Dynamic Process Discovery, Modeling, and Recovery:

Managing Knowledge Intensive Distributed Systems

Supplemental Award

Final Report

Walt Scacchi and John Noll
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3425
wscacchi@uci.edu, jnoll@scu.edu
June 2005

The research results documented in this final report benefited from a supplemental award to grant #0205679 from the National Science Foundation. No endorsement implied.

Summary

This report documents the results, findings, and research methods employed in examining issues in the management of knowledge-intensive distributed systems in a research project funded by through the National Science Foundation during the period of April 2003- June 2005. The project was based at the University of California, Irvine Institute of Software Research (www.isr.uci.edu), though the research effort was performed through collaborative studies conducted at ISR and at the Santa Clara University. The project was directed by Dr. Walt Scacchi at ISR in collaboration with Dr. John Noll at SCU and ISR.

This project focused on examining concepts, techniques, and tools that could be used to discover, model, analyze and repair hidden workflows that operate within or across multiple dispersed organizations, or multi-site enterprises. The initial results indicate that our approach are promising and can in fact be used to discover hidden workflows and processes operating within or across globally dispersed virtual enterprises, as demonstrated through our empirical studies examining large-scale open source software development projects as the distributed enterprise domain. Our results appear to offer a basis that could serve as part of an overall scheme for managing knowledge-intensive distributed systems (MKIDS).

The report is organized into five major sections, following a brief introduction. Each section contains at least two chapters that represent and document our approach and the results we produced. The first section provides an overview that starts from our proposed effort, followed by an example study that utilizes the various concepts, techniques, and tools that we investigated throughout the project. The second section examines our approach to discovering processes from examination of hidden workflows at play within large-scale, globally dispersed virtual enterprises, specifically those associated with large, corporate-sponsored open source software development projects. The third section examines our approach to developing tools and techniques for computationally modeling the processes or hidden workflows that have been discovered. The fourth section described techniques for analyzing and re-enacting modeled processes so as to verify or validate their form and substance through mechanisms separate from process discovery and modeling. The fifth and final section examines studies of how hidden workflows or processes can breakdown or fail due to unmanaged conflicts, lack of appropriate leadership and coordination mechanisms that rely on internal or external communication systems, discourse artifacts, and other objects that span the boundaries that separate the distributed and hidden workflows of people doing knowledge-intensive software development tasks.

Finally, this report closes with whom to contact for access or further information regarding the software tools and techniques developed or enhanced through this project.

Introduction

OSSD projects are knowledge-intensive efforts that are dispersed across participants in multiple locations working at different times in a loosely-coupled or mostly autonomous manner. OSSD projects are typically self-organizing and are at best weakly coordinated by a corporate sponsor, when such a sponsor exists. But in projects like NetBeans.org, Apache.org, Eclipse.org, and Mozilla.org, they can involve the active contribution of hundreds or thousands of knowledge workers who collectively act to develop complex software systems and related development documents or artifacts. Global OSSD projects are thus quintessential endeavors that are poorly understood, yet produce complex knowledge-intensive products (OSSD systems and development artifacts) through unseen or hidden workflows that are potentially globally dispersed. Thus we believe they are an ideal candidate for which to study MKIDS related phenomena, as well as potentially benefiting from a resulting MKIDS information infrastructure.

The materials in this report start from what we initially proposed as our approach for how to conduct this study. This is described in a chapter titled, Dynamic Process Enactment, Discovery and Recovery. The vision outlined proscribes an ambitious multi-year study, though the actual effort was limited to a two year effort. Thus, some aspects of the proposed effort were not fully examined or resolved, particularly in the area of how to diagnose, repair, and recover from processes or hidden workflows that breakdown or fail during enactment. Nonetheless, we were able to study process breakdowns or conflicts that arise in our chosen problem domain. Thus a foundation for further study in this area has been established.

The second chapter provides an overall summary of our research methods and results for how to discover, model, analyze, and enact (or re-enact) distributed processes (derived from hidden workflows) that span a globally dispersed, large-scale, corporate-sponsored open source software development project like NetBeans.org. Emphasis here is directed at the example case of that proposes a multi-modal approach to modeling the previously hidden process that determines how the NetBeans.org enterprise conducts its "requirements and release" activities associated with the periodic release of a new version of the NetBeans software system. These results are found in a chapter titled, Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes. This chapter and the one that precedes it thus provide an overview of where the project started and the kind of empirical results that we could document and demonstrate by the end of the project period.

The next section focuses on process discovery. The third chapter provides an introduction to the use of a process meta-model that can serve as a reference framework that can guide the discovery of processes associated with hidden workflows. It is titled, Applying a Reference Framework to Open Source Software Process Discovery. The fourth chapter titled, Data Mining for Software Process Discovery in Open Source Software Development Communities, examines the result of data mining of Web-based online artifacts associated with the knowledge-intensive development of large OSSD projects.

The third section examines issues arising from the modeling of processes, whether these processes are independently constructed, or else derived via process discovery. The fifth chapter, Flexible Process Enactment Using Low-Fidelity Models, examines the development of process models for automated workflows using a technique called, "low-fidelity" models. This technique embraces and encourages the modeling of processes in generic terms that eschew all but the minimum of technical detail needed to create a computer-based navigatible specification of a process that can be re-enacted using a Web-compatible process execution operating environment. This technique in turns helped to refine our concept of The Design of Evolutionary Process Modeling Languages, which is the subject of the next chapter. The seventh chapter then employs these modeling languages and techniques to examine the application of an experimental approach to Modeling Recruitment and Role Migration Processes in OSSD Projects. These languages and techniques are similarly applied in another related effort that focuses on Process Modeling Across the Web Information Infrastructure in the eighth chapter. This paper represents a significant research result in being the first such model that accounts for the ongoing development and evolution of a complex knowledge-intensive virtual enterprise of autonomous projects that collectively are responsible for the much of the core information infrastructure of the World Wide Web.

The fourth section examines the development of concepts and automatable techniques for analyzing constructed or derived models of processes so as to help determine their completeness, consistency, traceability and overall (internal) correctness. The ninth chapter presents and demonstrates an approach to the Automated Validation and Verification of Software Process Models. This method in turn gave rise to the need to further refine the scheme and computational methods needed for Process State Inference for Support of Knowledge Intensive Work, which Is the subject of the tenth chapter.

The fifth and last section begins to explore how knowledge-intensive processes and hidden workflows can breakdown, fail, or other disarticulate in the course of their enactment. One reason processes can fail is when the details and understanding of the roles people play in their enactment, the tools and resources they employ in performing the enactment, are hidden or unclear to others either upstream or downstream of the process workflow. The eleventh chapter examines these issues using a case study of Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture. The ethnographic results from this empirical study help lay the foundation for recognizing how beliefs, values, and norms that people do (or do not) share in the course of their knowledge work shapes the actions they take in accomplishing their work and workflow, in ways that subsequently may be visible or invisible (i.e., hidden) from the perspective of others. This insight in turn was then examined in a follow-up case study of a large-scale multi-site OSSD project, in chapter 12, in a paper titled, Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Software Development Community.

Overall, these twelve chapters provide a thorough documentation of the research project we engaged from Spring 2003 through Spring 2005. Our assessment is that the research effort met and exceeded what was originally proposed, given the resources provided to conduct the proposed effort. However, it is also our view that much work remains to be performed and completed before we would consider this line of research and development of an approach to managing knowledge-intensive distributed systems of work that span multiple organizations or project sites finished. Thus, this report serves to document the overall state of a significant work in progress, with further research and support needed to achieve its full potential.

Overview

This section contains the following two chapters. The first is a simplified version of the original proposal that gave rise to this research effort, while the second provides a condensed demonstration of the kinds of results that can be attained at present, when applied in a sample domain of knowledge-intensive work that is performed across a distributed multi-site enterprise.

Walt Scacchi and John Noll, *Dynamic Process Enactment, Discovery and Recovery*, MKIDS Proposal, November 2002.

Walt Scacchi, Chris Jensen, John Noll, and Margaret Elliott, Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes, *Proc. First Intern. Conf. Open Source Software*, Genoa, Italy, July 2005.

Dynamic Process Enactment, Discovery, and Recovery

Walt Scacchi, Institute for Software Research, University of California, Irvine
Irvine, CA 92697-3425 USA, 949-824-4130, Wscacchi@uci.edu

John Noll, Computer Engineering Dept., Santa Clara University
Santa Clara, CA 95053 USA, 408-554-2760, Jnoll@cse.scu.edu
November 2002

Research Goal

Our interest is in understanding three aspects of complex, online knowledge work processes. First is how to provide model-driven process enactment support and event data capture for globally dispersed enterprise processes, resources and users. Second is how to discover process structures and resource usage patterns from emergent and dynamic process enactments. Third is how to recover or repair knowledge work processes that breakdown or fail during enactment. Our goal is to develop and demonstrate concepts, techniques, mechanisms, system architectures, and tools that incorporate these three aspects to enable the subsequent construction of task scheduling and resource allocation/control strategies for coordinating the knowledge work of a distributed complex of people and computing systems.

Problem

The *research area* addressed here is: how to most effectively and efficiently deploy, monitor, learn, and repair the rules, objects, contexts, and teamwork structures that emerge during the enactment of globally distributed knowledge work processes. The *management issue* addressed here is: how to most effectively and efficiently enable flexible process modeling, enactment, reconfiguration and rescheduling of human and computational resources that enables intelligent management response to external change affecting routine, dynamic, or hidden process enactments[1]. Our investigation of these problems is targeted at analysis, design, and prototyping of software system mechanisms, data representations, and system architectures in an iterative and incremental manner, so as to ensure the production and delivery of research results.

Anticipated Results, Deliverables and Transition

We expect to develop and demonstrate concepts, techniques, mechanisms, system architectures and tools that enable the modeling, enactment, discovery, and recovery of complex, online knowledge work processes. The *concepts* will help specify the requirements and design of the tools, techniques, and information infrastructure mechanisms needed to acquire, represent, enact and repair models of dynamic, online knowledge work processes. The *techniques and system architectures* will provide the guiding heuristics and rules of application that coordinate and

[1] In our view, a "process" denotes a set or class of workflows. A computer-based "workflow" specification is an enactable instance of a process. Both entail one or more agents (human or computational) that perform a partially ordered set of tasks using tools that consume resources to produce intermediate or final products.

control the use of the concepts and tools for modeling, enacting, discovering, and recovering knowledge work processes. The *mechanisms and tools* will embody and support these concepts and techniques.

In addition, these results lay the foundation for a comprehensive, integrated knowledge work environment that integrates process modeling, enactment, discovery, and recovery with emerging capabilities for process/resource simulation, visualization, scheduling and allocation/deployment being investigated elsewhere. The design and demonstration of such an environment is an appropriate candidate for a follow-on research investigation. Beyond this, many of our prior research results have been transferred into products that have been commercialized by a variety of firms. Thus, we expect to transition our results to future research investigations and eventually to commercial applications.

