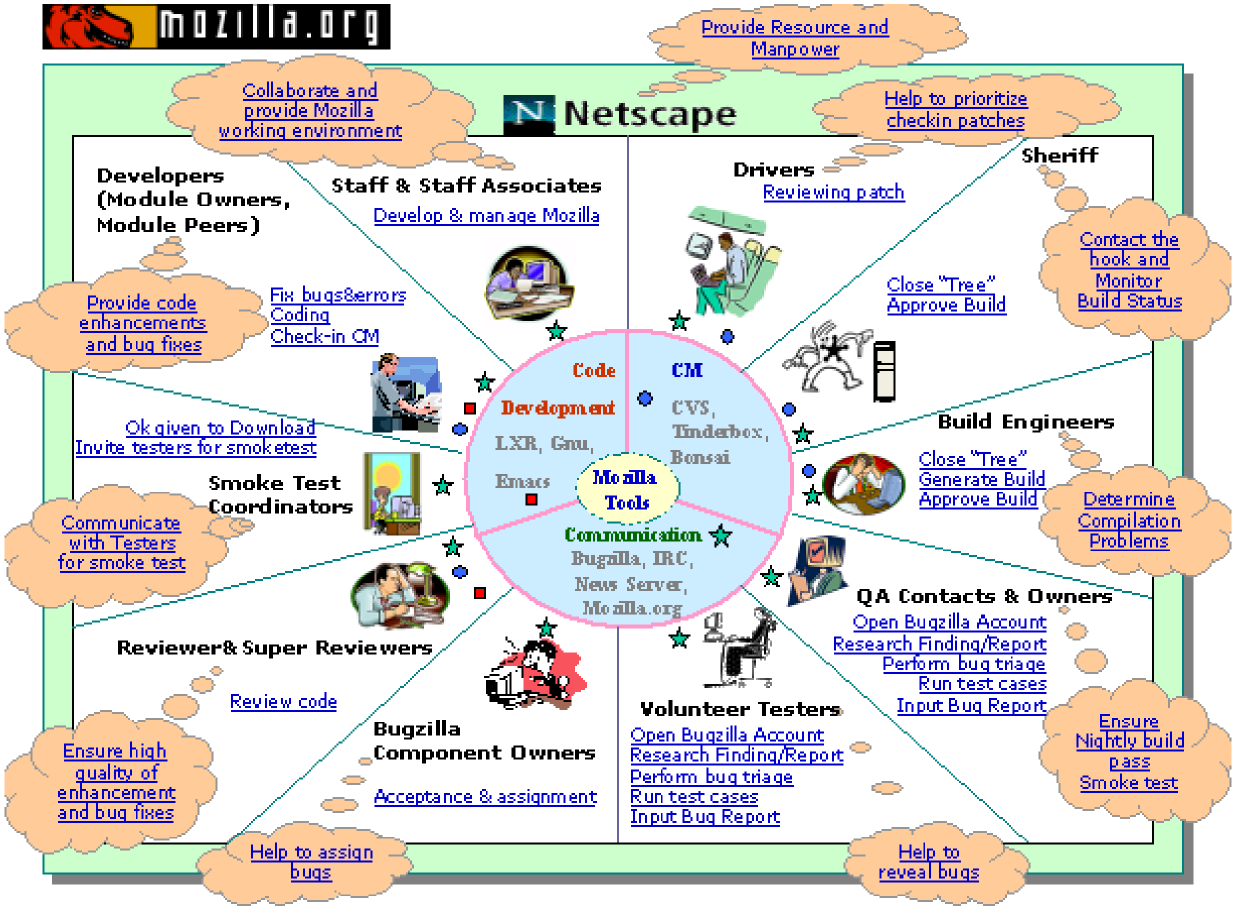


Figure 1: Mozilla quality assurance process rich picture [cf. Carder, et al 2002]

community Web site.[1] 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 process streamlining or reorganization. It allows us to better see the effects of duplicated work. Figure 4 displays a screenshot of the NetBeans Requirements and Release process [Jensen and Scacchi 2003].

3. Modeling Processes Within Web Information Infrastructure Projects

The Apache HTTP Web server, Mozilla Web browser, and NetBeans-based (Java) Web applications together form a Web information infrastructure. However, as the projects that develop each of these open source software systems operate as virtual enterprises [Noll and Scacchi 1999], we have no basis to assume that their development process activities, roles, or tools are identical or common. Thus, in order for these projects, and other open source software projects like them [cf. Scacchi 2002], to collectively produce and sustain a viable global Web information infrastructure, they must be able at some point to synchronize and stabilize their processes, their process activities, shared artifacts, and targeted software releases [cf. Cusumano and Yoffie 1999].

Before we can understand how software development processes in each of these three Web information infrastructure components fit with the others, we must understand them individually. We start by presenting a brief overview of the *quality assurance (QA) process* in the Mozilla Web browser release cycle, followed by the Apache *release process*, and lastly, the NetBeans *requirements and release* process.

3.1. Mozilla Quality Assurance Process

The daily Mozilla QA cycle [Carder, et al., 2002] (see Figure 1) begins with the closing of the

[1] For example, the NetBeans.org project posted a copy of our ProSim'03 Workshop paper [Jensen and Scacchi 2003] where some of these ideas were initially proposed and evaluated. See http://www.netbeans.org/community/articles/UCI_papers.html.

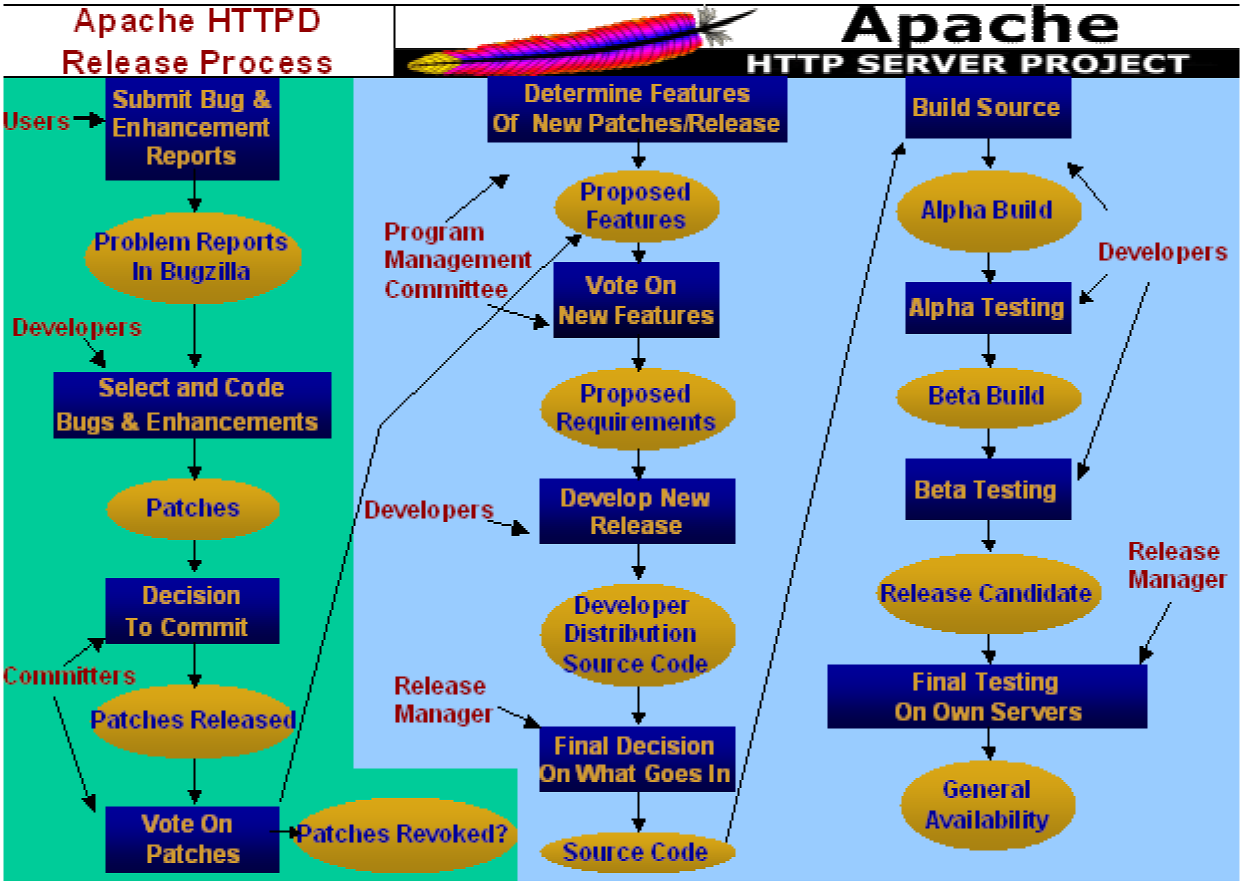


Figure 2: Apache HTTP server release process flow graph [cf. Ata, et al 2002]

source tree to submissions. After this, the "code sheriff" and system build engineer create a build of the source code tree using the Mozilla Tinderbox build tool. If build errors are present, the sheriff and build engineer contact the "on the hook" developers, reviewers, and super-reviewers who were responsible for the offending source, who are called on to correct the defects. When the defect is corrected or offending source removed, the source is rebuilt. This process iterates until all build errors are corrected.