Research Approach

We have been involved in systematically observing, modeling, scheduling, integrating, and enacting complex organizational processes for more than 10 years [cf. MS90, SM97, NS91, NS99, NS01, S98, S02b, SN97]. Most of this prior effort has focused on examining and engineering the processes involved in large-scale software system development for commercial, military, or academic applications. For example, we have recently focused on software development processes within globally dispersed virtual enterprises [NS99], particularly those developing open source software [S02a] with centralized corporate sponsorship or control[2]. These processes can be characterized as entailing:

- The production and consumption of knowledge based products (e.g., software programs, development artifacts, documentation, project management reports, and MIME object types)
- The acquisition of information (end-user requirements, software test case evaluation, bug/defect reports from end-users, etc.) from which knowledge products are built
- Having customers with a high level of urgency for these knowledge products
- Reliance on worldwide information sources and repositories that may not be owned or readily controlled

[2] Examples of large projects of this kind include the OpenOffice.org and NetBeans projects sponsored by SUN Microsystems Inc., and the Mozilla.org project sponsored by Netscape and AOL. Models of selected processes for these projects have already been captured and coded using traditional methods [CLC02, ONHJ02]. Another dozen or so small/mid-size open source software projects with centralized corporate sponsors are identified elsewhere [S02c]. Beyond these, organizations like *infoDev*, the Information for Development Program of the World Bank, appear interested in encouraging research, development, and policy studies of open source software for centrally controlled electronic government applications. Finally, large international financial institutions like Barclays Global Investors (BGI) and Dresdner Kleinwort and Wasserstein (DKW) are also centrally controlled enterprises that have invested in and rely on open source software development processes and collaborative development environments to support their global enterprise information systems development projects and operations [S02c].

- A high level of dependence on personnel (software developers) with specialized (application domain specific) expertise and expensive skills

- Access to a worldwide IT infrastructure (Internet, Web, SourceForge.net, etc.) to support product development.

We have also focused attention on the *life cycle engineering of complex organizational processes* [S98, SM97, SN97] for corporate financial operations, telecommunications systems design, military procurement and system acquisition, research grants management, feature film production, interactive teleradiology, and others. Thus we bring an extensive history of prior research results and experience in approaching the problem managing distributed enterprise processes associated with knowledge intensive dynamic systems.

Our approach to the overall problem of how to manage knowledge intensive dynamic systems is process centered. Problems of modeling, analyzing, simulating, integrating, and enacting routine enterprise processes with dynamic or hidden workflows, and how they may be associated with (semantic) hypertext webs of knowledge products, information assets and other repositories, are well known to us [NS91, NS99, NS01, S00, SN97] and others [e.g., HW99, LRS02, SHC01].

One thing we have learned is that *explicit models of complex processes* can directly contribute to continuous process improvement, process redesign, mitigate common process breakdowns, automate Web-based process enactment, and more [cf. NS99, NS01, S98, S00, S01b, S02b, SN97]. These capabilities in turn can lead to dramatic improvements (4X-20X reductions in process cycle time) in process efficiency and effectiveness, as well as cost savings [S01a]. However, acquiring the requisite organizational and process domain knowledge needed to create an explicit high quality process model is a slow, labor intensive endeavor. As a result, we have come to find that it is often nearly as effective to develop and engineer *low-fidelity models*[3] of routine and dynamic knowledge work processes. These simpler models serve as the "seed" from which adaptive process descriptions, proscriptions, or prescriptions can be modeled, grown, repaired, redesigned, and continuously improved.

Our focus in the proposed research effort is to investigate how to configure and rapidly reconfigure process control structures and resources when employing low-fidelity process models. This leads to three lines of study.

- We need to develop scaleable techniques for *modeling, enacting, and capturing* globally dispersed complex enterprise processes that use low-fidelity process models to coordinate and deploy (allocate) access to distributed resources by process users. Our focus is to develop mechanisms and system architectures that can enact low-fidelity process models, together with a capability to capture process and resource event histories in a form suitable

[3] Process models and process enactment instances are similar to abstract plans and instantiated plans found in knowledge-based systems [cf. MS99, SM96]. Low-fidelity process models are thus similar to abstract (or reusable) plans. Process models much like plans, are typically associated with resource allocation and scheduling system capabilities that support complex enterprise activities [MS93, MS99, SHC01, SM97].

for use in process discovery and recovery tasks.

- We need to develop scaleable techniques for *discovering* process control structures within emergent or hidden workflows that can serve as low fidelity process models. These models are created by transparently capturing and generalizing the history of events, conditions, and contexts associated with process enactment activities that create, update, delete, or associate (i.e., hyperlink) online object types or knowledge-based products.

- We need to develop scaleable techniques for *recovering and articulating* the process control structures that breakdown or fail during a routine or dynamic process enactment. Prior research has shown that weakly structured or constraint-relaxed process control structures are prone to breakdown or fail during enactment [BS89, MS93, SM97].

Each of these three investigations is envisioned to leverage and expand the process modeling and enactment framework that we have been investigating for the past few years [NS90, NS99, NS01, SN97]. This entails a collaborative research project (via subcontract) lead by Walt Scacchi (PI, UCI) and John Noll (Co-PI, SCU). Scacchi is leading the effort on process discovery and recovery techniques, while Noll is leading the effort on new techniques and mechanisms for modeling, coordinating, enacting and monitoring (for capture of enactment event histories of) complex enterprise processes.

The following sections briefly elaborate each of the three lines of study identified above.

Modeling, enacting, and capture of globally dispersed enterprise processes

Prior research has investigated how best to specify enterprise processes in sufficient detail to provide some form of active or Web-based process management support [e.g., HW99, LRS02]. In general, Web-based approaches focus effort on encoding work processes as programs that access and manipulate Web-based resources and services. Unfortunately, the power and adaptation of these programs is outside the realm of experience, skill set, or emergent needs of the users who must work with such systems. Instead, we envision a globally dispersed knowledge work environment where dynamic enterprise processes are more transparent, easy to modify, and adaptable by users, whether with or without the support of process programmers. This requires processes that can be both described and interpreted as high-level models [NS01], rather than lower-level workflow programs, middleware, or Web services [LRS02].

This leads us to propose a system to facilitate communication and collaboration among knowledge workers to disseminate process expertise as widely as possible. In this approach, users in different process roles are given high-level guidance (or generalized plans) about what activities to perform or what objectives to achieve, and how to perform them. Users should thus be free to carry out the details of those activities through process enactments that are consistent with or adapt to their expertise. Achieving this entails development of a globally dispersed, model-driven process enactment environment that integrates:

- A *distributed process deployment and execution mechanism* for enacting low fidelity process models. Our prior research experience [BEF02, NB02, NS99, NS01, SN97] suggests that these models can be easily specified or generated. Since low fidelity process models specify the minimal aspects of a work process (e.g., required and/or provided resources, appropriate tools, user roles, and proscribed activities), these models also tend to be stable, reusable, and reconfigurable, as well as enactable via navigational browsing [NB02, NS99, NS01].

- A *virtual repository of artifacts* [cf. NS91, NS99] providing access to distributed collections, repositories, and databases of information objects related to the work to be performed. Information resources (documents, images, diagrams, databases, etc.) will be found in globally dispersed repositories of different types with locally autonomous access and update procedures, that users will want to browse, manipulate, or hyperlink [BEF02, NS91, HW99, NS99, NS01].
- A *data capture facility* for monitoring, collecting, recording, and replaying resource and process enactment event histories [cf. CW98, SM97], that supports process discovery and recovery analyses. Continuous process improvement or emergent process redesign requires knowledge about who did what in a process enactment and shared resource space, when, where, how and why. Getting users to provide this information is too much effort, but providing an automated mechanism to capture (and replay) the data/events would alleviate such effort.

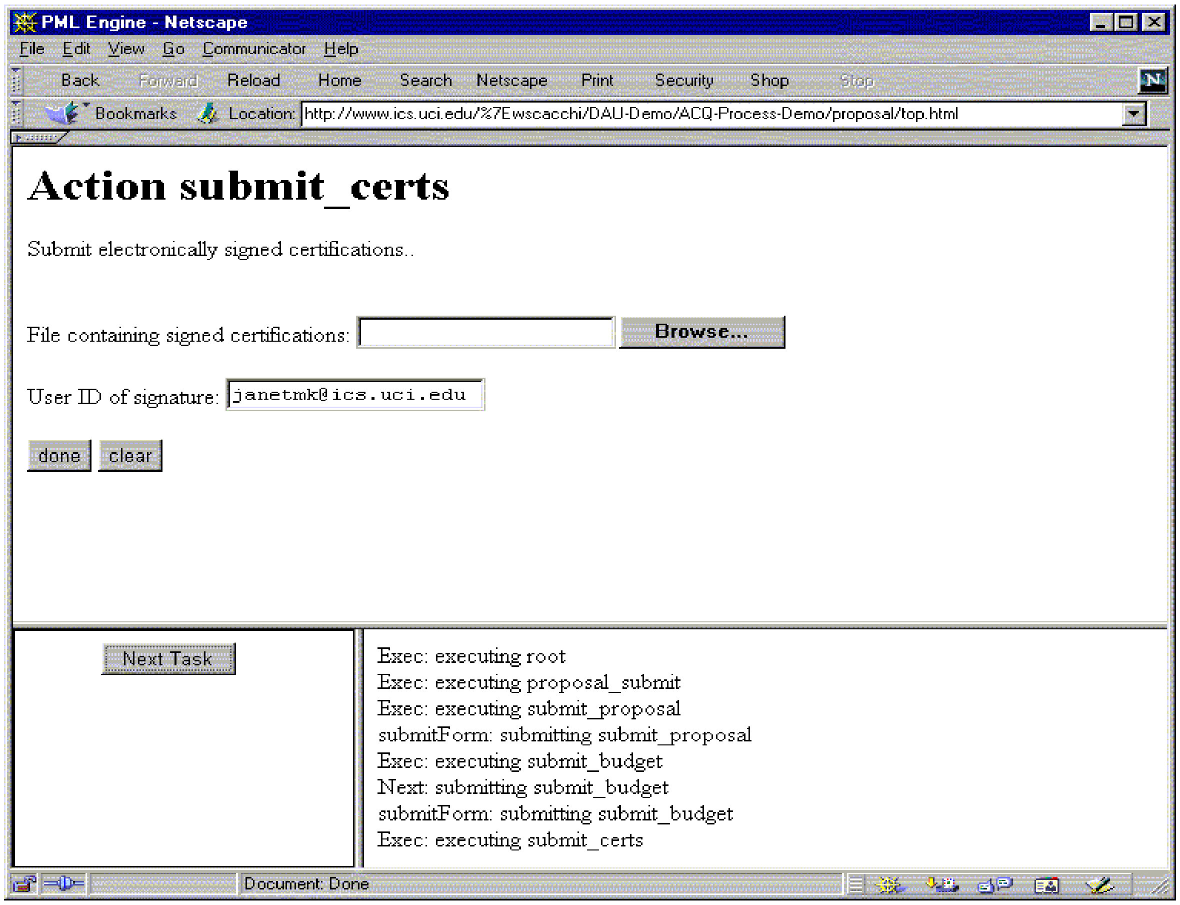
Overall, the model-driven approach to process modeling, enactment, and event capture provides a foundation for process discovery and recovery investigations. In addition, this approach establishes a foundation and dispersed process work environment that can subsequently integrate simulation, visualization, resource allocation/deployment, and scheduling mechanisms into a comprehensive Web-based environment in a follow-on study. As such, our approach to model-driven process enactment and event capture is a significant departure from existing approaches to workflow automation or Web-based services for process automation [cf. LRS02].