When no build errors are present, the source is placed on the community FTP server and the "smoke test" coordinator issues a call for developers and volunteer testers to download the build via the community Internet relay chat (IRC) channel. After this, QA contacts, QA owners, and volunteer testers will announce what they plan to test, download and install the build and perform a series of smoke tests, security specific (SSL) smoke tests, or less critical "general tests" (periodic regression checkups) based on bug reports submitted to the bug repository. Testers note and discuss the results over the IRC channel. Critical bugs are identified and assigned to the "on the hook" developers to be patched whereupon the source is retested. Non-critical bugs are set aside until they are confirmed by another tester, uploaded to the Bugzilla defect repository, and further dealt with at a later time. Once all critical defects are corrected, the sheriff and build engineer reopen the source tree to further development and source submission.

When first detected, defects are entered into Bugzilla as unconfirmed, noting their severity, component, and platform where the defect was observed. A member of the quality assurance team (either a QA contact or owner) must then research the defect and certify it as a new defect or marking it as a duplicate of another known defect. Patches are then created by developers during the course of development or by drivers as the release date approaches to ensure the overall quality of the product, and the status revised to reflect the changes.

3.2. Apache HTTP Server Release Process

The Apache release process [Ata, *et al.*, 2002; Erenkrantz 2003] follows a somewhat similar path as in NetBeans, though individual roles in the process are different. As shown by the flow graph in Figure

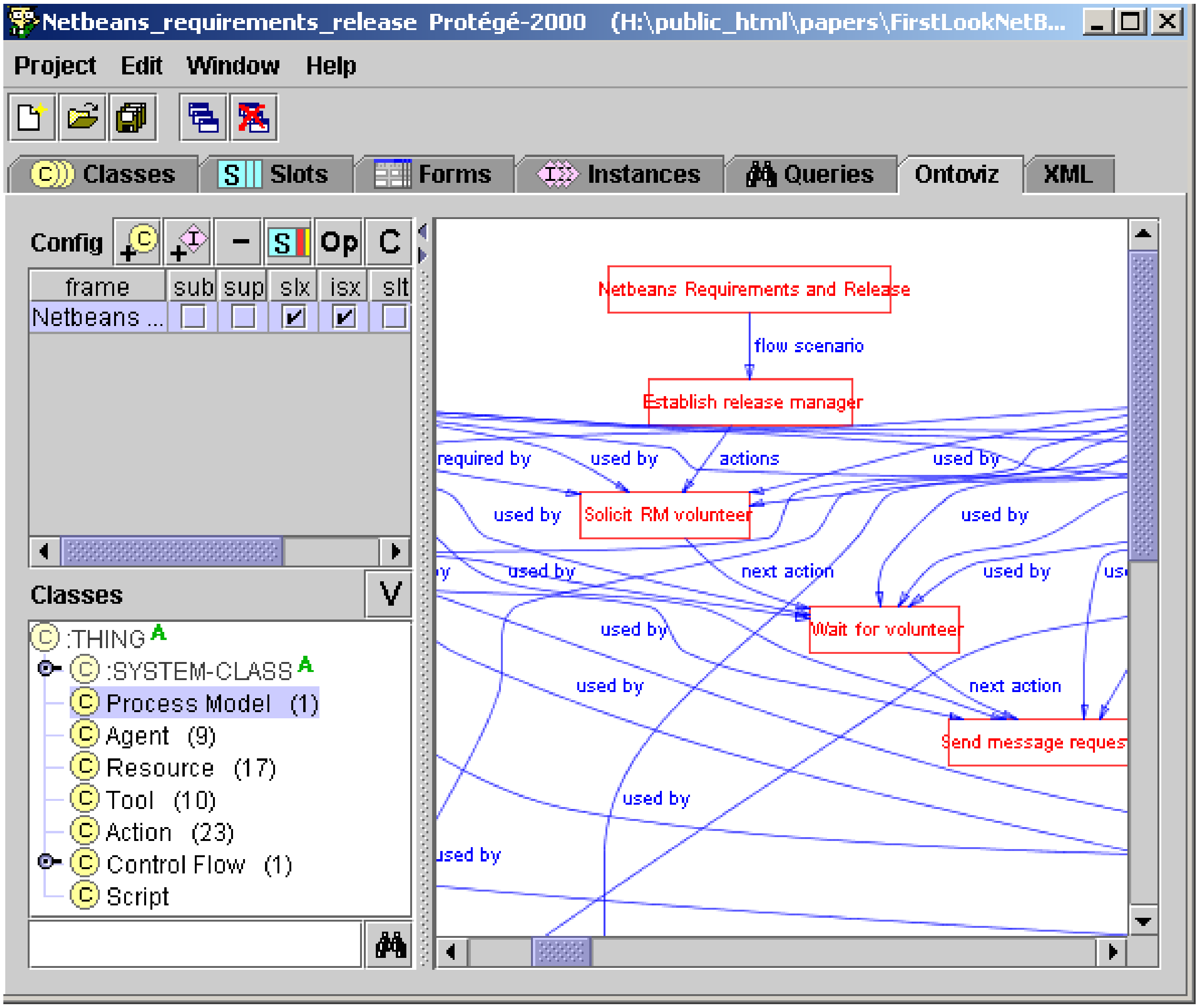
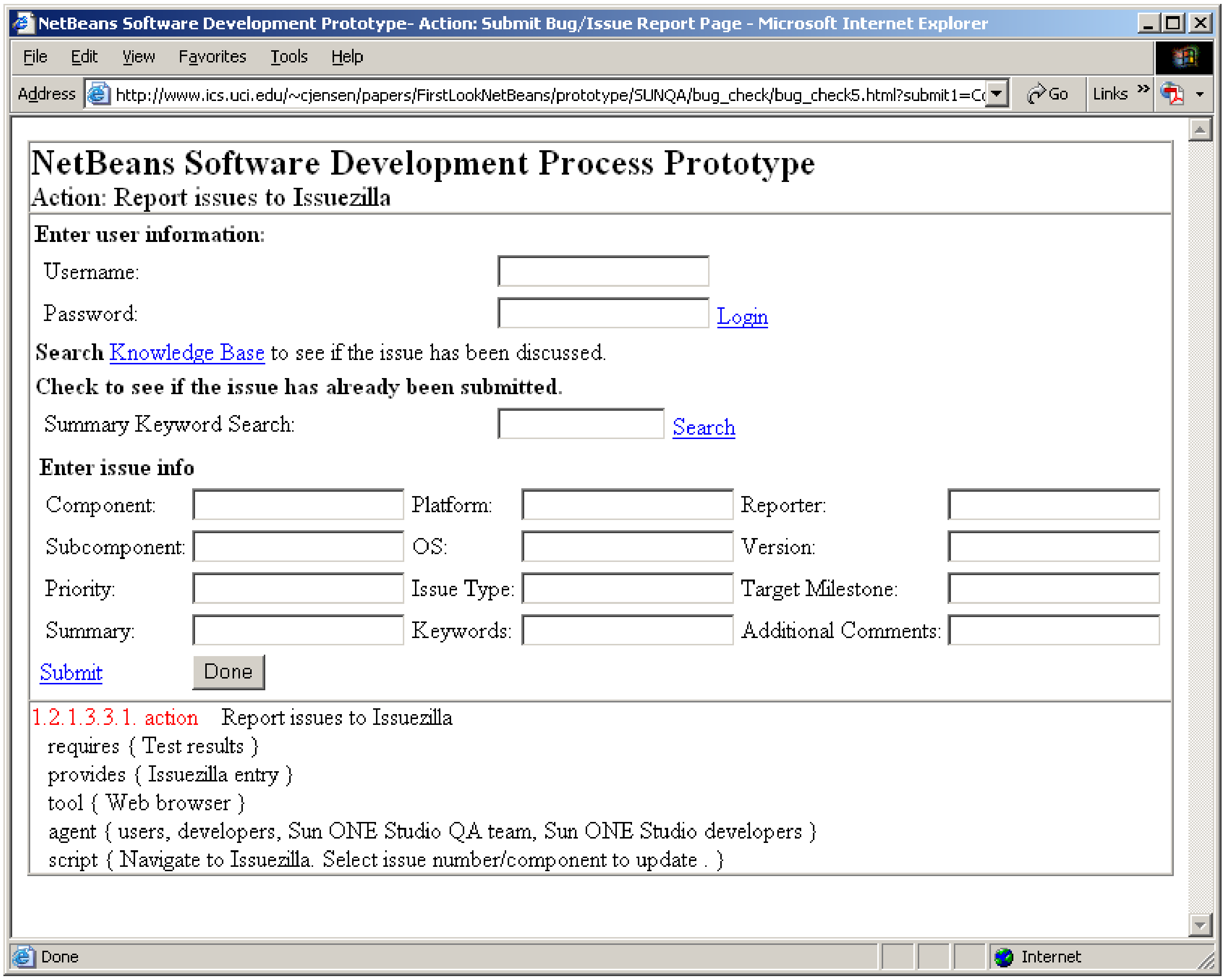