Discovering processes from routine, dynamic, or hidden workflow instances

As already noted, acquiring the requisite organizational and process domain knowledge to construct high fidelity process models is valuable, but costly and time-consuming. We need to be able to rapidly construct or (re)configure process models in a less costly, less time consuming, and more transparent manner. In a complex enterprise setting where routine, dynamic or hidden knowledge work processes occur, process enactments will be emergent and reactive, rather than procedural and planned in detail. Furthermore, when process enactments are physically distributed but logically centralized for scheduling and control purposes [NS99], then we need a more innovative technique than traditional for process modeling, analysis or simulation [cf. MS90, S98, SM97].

Prior research in this area has applied grammatical inference, Markov modeling, or temporal constraint ordering techniques to identify process fragments from partially ordered or time-stamped event records to identify or generate process models [CW98, HY03]. Use of Web-based resource usage or process enactment [HW99, LRS02, NS99] event streams has not been explored for discovery purposes, nor have other techniques from knowledge discovery approaches [FPS92]. However, our approach is to employ, evaluate, and refine these kinds of capabilities. For example, in our work, we have experimented with the creation of Web-based process modeling and enactment mechanisms [NS99, NS01, SM97, SN97] that capture process enactment events associated with the manipulation of globally distributed information resources and computing services. Figure 1 displays a screenshot of a step in a procurement process that reveals contextual information and process enactment event history, which is not found in command shell histories [NS01].

We will investigate, evaluate, and develop new techniques for discovering patterns of local-to-global resource and process enactment events, actions, and conditions that manipulate information resources or artifacts (e.g., creating, updating, deleting, or hyperlinking information objects, diagrams, messages, files, Web sites, or reports). This will also require automatically capturing aspects of the process enactment context (e.g., human roles, tools invoked, repositories accessed, network host addresses, and event timestamps) that are attributes of the activities or artifacts involved. We anticipate that we will be able to automatically find process control fragments that are interspersed among a longer set of situated but coincidental (i.e., not process related) actions. Process resource usage events and process execution event histories will be evaluated as one direction for determining what process information can be gathered automatically and transparently.


PML Engine - Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop Stop
Bookmarks Location: http://www.ics.uci.edu/%7Ewscacchi/DAU-Demo/ACQ-Process-Demo/proposal/top.html
Browse...
done clear
Next Task
Document: Done

Action submit_certs

Submit electronically signed certifications..

File containing signed certifications:

User ID of signature: janetmk@ics.uci.edu

```
Exec: executing root
Exec: executing proposal_submit
Exec: executing submit_proposal
submitForm: submitting submit_proposal
Exec: executing submit_budget
Next: submitting submit_budget
submitForm: submitting submit_budget
Exec: executing submit_certs
```

Figure 1. Enactment of a Web-based procurement process with captured history (lower right frame) [Noll and Scacchi 2001, Scacchi 2001a].

Recovering and articulating processes that breakdown or fail during their enactment

Process control structures both shape, and are shaped by, teamwork structures [BS89]. Low-fidelity process models can specify what a process needs to accomplish, but the exigencies of resources, knowledge products, and team membership at hand, give rise to unanticipated conditions or events that must be resolved for work to proceed. These exigencies and unanticipated conditions help characterize how the process enactment at hand is breaking down or failing [MS93, SM97]. These process breakdowns or failures are a primary, recurring cause of business process dynamics [SM97].

In related research, we have developed an approach for diagnosing, replanning, and rescheduling process enactments that breakdown or fail during enactment. This approach is called "articulation" [MS93], and Figure 2 below provides an overview of process enactment repair and recovery control scheme that employs articulation.

Our prior effort with articulation was focused on reasoning about how centralized software engineering processes could be repaired and recovered once they failed. In the proposed effort, we will focus attention on physically distributed but logically centralized processes for knowledge work that utilize globally distributed resources and IT infrastructure. This approach stands in contrast to related efforts that seek to provide Web-based process enactment support [HW99, LRS02] under the assumption that those processes will neither fail in practice, nor need to be adapted, reconfigured, or redesigned on demand. Nonetheless, our approach to articulation appears to be more closely aligned to those efforts that embrace more of a mixed initiative of system provided guidance and user-driven adaptation (e.g., for process enactment planning, resource allocation and task scheduling) that supports a globally dispersed community of knowledge workers through a Web-based environment [cf., BEF02, NS99, MS93, SCH01, SM97].

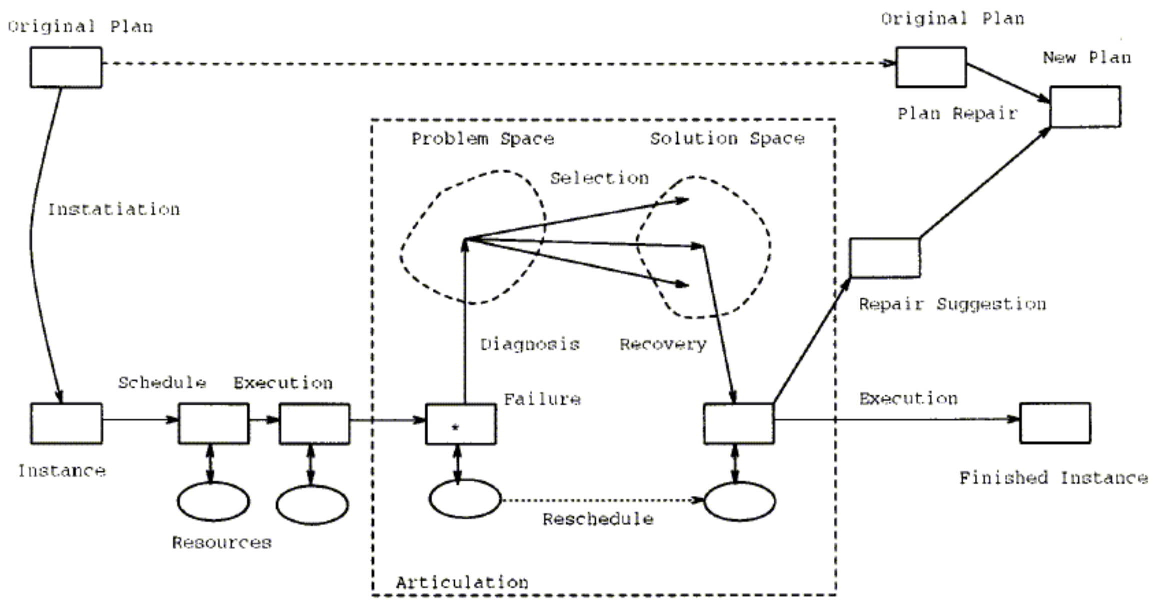


Figure 2. An overview of a knowledge-based approach to diagnosing, replanning and rescheduling process plans that breakdown or fail during enactment [Mi and Scacchi 1993].

References

[BS89] S. Bendifallah and W. Scacchi, Work Structures and Shifts: An Empirical Analysis of Software Specification Teamwork, *Proc. 11th. Intern. Conf. Software Engineering*, IEEE Computer Society Press, Pittsburgh, PA. 260-270, (May 1989).

[BEF02] M. Beiber, D. Engelbart, R. Furuta, S.R. Hiltz, J. Noll, J. Preece, E. Stohr, M. Turoff, and B. Van De Walle, Towards virtual community knowledge evolution, *J. Management Information Systems*, 18(4), 11-36, 2002.

[CLC02] B. Carder, B. Le, and Z. Chen, The Process of Quality Assurance and in the Mozilla Project Release Cycle, Internal project report, Institute for Software Research, June 2002.

[CW98] J.E. Cook and A.L. Wolf, Discovering Models of Software Processes from Event-Based Data, *ACM Trans. Software Engineering and Methodology*, 7(3), 215-249, July 1998,

[FPM92] W.J. Frawley, G. Piatetsky-Shapiro, C.J. Matheus, Knowledge Discovery in Databases: an Overview, *AI Magazine*, 13, 57-70, 1992.

[HW99] J.M. Haake and W. Wang, Flexible support for business processes: extending cooperative hypermedia with process support, *Information and Software Technology*, 41, 355-366, 1999.

[HY03] S.-Y. Hwang and W.-S. Yang, On the discovery of process models from their instances, *Decision Support Systems*, 34(1), 41-57, 2003.

[LRS02] F. Leymann, D. Roller, and M.-T. Schmidt, Web services and business process management, *IBM Systems J.*, 41(2), 198-211, 2002.

[MS90] P. Mi and W. Scacchi, A Knowledge-Based Environment for Modeling and Simulating Software Engineering Processes, *IEEE Trans. Data and Knowledge Engineering*, 2(3), 283-294, September 1990. Reprinted in *Nikkei Artificial Intelligence*, 20(1), 176-191, January 1991, (in Japanese). Reprinted in *Process-Centered Software Engineering Environments*, P.K. Garg and M. Jazayeri (eds.), IEEE Computer Society, 119-130, 1996.

[MS93] P. Mi and W. Scacchi, Articulation: An Integrated Approach to the Diagnosis, Replanning, and Rescheduling of Software Process Failures, *Proc. 8th. Knowledge-Based Software Engineering Conference*, Chicago, IL, IEEE Computer Society, 77-85, 1993.

[MS96] P. Mi and W. Scacchi, A Meta-Model for Formulating Knowledge-Based Models of Software Development, *Decision Support Systems*, 17(4), 313-330, 1996.

[MS99] K. Myers and S. Smith, Issues in the Integration of Planning and Scheduling for Enterprise Control, *Proc.DARPA Symposium on Advances in Enterprise Control,* 1999.

[NB02] J. Noll and B. Billinger, Modeling coordination as resource flow: An object-based approach, *Proc. 2002 IASTED Conf. Software Engineering Applications*, Cambridge, MA, November 2002.

[NS91] J. Noll and W. Scacchi, Integrating Diverse Information Repositories: A Distributed Hypertext Approach, *Computer*, 24(12), 38-45, December 1991.

[NS99] J. Noll and W. Scacchi, Supporting Software Development in Virtual Enterprises, *Journal of Digital Information*, 1(4), February 1999.

[NS01] J. Noll and W. Scacchi, Specifying Process-Oriented Hypertext for Organizational Computing, *J. Network and Computer Applications,* 24(1), 39-61, 2001.

[ONHJ02] M. Oza, E. Nistor, S. Hu, and C. Jensen, NetBeans Requirements and Release Processes, Internal project report, Institute for Software Research, June 2002.

[S98] W. Scacchi, Modeling, Integrating, and Enacting Complex Organizational Processes, appears in K. Carley, L. Gasser, and M. Prietula (eds.), *Simulating Organizations: Computational Models of Institutions and Groups*, 153-168, MIT Press, 1998.

[S00] W. Scacchi, Understanding Software Process Redesign using Modeling, Analysis and Simulation, *Software Process--Improvement and Practice*, 5(2/3), 183-195, 2000.

[S01a] W. Scacchi, Redesigning Contracted Service Procurement for Internet-based Electronic Commerce: A Case Study, *J. Information Technology and Management*, 2(3), 313-334, 2001a.

[S01b] W. Scacchi, Modeling and Simulating Software Acquisition Process Architectures, *J. Systems and Software*, 59(3), 343-354, 15 December 2001b.

[S02a] W. Scacchi, Understanding the Requirements for Developing Open Source Software

Systems, *IEE Proceedings--Software*, 149(1), 2002a.

[S02b] W. Scacchi, Process Models in Software Engineering, in J. Marciniak (ed.), *Encyclopedia of Software Engineering* (Second Edition), 993-1005, Wiley, New York, 2002b.