Figure 3: NetBeans requirements and release process formal model rendering with Protégé-2000

2, in the release process, the program management committee puts forth a set of proposed features, which are gleaned from the project roadmap, patches, enhancement reports submitted to the Bugzilla repository, and suggestions from committee members. These are then voted on by the committee and fashioned into a requirements proposal that guides development. Developers volunteer for implementing the features ratified by the voting process.

Feature implementations are submitted as patches to the server. Apache developers with committer status review the submitted patches and vote on whether to accept into the source tree or revoke each based on quality and completeness. As development moves towards completion, the release manager determines which features are fit for inclusion in the release and which are not. Those that pass are compiled into an alpha build, which is made available on the community Web and announced on the developer mailing lists. Developers and committers are then called upon to test the build on their own servers manually or using the Apache automated test suite. Discovered defects are submitted to Bugzilla and patched by developers and subsequently subjected to the patch review process.

When the release manager is adequately satisfied with quality of the source, s/he will declare the release suitable for beta or final release candidacy. When s/he announces this, the builds are made available on the main page of the community Web and adopted by a wider audience, for continued testing and patching. At some point, the release manager deems the source fit for general public use and creates a general availability build, announcing it on the development, committer, and tester mailing lists. This build is then voted on by the committers and tested on the Apache community Web site. If


NetBeans Software Development Prototype- Action: Submit Bug/Issue Report Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://www.ics.uci.edu/~cjensen/papers/FirstLookNetBeans/prototype/SUNQA/bug_check/bug_check5.html?submit1=C
Go
Links
Done
Internet

NetBeans Software Development Process Prototype

Action: Report issues to Issuezilla

Enter user information:

Username:

Password: Login

Search Knowledge Base to see if the issue has been discussed.

Check to see if the issue has already been submitted.

Summary Keyword Search: Search

Enter issue info

Component: Platform: Reporter:

Subcomponent: OS: Version:

Priority: Issue Type: Target Milestone:

Summary: Keywords: Additional Comments:

Submit Done

1.2.1.3.3.1. 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 . }

Figure 4: NetBeans requirements and release process reenactment simulator

there is a simple majority of approval and at least three positive votes, the release is declared final. The finality is announced via the community Web and mailing lists and distributed via a system of mirrored Web sites.

3.3. NetBeans Requirements and Release Process

The NetBeans requirements and release process is depicted formally and reenacted as shown in Figures 3 and 4, respectively. The first step in the NetBeans requirements and release process [Oza, *et al.*, 2002; Jensen and Scacchi 2003] is to establish a release manager, a set of development milestones (with estimated completion dates), and a central theme for the release. The theme is selected by the community members who have taken charge of the release, with the goal of overcoming serious deficiencies in the product (e.g. quality, performance, and usability), in addition to new features and corrective maintenance planned by module teams. Historically, most releases have been led by members employed by Sun Microsystems, which provides development and financial support for the community, though volunteer releases also occur. Based on this, and in conjunction with input from the feature request reports, lead developers will draft a release plan, providing the milestones, target dates, and features to be implemented in the upcoming release. After review and revision by the community, the plan is accepted and developers are asked to volunteer to complete the tasks outlined therein and a volunteer is sought to act as release manager and coordinate efforts of community. Usually, a developer will either volunteer or be volunteered for the role via the mailing list by and accepts the nomination or is accepted through community consensus.