[S02c] W. Scacchi, Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development, Final Report, NSF CRITO University-Industry Consortium, July 2002c.

[SM97] W. Scacchi and P. Mi, Process Life Cycle Engineering: A Knowledge-Based Approach and Environment, *Intern. J. Intelligent Systems in Accounting, Finance, and Management*, 6(1), 83-107, 1997.

[SN97] W. Scacchi and J. Noll, Process-Driven Intranets: Life Cycle Support for Process Reengineering, *IEEE Internet Computing*, 1(5), 42-49, 1997.

[SHC01] S. Smith, D. Hildum, and D.R. Crimm, Toward the Design of Web-Based Planning and Scheduling Services, *Proc. ECP-01 / Planet Workshop on Automated Planning and Scheduling Technologies in New Methods for Electronic, Mobile and Collaborative Work*, September, 2001.

[VS99] A. Valente and W. Scacchi, Developing a Knowledge Web for Business Process Redesign, Presented at the *1999 Knowledge Acquisition Workshop*, Banff, Canada, October 1999.

Multi-Modal Modeling, Analysis and Validation of Open Source Software Requirements Processes

Walt Scacchi[1], Chris Jensen[1], John Noll[1,2], and Margaret Elliott[1]
[1]*Institute for Software Research*
University of California, Irvine
Irvine, CA, USA 92697-3425
[2]*Santa Clara University*
Santa Clara, CA
Wscacchi@uci.edu

Abstract

Understanding the context, structure, activities, and content of software development processes found in practice has been and remains a challenging problem. In the world of free/open source software development, discovering and understanding what processes are used in particular projects is important in determining how they are similar to or different from those advocated by the software engineering community. Prior studies however have revealed that the requirements processes in OSSD projects are different in a number of ways, including the general lack of explicit software requirements specifications. In this paper, we describe how a variety of modeling perspectives and techniques are used to elicit, analyze, and validate software requirements processes found in OSSD projects, with examples drawn from studies of the NetBeans.org project.

Keywords: software process, process modeling, software requirements, open source software development, empirical studies of software engineering

1. Introduction

In the world of globally dispersed, free/open source software development (OSSD), discovering and understanding what processes are used in particular projects is important in determining how they are similar to or different from those advocated by the software engineering community. For example, in our studies of software requirements engineering processes in OSSD projects across domains like Internet infrastructure, astrophysics, networked computer games, and software design systems [25,26], we generally find there are no explicit software requirements specifications or documents. However, we readily find numerous examples of sustained successful and apparently high-quality OSS systems being deployed on a world-wide basis. Thus, the process of software requirements engineering in OSSD projects must be different that the standard model of requirements elicitation, specification, modeling, analysis, communication, and management [22]. But if the process is different, how is it different, or more directly, how can we best observe and discover the context, structure, activities, and content software requirements processes in OSSD projects? This is the question addressed here.

Our approach to answering this question uses multi-modal modeling of the observed processes, artifacts, and other evidence composed as an ethnographic hypermedia that provides a set of informal and formal models of the requirements processes we observe, codify, and document. Why? First, our research question spans two realms of activity in software engineering, namely, software process modeling and software requirements engineering. So we will need to address multiple perspectives or viewpoints, yet provide a traceable basis of evidence and analysis that supports model validation. Second, given there are already thousands of self-declared OSSD projects affiliated with OSS portals like SourceForge.net and Freshmeat.net, then our answer will be constrained and limited in scope to the particular OSSD project(s) examined. Producing a more generalized model of the OSS requirements process requires multiple, comparative project case studies, so our approach should be compatible with such a goal [25]. Last, we want an approach to process modeling that is open to independent analysis, validation, communication, and evolution, yet be traceable to the source data materials that serve as evidence of the discovered process in the OSSD projects examined [cf. 15].

Accordingly, to reveal how we use our proposed multi-model approach to model requirements processes in OSSD projects, we first review related research to provide the foundational basis for our approach. Second, we describe and provide examples of the modeling modes we use to elicit and analyze the processes under study. Last, we examine what each modeling mode is good for, and what kind of analysis and reasoning it supports.

2. Related Research and Approach

There is growing recognition that software requirements engineering can effectively incorporate multi-viewpoint [7,16,22] and ethnographic techniques [22,31] for eliciting, analyzing, and validating functional and non-functional software system *product* requirements. However, it appears that many in the software engineering community treat the *process* of requirements engineering as transparent and prescriptive, though perhaps difficult to practice successfully. However, we do not know how large distributed OSSD projects perform their development processes [cf. 3].

Initial studies of requirements development across multiple types of OSSD projects [25,26] find that OSS product requirements are continuously emerging [8,9,30] and asserted after they have been implemented, rather than relatively stable and elicited before being implemented. Similarly, these findings reveal requirements practice centers about reading and writing many types of communications and development artifacts as “informalisms” [25], as well as addressing new kinds of non-functional requirements like project community development, freedom of expression and choice, and ease of information space navigation. Elsewhere, there is widespread recognition that OSSD projects differ from their traditional software engineering counterparts in that OSSD projects do not in general operate under the constraints of budget, schedule, and project management constraints. In addition, OSS developers are also end-users or administrators of the software products they develop, rather than conventionally separated as developers and/versus users. Consequently, it appears that OSSD projects create different types of software requirements using a different kind of requirements engineering process, than compared to what the software engineering community has addressed. Thus, there is a fundamental need to discover and understand the process of requirements development in different types of OSSD projects.

We need an appropriate mix of concepts, techniques, and tools to discover and understand OSSD processes. We and others have found that process ethnographies must be empirically grounded, evidence-based, and subject to comparative, multi-perspective analysis [3,7,10,15,22,25,28].

However, we also recognize that our effort to discover and understand OSSD processes should reveal the experience of software development newcomers who want to join and figure out how things get done in the project [27].

As participant observers in such a project, we find that it is common practice for newcomers to navigate and browse the project's Web site, development artifacts, and computer-mediated communication systems (e.g., discussion forums, online chat, project Wikis), as well as to download and try out the current software product release. Such traversal and engagement with multiple types of hyperlinked information provide a basis for making modest contributions (e.g., bug reports) before more substantial contributions (code patches, new modules) are offered, with the eventual possibility of proposing changing or sustaining the OSS system's architecture. These interactive experiences reflect a progressive validation of a participant's understanding of current OSSD process and product requirements [1,19]. Thus, we seek a process discovery and modeling scheme that elicits, analyzes, and validates multi-mode, hypertext descriptions of a OSSD project's requirements process. Furthermore, these process descriptions we construct should span informal through formal process models, and accommodate graphic, textual, and computationally enactable process media. Finally, our results should be in a form open to independent analysis, validation, extension, and redistribution by the project's participants.

3. Multi-Mode Process Modeling, Analysis and Validation using Ethnographic Hypermedia

An ethnographic hypermedia [4] is a hypertext that supports comparative, cross-linked analysis of multiple types of qualitative ethnographic data [cf. 28]. They are a kind of semantic hypertext used in coding, modeling, documenting, and explaining patterns of social interaction data and analysis arising in contemporary anthropological, sociological, and distributed cognition studies. The media can include discourse records, indigenous texts, interview transcripts, graphic or photographic images, audio/video recordings, and other related information artifacts. Ideally, they also preserve the form and some of the context in which the data appear, which is important for subsequent (re)analysis, documentation, explanation, and presentation.

Ethnographic studies of software development processes within Web-based OSSD projects are the focus here. Ethnographic studies that observe and explain social action through online participant observation and data collection have come to be called "virtual ethnography" [12]. Virtual ethnography techniques have been used to observe the work practices, compare the artifacts produced, and discover the processes of OSSD projects found on and across the Web [5,6,13,14,23,25,26,27]. In particular, an important source of data that is examined in such studies of OSSD projects is the interrelated web of online documents and artifacts that embody and characterize the medium and continuously emerging outcomes of OSSD work. These documents and artifacts constitute a particular narrative/textual genre ecology [29] that situate the work practices and characterize the problem solving media found within OSSD projects.

We have employed ethnographic hypermedia in our virtual ethnographic studies of OSSD projects. What does this mean, and what challenges or opportunities for requirements elicitation, analysis, and validation have emerged along the way? These questions are addressed below through examples drawn from case studies of OSSD projects, such as the NetBeans.org project [13,14], which is one of the largest OSSD projects we have studied.

As noted, the OSSD projects we study are found on the Web. Web sites for these projects consist of a network of hyperlinked documents or artifacts. Samples of sites we have studied include NetBeans.org, Mozilla.org, pache.org, and GNUenterprise.org among dozens of others. The artifacts we examine include Web pages, email discussion lists, bug reports, project to-do lists, source code files and directories, site maps, and more. These artifacts are an important part of the data we collect, examine, study, code, and analyze in order to identify OSSD work practices and development processes that arise in a given project.

We create a hypermedia of these artifacts in ways that allow us to locate the originating source(s) of data within the focal project's Web site. This allows us to maintain links to the source data materials that we observe as evidence of the process at hand, as well as to allow us to detect when these data sources have been updated or removed. (We also archive a local copy of all such data). However, we create codings, annotations, and assembled artifacts that embed hyperlinks to these documents as part of our ethnographic hypermedia. As a result, multiple kinds of ethnographic records are created including annotated artifacts, rich hypermedia pictures, and ethnographic narratives. Juxtaposed about these records are other kinds of models including a process meta-model, attributed directed graph model, process domain ontology, and a formal, computationally enactable process model. Each is described next, and each is hyperlinked into an overall ethnographic hypermedia that provides cross-cutting evidence for the observed OSS requirements processes.

Annotated artifacts

Annotated artifacts represent original software development artifacts like (publicly available) online chat transcripts that record the dialogue, discussions, and debate that emerge between OSS developers. These artifacts record basic design rationale in an online conversation form. The textual content of these artifacts can be tagged, analyzed, hyperlinked, and categorized manually or automatically [24]. However, these conversational contents also reveal much about how OSS developers interact at a distance to articulate, debate, and refine the continuously emerging requirements for the software system they are developing. For example, Elliott and Scacchi [5,6] provide conversational transcripts among developers engaged in a debate over what the most important properties of software development tools and components to use when building free software. They provide annotations that identify and bracket how ideological beliefs, social values, and community building norms constrain and ultimately determine the technical choices for what tools to use and what components to reuse when developing OSS.

Navigational rich pictures

Rich pictures [18] provide an informal graphical scheme for identifying and modeling stakeholders, their concerns, objects and patterns of interaction. We extend this scheme to form navigational rich pictures constructed as an Web-compatible hypertext image map that denotes the overall context as the composition and relationships observed among the stakeholder-roles, activities, tools, and document types (resources) found in a OSSD project. Figure 1 displays such a rich picture constructed for NetBeans.org. Associated with each relationship is a hyperlink to a *use case* [2] that we have constructed to denote an observable activity performed by an actor-role using a tool that consumes or produces a document type. An example use case is shown in Figure 2. Each other type of data also is hyperlinked to either a descriptive annotation or to a Web site/page where further information on the object type can be found.

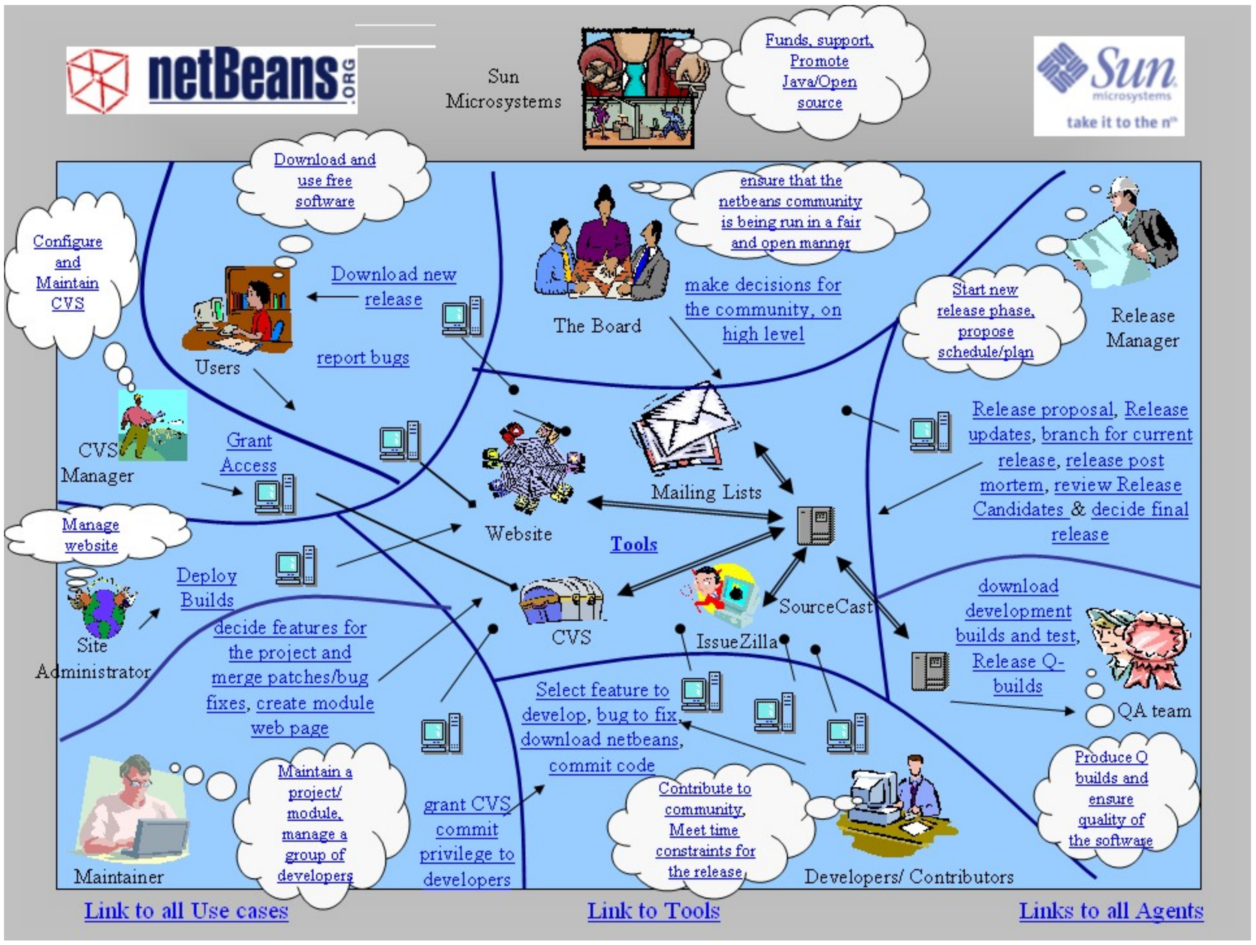
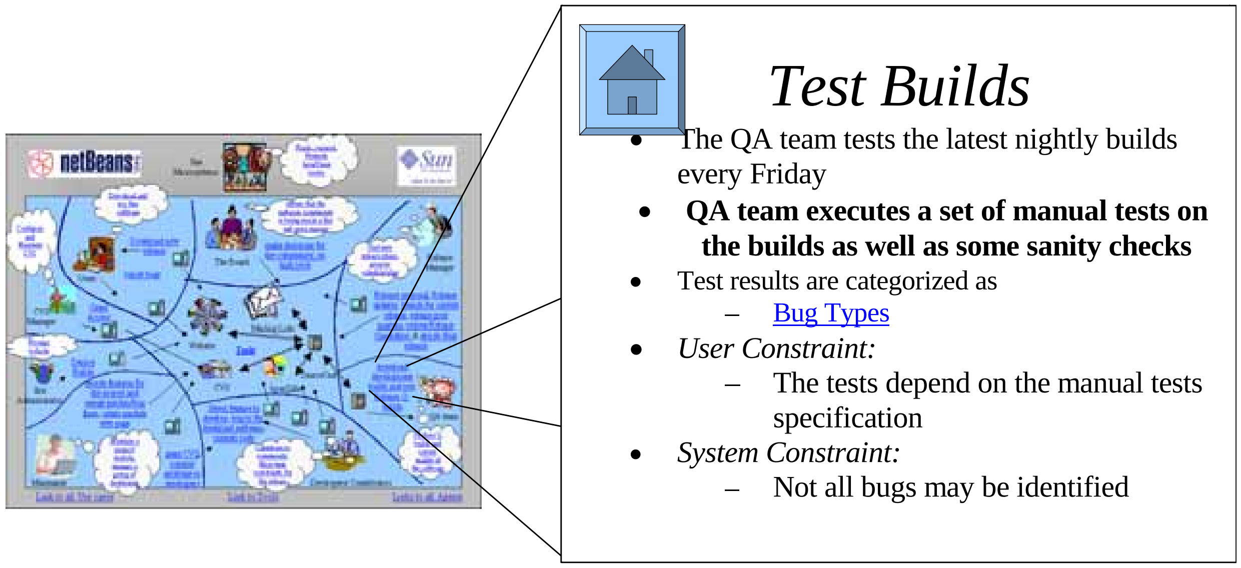

netBeans.org
Sun Microsystems
Funds, support, Promote Java/Open source
Sun microsystems take it to the nth
Download and use free software
Configure and Maintain CVS
Download new release
report bugs
Users
ensure that the netbeans community is being run in a fair and open manner
The Board
make decisions for the community, on high level
Start new release phase, propose schedule/plan
Release Manager
CVS Manager
Grant Access
Website
Mailing Lists
Tools
Release proposal, Release updates, branch for current release, release post mortem, review Release Candidates & decide final release
Manage website
Deploy Builds
Site Administrator
decide features for the project and merge patches/bug fixes, create module web page
CVS
IssueZilla
SourceCast
download development builds and test, Release Q-builds
QA team
Select feature to develop, bug to fix, download netbeans, commit code
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 1. A rich picture image map of the requirements and release process in the NetBeans.org



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

Directed resource flow graph
A directed resource flow graph denotes a recurring workflow pattern that has been discovered in an OSSD project. These workflows order the dependencies among the activities that actor-roles perform on a recurring basis to the objects/resources within their project work. These resources appear as or within Web pages on an OSSD project's Web site. For example, in the NetBeans.org project, we found that software product requirements are intertwined with software build and release management. Thus, the "requirements and release process" entails identifying and programming new/updated system functions or features in the course of compiling, integrating, testing, and progressively releasing a stable composition of source code files as an executable software build version for evaluation or use by other NetBeans.org developers [5,6,23]. An example flow graph for this appears in Figure 3. The code files, executable software, updated directories, and associated email postings announcing the completion and posting the results of the testing are among the types of resources that are involved. Last, the rendering of the flow graph can serve as an image map to the online (i.e., on the NetBeans.org Web site) data sources from where they are observed.

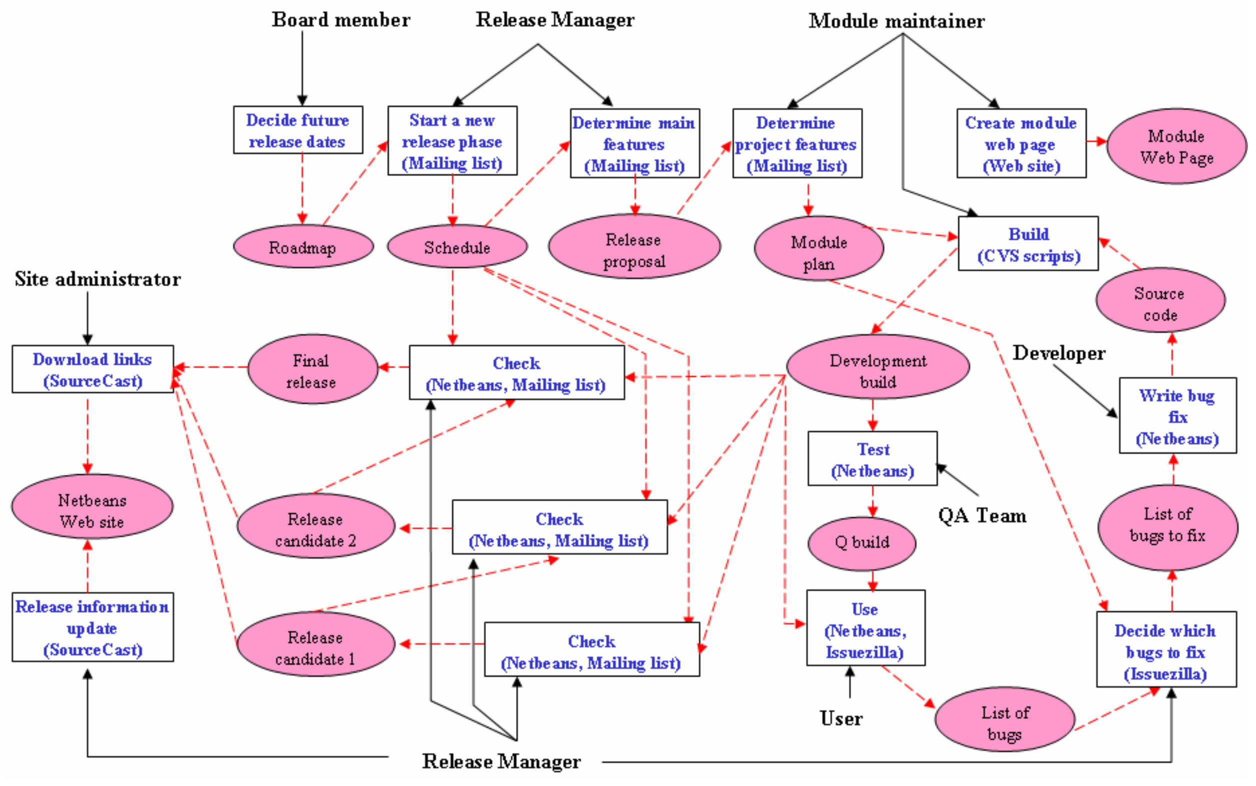

Board member
Release Manager
Module maintainer
Decide future release dates
Start a new release phase (Mailing list)
Determine main features (Mailing list)
Determine project features (Mailing list)
Create module web page (Web site)
Module Web Page
Roadmap
Schedule
Release proposal
Module plan
Build (CVS scripts)
Site administrator
Source code
Download links (SourceCast)
Final release
Check (Netbeans, Mailing list)
Development build
Developer
Write bug fix (Netbeans)
Test (Netbeans)
Netbeans Web site
Release candidate 2
Check (Netbeans, Mailing list)
QA Team
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 3. An attributed directed graph of the resource flow for the NetBeans.org requirement and release process. Boxes denote tasks/actions, ellipses denote resources/objects, dashed lines denote resource flows, and solid lines and labels denote agent/stakeholder roles performing tasks that transform input resources into output resources.