All creative development must be completed by the feature freeze milestone date specified in the release proposal, which signals the end of the

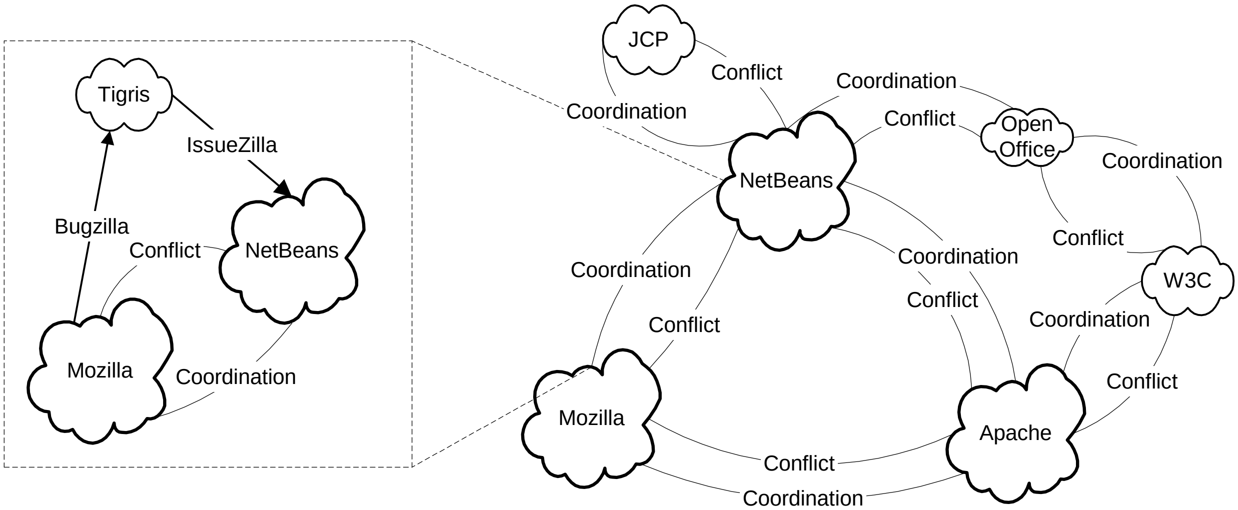


Figure 5: Intercommunity interprocesses communication in the Web information infrastructure

requirements subprocess and the beginning of the stabilization phase- the release subprocess. At this point, only bug fixes may be submitted to the source tree. The stabilization phase consists of a build-test-debug cycle. Nightly builds are generated by a series of automated build scripts and subsequently subjected to a series of automated test scripts, the results of which are posted to the community Web site. Additionally, the quality assurance team performs a series of automated and manual testing every few weeks, as part of the Q-Build program with the aim of ensuring that source code submitted regularly meets reasonable quality standards. Defects discovered during testing are then recorded in the IssueZilla issue repository and subsequently corrected. When the release branch is believed to be devoid of critical "showstopping" defects, it is labeled a release candidate. If a week passes without any further showstoppers, the release candidate is declared final; else the defect is corrected and another release candidate is put forth.

4. Modeling Processes Across Web Information Infrastructure Projects

The successful interoperation between the components of a Web information infrastructure depends on adherence to a shared set of standards. For Mozilla to correctly present Web artifacts, it must implement both protocols for processing Web transactions to the Apache server, and also standards for displaying content of the document or object types generated by NetBeans. Similarly, NetBeans must produce artifacts and applications forms that artifact consumers, including Mozilla and Apache, expect. Apache, for its part, must comply with the transaction protocol Mozilla anticipates (e.g., HTTP), and provide Web application module support required by applications produced by NetBeans.

The inter-community synchronization and stabilization process is a continuous define-implement-revise cycle between communities. When an individual community varies from the standard or implements a new standard, the other communities must act to support it. Likewise, defects in data representations or operations of one tool can cause breakdowns or necessitate workarounds by the others. Thus, synchronization and stabilization of shared artifacts, data representations, and operations or transactions on them is required for a common information infrastructure to be sustained.

This process is not "owned", located within, or managed by a single organization or enterprise. Instead, it represents a collectively shared set of activities, artifacts, and patterns of communication across the participating communities. Thus, it might better be characterized as an ill-defined or ad hoc process that differs in form during each enactment.

4.1. Community Interoperation

Dependencies between communities roughly fall into two categories. On the one hand, we have technologies including protocols, such as HTTP, specifying the process for data communication between software (or hardware) tools, as well as formats specifying how data is organized within a document (e.g. XML, Javascript). Secondly, we find instances where one community integrates another's software tool into their own. If we believe process discovery and modeling are progressive endeavors, these dependencies suggest certain bodies of evidence that will lead to greater understanding of an intercommunity process characterizing their relationship with respect to some larger end. In this case, the intercommunity process is the ongoing development of the infrastructure and the end entails the alignment, integration, or interoperation of system components from each community within the shared information infrastructure. Accomplishing such an end is continually negotiated and potentially reconfigured by the individual goals of each of the participating stakeholders. The first insights into the infrastructure process are community interactions, stakeholder goals and concerns. The rich hypermedia captures these data, though at too a high level to declare a sequence of interprocess communication across communities. We address this next.