Process domain ontology
A process ontology represents the underlying *process meta-model* [17,20] that defines the semantics and syntax of the process modeling constructs we use to model discovered processes. It provides the base object classes for constructing the requirements process (domain) taxonomies of the object classes for all of the resource and relation types found in the rich picture and directed resource flow graph. However, each discovered process is specific to an OSSD project, and knowledge about this domain is also needed to help contextualize the possible meanings of the processes being modeled. This means that a process domain entails objects, resources or relations that may or may not be have been previously observed and modeled, so that it may be necessary to extend to process modeling constructs to accommodate new types of objects, resources, and relations, as well as the attributes and (instance) values that characterize them, and attached methods that operationalize them.

We use an ontology modeling and editing tool, Protégé-2000 [21] to maintain and update our domain ontology for OSS requirements processes. Using Protégé-2000, we can also visualize the structure of dependencies and relations [11] among the objects or resources in a semantic web manner. An example view can be seen in Figure 4. Furthermore, we can create translators that can transform syntactic form of the modeling representations into XML forms or SQL schema definitions, which enables further process modeling and tool integration options [cf. 14].

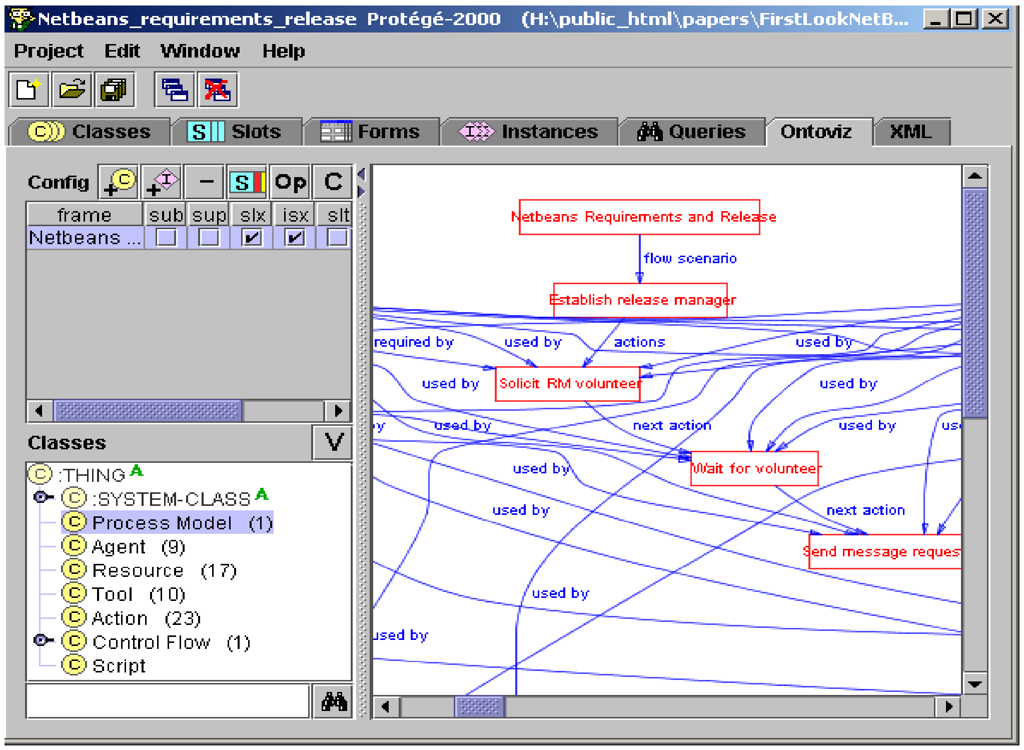


Figure 4. A view of the process domain ontology for the NetBeans.org software requirements and release process.

Formal process model and its enactment

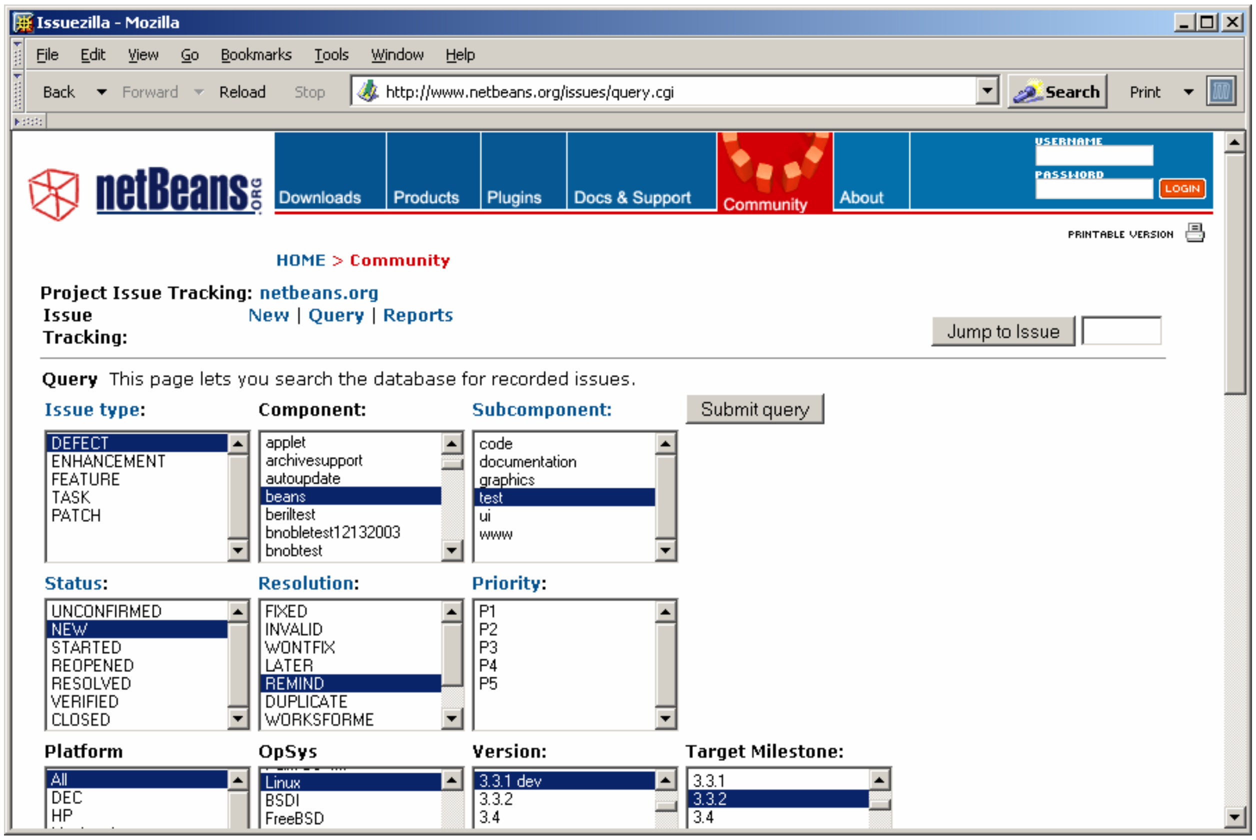
A formal process model denotes a syntactically precise and semantically typed specification of the resource objects, flow dependencies, actor-roles, and associated tools that specifies an enactable (via interactive process-guided user navigation) hypertext representation we call an *organizational process hypertext* [20]. This semantic hypertext, and its supporting run-time environment, enables the ability to walkthrough or simulate enactment of the modeled OSSD process as a process-guided, navigational traversal across a set of process linked Web pages. The semantic hypertext is automatically rendered through compilation of the process models that are output from the ontology editor in a process modeling language called PML [20]. A PML-based model specification enables automated consistency checking at compile-time, and detection of inconsistencies at compile-time or run-time. An example of an excerpt from such a model is shown in Figure 5. The compiled version of the PML produced a non-linear sequence of process-linked Web pages, each one of which corresponds to one step in the modeled process. An example showing the result of enacting a process (action) step specified at the bottom of Figure 5 appears in Figure 6.

```
...
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 { /* Executed off-site */ } }
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 { /* Executed off-site */ } }
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 {
      <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><a href="http://www.netbeans.org/issues/enter_bug.cgi">Enter issue </a> } }
...
```

Figure 5. An excerpt of the formal model of the Netbeans.org requirements and release process coded in PML.

Ethnographic hypermedia narrative

An ethnographic narrative denotes the final view ethnographic hypermedia. This is an analytical research narrative that is structured as a document that is (ideally) suitable for dissemination and publication in Web-based and printed forms. It is a composite derived from selections of the preceding representations in the form of a narrative with embedded hyperlinked objects, and hyperlinks to related materials. It embodies and explains the work practices, development processes, resource types and relations, and overall project context as a narrative, hyperlinked ethnographic account that discovered at play within a given OSSD project, such as we documented for the NetBeans requirements and release process [23]. In printed form, the narratives we have produced so far are somewhere between 1/4 to 1/15 the number of pages compared to the overall set of project-specific data (documents) at the first two levels of hyperlink connectivity; said differently, if the ethnographic report is 30 or so printed pages (i.e., suitable for journal publication), the underlying ethnographic hypermedia will correspond to a hypermedia equivalent to 120-450 printed pages.



http://www.netbeans.org/issues/query.cgi

Project Issue Tracking: netbeans.org

Issue Tracking: New | Query | Reports

Query This page lets you search the database for recorded issues.

Issue type: DEFECT, ENHANCEMENT, FEATURE, TASK, PATCH

Component: applet, archivesupport, autoupdate, beans, beriltest, bnobletest12132003, bnobtest

Subcomponent: code, documentation, graphics, test, ui, www

Status: UNCONFIRMED, NEW, STARTED, REOPENED, RESOLVED, VERIFIED, CLOSED

Resolution: FIXED, INVALID, WONTFIX, LATER, REMIND, DUPLICATE, WORKSFORME

Priority: P1, P2, P3, P4, P5

Platform: All, DEC, HP

OpSys: Linux, BSDI, FreeBSD

Version: 3.3.1 dev, 3.3.2, 3.4

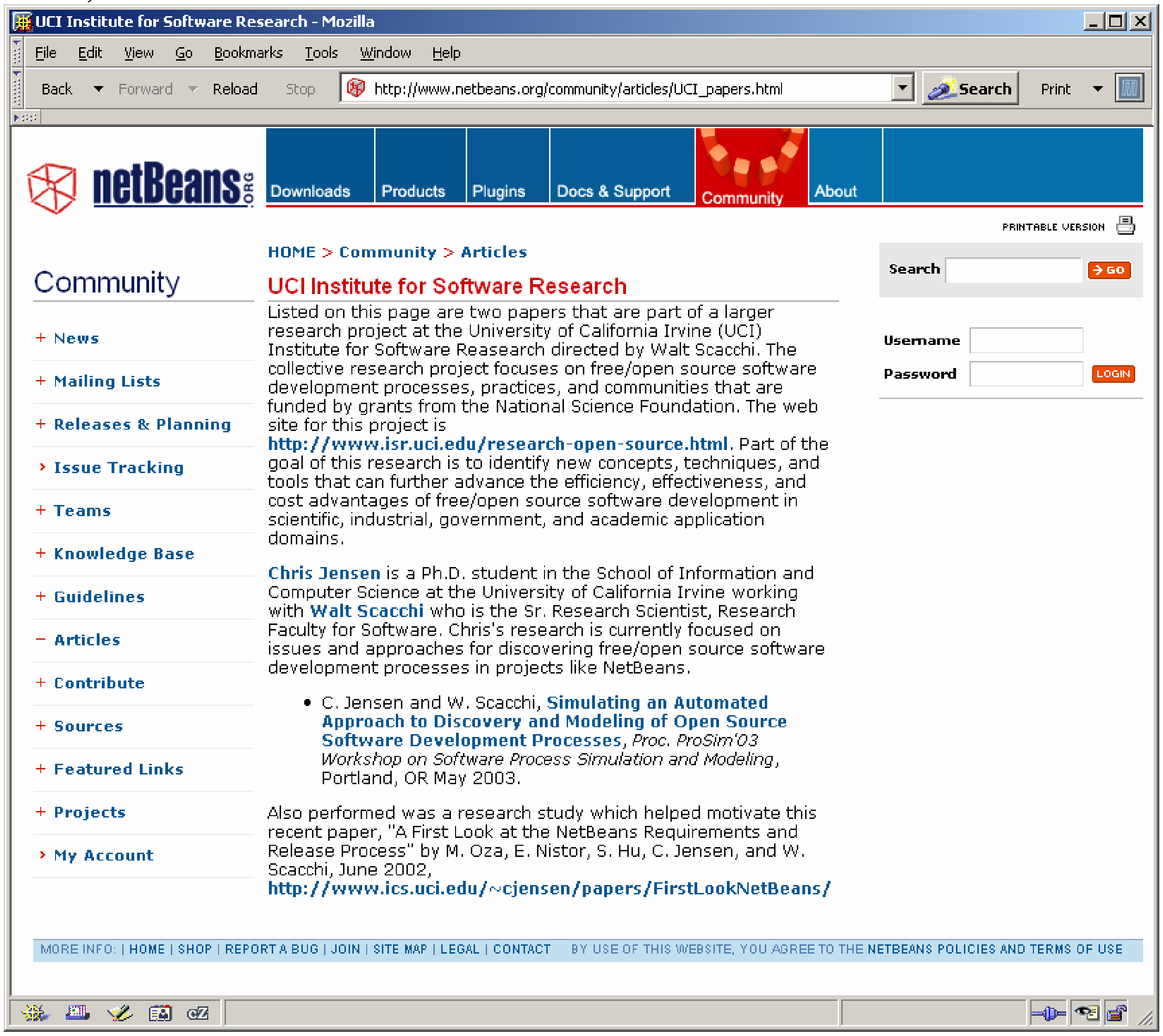
Target Milestone: 3.3.1, 3.3.2, 3.4

Figure 6. A screenshot displaying the result of the PML-based re-enactment of one step ("Action Report issues to Issuezilla—Query Issuezilla") in the NetBeans,org requirements and release process.

4. Discussion

We have learned a number of things based on applying our approach to requirements processes in different OSSD projects. First, no single mode of process description adequately subsumes the others, so there is no best process description scheme. Instead, different informal and formal descriptions respectively account for the shortcomings in the other, as do textual, graphic, and computationally enactable process representations. Second, incremental and progressive elicitation, analysis, and validation occur in the course of developing multi-mode requirements process models. Third, multi-mode process models are well-suited for discovery and understanding of complex software processes found in OSSD projects. However, it may not be a suitable approach for other software projects that do not organize, discuss, and perform software development activities in an online, persistent, open, free, and publicly accessible manner. Fourth, multi-mode process modeling has the potential to be applicable to the discovery and modeling of software product requirements, although the motivation for investing such effort may not be clear or easily justified. Process discovery is a different kind of problem than product development, so different kinds of approaches are likely to be most effective.

Last, we observed that the software product requirements in OSSD projects are continually emerging and evolving. Thus, it seems likely that the requirements process in such projects is also continuously. Thus, supporting the evolution of multi-mode models of OSS requirements processes will require either automated techniques for process discovery and multi-mode update propagation techniques, or else the participation of the project community to treat these models as open source software process models, that can be continuously elicited, analyzed, and validated along with other OSSD project assets, as suggested in Figure 7, which are concepts we are currently investigating. However, it seems fair to note that ethnographic accounts are situated in time, and are not intended for evolution.



HOME > Community > Articles

UCI Institute for Software Research

Listed on this page are two papers that are part of a larger research project at the University of California Irvine (UCI) Institute for Software Reasearch directed by Walt Scacchi. The collective research project focuses on free/open source software development processes, practices, and communities that are funded by grants from the National Science Foundation. The web site for this project is http://www.isr.uci.edu/research-open-source.html. Part of the goal of this research is to identify new concepts, techniques, and tools that can further advance the efficiency, effectiveness, and cost advantages of free/open source software development in scientific, industrial, government, and academic application domains.

Chris Jensen is a Ph.D. student in the School of Information and Computer Science at the University of California Irvine working with Walt Scacchi who is the Sr. Research Scientist, Research Faculty for Software. Chris's research is currently focused on issues and approaches for discovering free/open source software development processes in projects like NetBeans.

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

Also performed was a research study which helped motivate this recent paper, "A First Look at the NetBeans Requirements and Release Process" by M. Oza, E. Nistor, S. Hu, C. Jensen, and W. Scacchi, June 2002, http://www.ics.uci.edu/~cjensen/papers/FirstLookNetBeans/

MORE INFO: | HOME | SHOP | REPORT A BUG | JOIN | SITE MAP | LEGAL | CONTACT BY USE OF THIS WEBSITE, YOU AGREE TO THE NETBEANS POLICIES AND TERMS OF USE

Figure 7. Getting captured and analyzed process models out for validation and possible evolution by NetBeans.org project participants.

5. Conclusion

Ethnographic hypermedia are an important type of semantic hypertext that are well-suited to support the navigation, elicitation, modeling, analysis and report writing found in ethnographic studies of OSSD processes. We have described our approach to developing and using ethnographic hypermedia in support of our studies of requirements processes in OSSD projects like NetBeans.org, where multiple modes of informal to formal representations are involved. We find that this hypermedia is well-suited for supporting qualitative research methods that associated different type of project data, with process descriptions rendered in graphic, textual and computationally enactable descriptions. We provided examples of the various kinds of hypertext-based process descriptions and linkages that we constructed in moving from abstract, informal representations of the data through a series of ever more formalized process models resulting from our studies.

6. Acknowledgements

The research described in this report is supported by grants #0083075, #0205679, #0205724, and #0350754 from the National Science Foundation. No endorsement implied. Mark Ackerman at University of Michigan, Ann Arbor; Les Gasser at University of Illinois, Urbana-Champaign; and others at ISR are collaborators on the research described in this paper.

7. References

1. Bolchini, D. and Paolini, P., Goal-Driven Requirements Analysis for Hypermedia-Intensive Web Applications, *Requirements Engineering*, 9, 85-103, 2004.

2. Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, New York, 2001.

3. Curtis, B., Krasner, H., and Iscoe, N., A Field Study of the Software Design Process for Large Systems, *Communications ACM*, 31(11), 1268-1287, 1998.

4. Dicks, B. and Mason, B., Hypermedia and Ethnography: Reflections on the Construction of a Research Approach, *Sociological Research Online*, 3(3), 1998. www.socresonline.org.uk

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

6. 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 Group Publishing, Hershey, PA, 152-172, 2004.

7. Finkelstein, A.C.W., Gabbay, D., Hunter, A., Nuseibeh, B., Inconsistency Handling in Multi-perspective Specifications, *IEEE Trans. Software Engineering*, 20(8), 569-578, 1994.

8. Gans, G., Jarke, M., Kethers, S., and Lakemeyer, G., Continuous Requirements Management for Organisation Networks: A (Dis)Trust-Based Approach, *Requirements Engineering*, 8, 4-22, 2003.

9. Gasser, L., Scacchi, W., Penne, B., and Sandusky, R., Understanding Continuous Design in OSS Projects, *Proc. 16th. Int. Conf. Software & Systems Engineering and their Applications*, Paris, December 2003.

10. Glaser, B. and Strauss, A., *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine Publishing Co., Chicago, Il, 1967.

11. Grinter, R.E., Recomposition: Coordinating a Web of Software Dependencies, *Computer Supported Cooperative Work*, 12(3), 297-327, 2003.

12. Hine, C., *Virtual Ethnography*, Sage Publications, Newbury Park, CA, 2000.

13. Jensen, C. and Scacchi, W., Collaboration, Leadership, Control, and Conflict Management in the NetBeans.Org Community, *Proc. 5th Open Source Software Engineering Workshop*, Edinburgh, May 2004a.

14. Jensen, C. and Scacchi, W., Process Modeling Across the Web Information Infrastructure, *Proc. 5th Software Process Simulation and Modeling Workshop*, Edinburgh, Scotland, May 2004b.

15. Kitchenham, B.A., Dyba, T., and Jorgensen, M., Evidence-based Software Engineering, *Proc. 26th Int. Conf. Software Engineering*, 273-281, Edinburgh, Scotland, IEEE Computer Society, 2004.

16. Leite, J.C.S.P. and Freeman, P.A., Requirements Validation through Viewpoint Resolution, *IEEE Trans. Software Engineering*, 17(12), 1253-1269, 1991.

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

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

19. Narayanan, N.H. and Hegarty, M., Multimedia Design for Communication of Dynamic Information, *Int. J. Human-Computer Studies*, 57, 279-315, 2002.

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

21. 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.

22. Nuseibeh, B. and Easterbrook, S., Requirements Engineering: A Roadmap, in Finkelstein, A. (ed.), *The Future of Software Engineering*, ACM and IEEE Computer Society Press, 2000.

23. 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/FirstLook
NetBeans/, February 2004 (Original May 2002).

24. Rao, R., From Unstructured Data to Actionable Intelligence, *IT Pro*, 29-35, November 2003.

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

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

27. Scacchi, W., Socio-Technical Interaction Networks in Free/Open Source Software Development Processes, in S.T. Acuña and N. Juristo (eds.), *Peopleware and the Software Process*, World Scientific Press, to appear, 2004b.

28. Seaman, C.B., Qualitative Methods in Empirical Studies of Software Engineering, *IEEE Trans. Software Engineering*, 25(4), 557-572, 1999.

29. Spinuzzi, C. and Zachry, M., Genre Ecologies: An Open System Approach to Understanding and Constructing Documentation, *ACM J. Computer Documentation*, 24(3), 169-181, August 2000.

30. Truex, D., Baskerville, R., and Klein, H., Growing Systems in an Emergent Organization, *Communications ACM*, 42(8), 117-123, 1999.

31. Viller, S. and Sommerville, I., Ethnographically Informed Analysis for Software Engineers, *Int. J. Human-Computer Studies*, 53, 169-196, 2000.

Process Discovery

This section contains the following two chapters. The first proposes a scheme using a process meta-model guided scheme to facilitate manual, semi-automated, or automated discovery of processes from hidden workflows. The second employs the reference model scheme together with textual data mining tools used in analyzing large textual corpora to determine the overall efficacy of such an approach to semi-automated process discovery.

Chris Jensen and Walt Scacchi, Applying a Reference Framework to Open Source Software Process Discovery. In *Proceedings of the First Workshop on Open Source in an Industrial Context,* OOPSLA-OSIC03, Anaheim, CA October 2003.