4.2. Interprocess Communication across Communities

Communications between communities provide both opportunities for collaboration and sources of conflict between them [Elliott and Scacchi 2003, Jensen and Scacchi 2004]. Communication is collaborative if it identifies compatibilities or potential compatibilities between development projects. From a process perspective, collaborative communications enable external stakeholders to continue following their internal process as normal, perhaps with a small degree of accommodation. They also reinforce infrastructural processes since they do not require changes in the interoperations between communities. If the degree of accommodation becomes too great, the communication can precipitate conflict between communities. Conflict may occur due to changes in tools or technologies shared between them, or in contentious views/beliefs for how best to structure or implement new functionality or data representations across projects. These conflicts require extensive process articulation to adapt.

With few exceptions (e.g. open letters between IBM/Eclipse and Sun/NetBeans), communication between communities is not direct. Instead, we see it in the form of version changelogs announcing support (and changes in support) for tools and technologies integrated into development. It may also appear in defect/feature request repositories, email discourse, and community newsletters within the respective community Web sites, in addition to external news sources (e.g. slashdot.org and freshmeat.org). Communities must monitor these information sources to assess their degree of impact and whether the impact is directly or indirectly collaborative or conflictive. NetBeans, for example, uses the IssueZilla bug/feature request repository developed by the Tigris community, which is, in turn, an extension of Mozilla's Bugzilla tool (see Figure 5).

Communication "channels" (i.e., recurring patterns of communication of shared artifacts, data representations, or protocols) connect process inputs and outputs of each community within the infrastructure. Each channel between communities denotes ad hoc processes or process fragments that describe the interoperability of tools and technologies between them, as well as the "boundary objects"[2] [Star 1989] that are shared between them. The Web information infrastructure development process can therefore be characterized by the communication flow between its constituent organizations. Subsequently, if this communication flow is discernable, it can be represented as a semi-structured rich hypermedia image map, a flow graph, or as a low-fidelity formal process model.

Thus, we have established that identification of shared tools and technologies between Apache, Mozilla, and NetBeans are a first step to discovery and modeling of the Web infrastructure development process. Secondary and tertiary relationships may be worth noting, however these may indicate that prominent communities are being marginalized in the constructed models. Next, collaboration and conflict processes are observed and loosely modeled as rich hypermedia. Extraction or process fragments guides creation of process flow graph models, which permit formalization and reenactment simulation.

Among coordination and conflict interactions, we can identify several types of issues which we have hinted at above. We now discuss in greater detail.

4.3 Coordination

Coordinative interactions may be communication and collaboration activities or leadership and control activities [Jensen and Scacchi 2004]. Communication and collaboration interaction across communities may occur in the form of bug reports submitted referencing a tool or technology implementation on which another community depends. Collaborative organizations may participate in discussions on newsgroups, email lists, IRC chat channels, and message forums on each other's community Web. Community discussion mediums and newsgroups serve as information outposts for stakeholders, both internal and external to a community. From these sources, members of the infrastructure determine ways in which their tools and technologies can become compatible with one another. Further, meta-communities have appeared to support coordination of independent efforts of several communities towards a common goal. The

[2] Boundary objects are those that both inhabit several communities of practice and satisfy the informational requirements of each of them.

Java Tools Community (JCT) is one such community whose goal is to create a technology by establishing standards for tool interoperability between IDEs.

One common way open source software development communities define success is in terms of market share. To achieve and maintain market share, communities must interact with other members of the infrastructure in ways that the target demographic of users will find somehow compelling, and more so than alternative products and services. Gaining an advantage often requires influencing the evolution of external tools and technologies to the benefit of one particular use (and often to the detriment of others) or through increased coupling between communities for mutual benefit. In this way, leadership and control of the evolution of the infrastructure are causes for coordination, as well as potentially for conflict with other organizations. The JTC is one such example where establishing a particular standard for interoperability between several high profile tools may be a contentious goal among communities that seek to increase their market share. Such communities may be enticed to follow the standard and gain entrance into the JTC in an attempt to woo existing users of compatible tools to adopt and use the new "standard."

4.4 Conflict

Conflictive activities arise often from organizations competing for market share and control of the technical direction of infrastructure and shared technologies. It also arises from common and less belligerent activities, such as introducing a new version of a tool or database that other organizations depend on, requiring massive effort to incorporate. In these cases, the organization placed into conflict may simply choose to reject adopting the new tool or technology alterations, possibly selecting a suitable replacement tool/technology if the current one is no longer viable. This path was chosen by the shareware/open source image editing community infrastructure due to patent conflicts with the GIF image format in the early and mid 1990s, leading to the creation of the portable network graphics (PNG) image format standard.

Conflicts across open source software development projects are resolved in collaborative means, through communication on message forums and the like. Alternatively, an organization causing or resisting a tool or technology may cave to pressure exerted by support from rest of the infrastructure. Irreconcilable differences, if they become persistent and strongly supported can lead to divisions in the infrastructure.

5. Discussion

Apache, Mozilla, and NetBeans are three prominent members of a larger organizational ecosystem. In the three space of software development, this ecosystem forms a plane as a development domain: the Web information infrastructure. Other prominent members of this ecosystem include OpenOffice.org, Tigris.org, the World Wide Web Consortium (W3C), and the Java Community Process (JCP). We look at these three communities because they developing large-scale software systems and related products through complex processes that coordinate efforts of tens of thousands of developers with millions of users. At the same time, the ecosystem is not static. Communities rise and fade from prominence. As they increase in mass (membership) and interconnectivity, they create a sense of both gravity and inertia around them, and other organizations may seek coordinative relationships. While closed source projects tend to enjoy tightly coupled integration with relatively few counterparts, open source software communities tend towards loosely coupled interoperability with many counterparts. The effect of this is that there are more organizations impinging on the ecosystem with more complex but weaker bindings than those of proprietary system relationship networks, which are both sparser and less changing.

6. Conclusion

In this paper, we described techniques and issues in modeling software processes used within three large open source software development communities. The software developed in these communities form an information infrastructure for creating, serving, and consuming Web information artifacts. We demonstrated how development processes within these communities interact in terms of ad hoc or fragmentary processes across communities. Finally, we show the potential for Web information artifacts to model the processes of the Web information infrastructure that promotes a more comprehensive, multi-model understanding of the processes rendered. Through increased process understanding, organizations may gain insight into modes of process improvement and interaction with components of their respective work systems.

7. Acknowledgments

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075, #ITR-0205679, and #ITR-0205724. No endorsement implied. Contributors to work described in this paper include Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; John Georgas, Maulik Oza, Eugen Nistor, Susan Hu, Bryce Carder, Baolinh Le, Zhaoqi Chen, Veronica Gasca, Chad Ata, Michele

Rousseau, and Margaret Elliott at the UCI Institute for Software Research.

8. References

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

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

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

Cusumano, M. and Yoffie, D., Software Development on Internet Time, *Computer*, 32(10), 60-69, October 1999.

Elliott, M. and Scacchi, W., Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration, *Proc. ACM Intern. Conf. Supporting Group Work*, 21-30, Sanibel Island, FL, November 2003.

Erenkrantz, J. "Release Management Within Open Source Projects," *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, May 2003.

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

Georgas, J. “Software development process using Protégé.” University of California, Irvine. 9 June, 2002. http://www.ics.uci.edu/~jgeorgas/ics225/index.htm

Jensen, C., Scacchi, W. “Simulating an Automated Approach to Discovery and Modeling of Open Source Software Development Processes.” *In Proceedings of ProSim'03 Workshop on Software Process Simulation and Modeling,* Portland, OR May 2003.

Jensen, C., Scacchi, W. “Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Community.” *In Proceedings of the Fourth Workshop on Open Source Software Engineering ICSE04-OSSE04*, Edinburgh, Scotland, (to appear), May 2004.

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

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

Noll, J. and Scacchi, W., Supporting Software Development in Virtual Enterprises, *J. Digital Information*, 1(4), February 1999.

Noll, J. and Scacchi, W. “Specifying Process-Oriented Hypertext for Organizational Computing,” *J. Network and Computer Applications*, 24(1):39-61, 2001.

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

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/

Scacchi, W. and Mi, P., Process Life Cycle Engineering: A Knowledge-Based Approach and Environment, *Intern. J. Intelligent Systems in Accounting, Finance, and Management*, 6(1):83-107, 1997.

Scacchi, W., Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings—Software,* 149(1), 24-39, February 2002.

Star, S. L., The Structure of Ill-Structured Solutions: Boundary Objects and Heterogeneous Distributed Problem Solving, in *Distributed Artificial Intelligence* (eds. L. Gasser and M. N. Huhns), Vol. 2, pp. 37-54. Pitman, London.