Chris Jensen and Walt Scacchi, Data Mining for Software Process Discovery in Open Source Software Development Communities, *Proc. Workshop on Mining Software Repositories,* 96-100, Edinburgh, Scotland, May 2004.

Applying a Reference Framework to Open Source Software Process Discovery

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

ABSTRACT
The successes of open source software development have inspired commercial organizations to adopt similar techniques in hopes of improving their own processes without regard to the software process context that provided this success. This paper describes a reference framework for software process discovery in open source software development communities that provides this context. The reference framework given here characterizes the entities present in open source communities that interplay in the form of software processes, discusses how these entities are encoded in data found in community Web spaces, and demonstrates how it can be applied in discovery.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management– *lifecycle, software process models*

General Terms
Management, Measurement, Documentation.

Keywords
Reference Framework, Process Discovery, Open Source

1. INTRODUCTION
Open source software development has existed for decades, though only more recently has it piqued the curiosity of industry and academia. While many, like Eric Raymond in his essay, "The Cathedral and the Bazaar" and Garg [5], with his work on corporate sourcing, have extolled the virtues of the open source development paradigm, seeking methods of bringing the benefits of open source to industry, we still lack an understanding of the process context that enables such successes and in which these techniques lie. With this understanding, we may analyze other process activities and social and technical factors on which these techniques depend, whether they are compatible with existing processes and the larger organizational landscape, and how process techniques may be configured to realize such benefits as have been seen in an open source forum. However, process engineering activities for such analysis and that guide redesign and (continuous) improvement all require a process specification. Thus motivates our interest in process discovery. In previous work [7], we demonstrated the feasibility of automating process discovery in open source software development communities by first simulating what an automated approach might consist of through a manual search of their online Web information spaces. Here, we discuss an approach to constructing the open source software development process reference framework that helps make such automation possible. This framework is the means to map evidence of an enacted process to a classification of agents, resources, tools, and activities that characterize the process. In traditional corporate development organizations, we may be able to readily determine such things by examining artifacts such as the org-chart and so forth. But open source communities often lack such devices. While components of the framework may be known, no such mapping framework exists that enables open source process discovery.

2. RELATED WORK
Weske, et al. [17] describe what they refer to as a reference model for workflow application development processes, though theirs is more of a software development lifecycle model than a software development reference model and provide no insight for mapping the Web information space to a process.

Srivasta, et al. [16] details a framework for pattern discovery and classification of Web data. The discussion relates site content, topology, session information garnered from site files and logs and applies association rules and pattern mining to obtain rules, patterns, and statistics of Web usage. However, they offer no help in constructing the pattern discovery techniques that process the data to arrive at those usage rules.

Lowe, et al. [8] on the other hand, propose a reference model for hypermedia development process assessment. This model, however lacks their domain model does not reflect software development and their process meta-model is awkwardly configured. Nevertheless, the overlap between hypermedia development and open source software development makes is apparent in comparing their reference model with the one presented here.

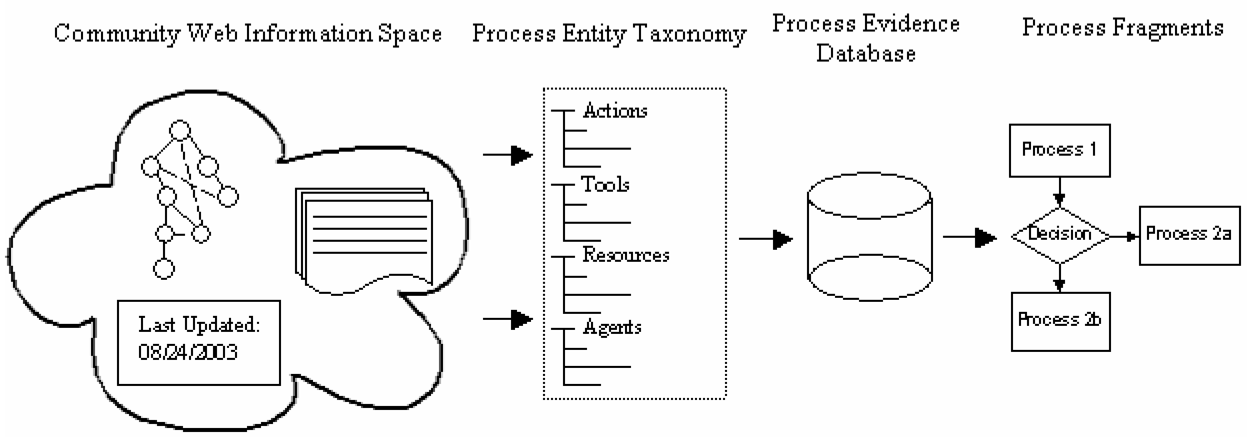


Figure 1. The open source software development reference framework, mapping Web content, structure, and usage/update data to process entities, which are assembled into process fragments via association rules

3. ESSENTIAL ATTRIBUTES OF THE FRAMEWORK

The job of the reference framework is to provide a mapping between process evidence discovered by searching the community Web and a classification scheme of process attributes. Software lifecycle models in combination with probabilistic relational modeling techniques then provide guidance for integrating these relations together into a sequence of process fragments that can be pieced together to form a meaningful model of the development process as shown in Figure 1. Our reference framework is based on the process meta-model of Noll and Scacchi [12]. This meta-model consists of actions, tools, resources, and agents. Whereas Lowe and associates adopted the Spearmint framework, the skill level of the agent is unnecessary for the specification of the software development process. The abstract resource entity is likewise excessive as it caries little semantic benefit in software development process specification. These ingredients are not specific to software development processes, however the reference framework is domain specific. Furthermore, some variance is expected between communities based on the community size, the extent of its maturity, and preferences of the individuals in the community. Thus, while it is not possible to assert that any given community uses a specific testing suite, it is likewise impossible to say that they use a testing suite at all. However, that is the purpose of process discovery, and not the reference framework. With this in mind, we can discuss the contents of the framework.

Surveys of Apache [1], Mozilla [3], and NetBeans [13] led to a taxonomy [14, 15, 18] of tasks, tools, resources, and roles common in open source development. The approach chosen characterizes these process entities in two dimensions: breadth (e.g. communication tools, code editing tools, etc.) and genericity (e.g. an instant messaging client as a type of synchronous communication tool subset of the larger category of all communication tools). This classification scheme is necessary in order to relate instances of process entities to their entity type, which may then be associated with related entities (such as other tasks, tools, resources, and roles).

4. PROCESS MAPPING OF OSSD WEB INFORMATION SPACES

As noted elsewhere [7], there are three dimensions of the information space that encode process evidence:

- Structure: how the Web of project-related software development artifacts is organized
- Content: what types of artifacts exist and what information they contain
- Usage Patterns: user interaction and content update patterns within the community Web

The structure of the community Web is evident in two forms. The physical form consists of the directory structure of the files of which the site is composed. But, it is also apparent on a logical level, in terms of the site layout, as might be given by a site map or menu. These may or may not be equivalent. Nevertheless, each layer in the hierarchy provides a clue to the types of agents, resources, tools, and processes of the community. Structure hierarchy names may be mapped to instances of tools, agents, resources, and activities found in the open source software development meta-model taxonomy, thus fulfilling the first role of the reference framework. Additionally, directories with a high amount of content, both due to file numbers and file size may indicate a focus on activity in that area. Claims such as these may then be reinforced or refuted based on additional information gathered during discovery. Common to most open source communities are mailing lists and discussion forums, source repositories, community newsletters, issue repositories, and binary release sections, among others. The mere presence of these suggests certain activities in the development process. These also signal what types of data may be contained therein. If we just look at source repositories, we can obtain a process specification of a limited set of activities- those that involve changes to the code, just as issue and bug databases tell us that some testing is done on which the issue reports are based. In some communities, issue reports are also used to file feature requests. Such information may also be found within discussion forums or email lists.

The bulk of the process data is found within the content of Web artifacts. Much of the mapping consists of text matching between strings in artifacts such as web pages, and email messages and process related keywords as was demonstrated for structure-based data. In the case of web content, we are also looking for items like date stamps on email messages to place the associated events in time, document authors, and message recipients. In some cases, it is possible to uncover "how-to" guides or partial process prescriptions. Like other content, these may not accurately reflect the process as it is currently enacted, if they ever did. Therefore, each datum must be verified by others.

Usage patterns, like content size, are indicators of which areas of the Web space are most active, which reinforces the validity of the data found therein and also what activities in the process may be occurring at a given time. Web access logs, if available, provide a rich source of data. Page hit counters and last update statistics are also useful for this purpose. Work by Cadez [4] and Hong, et al [6] demonstrate two techniques for capturing Web navigation patterns, however neither can be done in a strictly noninvasive manner. The first cannot provide tours of the Web space and the latter requires members to access the community Web through a proxy server used to track trips.

OSSD artifacts vary along these three dimensions over time, and this variance is the source of process events. To effectively discover processes, our reference framework must be able to relate artifacts in the community Web space with process actions, tools, resources, and roles.

5. RESULTS

Our experiences in process discovery have shown this framework to be adequate and effective for use in discovering software development processes in OSSD communities. Nevertheless, open source communities vary drastically in size and process due to factors such as degrees of openness, product, motivations, authority structure, and more. These all affect the development paradigm and, in turn, the process and the landscape of the community Web space. The challenge in process discovery is then, determining relationships between entity instances discovered. A directory such as "x-test results" is positive evidence that some sort of testing is conducted. It is likely that the files in this directory relate to this testing. Additionally, hyperlinks in the content of these artifacts may point to other sources of testing-related evidence as indicated by the context of the reference. Detecting relationships between unlinked or indirectly linked artifacts is more challenging. These connections may be established by analyzing the context of the data collected in light of a priori knowledge of software development practices provided by the process entity classification scheme. For example, the automated XTest results report summary found in the "xtest-results/netbeans_dev/200308200100/development-unit" [11] subdirectory of the NetBeans community Web may be linked to the "Q-Build Verification Report" in the QA engineer build test subdirectory "q-builds" [10] even though there is no hyperlink to relate them by observing a match between the build numbers found on each page, which can, in turn, be matched with a binary file found on the "downloads" page [9]. This shows a relation between automated testing, manual testing, and source building efforts. Date stamps on each artifact give us a basis to assert the duration of each activity. 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 community views as more essential to development and which are peripheral.

6. DISCUSSION

Edward Averill [2] states that reference models must be a set of conceptual entities and their relationships, plus a set of rules that govern their interactions. The reference framework described above does this by defining a particular application domain, fully classifying it without prescribing how particular roles, resources, tools, and activities should be assembled, or which meta-model entities are required for a process. In doing so, the reference framework is therefore community and process model independent. It is also discovery technique independent. Though we have applied it to discovery through manual search of the community Web information space, there is nothing in the specification that restricts its application to a more automated approach to process discovery as is our goal.

The reference framework is development process independent but it is not independent of the classes of tools, agents, activities, and resources. If a new role, for example, is incorporated into the development process, it must be added to the framework in order to be found through automated discovery techniques. It is worth recalling that the resulting process model shows an example of a process instance, which is subject to variation across executions. The degree of variation between instances may indicate stability and maturity in the process, as well as showing signs of a direction of evolution.

Though we have outlined a framework for discovering software development processes an abstract open source development paradigm, it is a framework that may easily be tailored to communities with commercial-corporate influences such as NetBeans and Eclipse, as well as corporate source projects, adjusting the meta-model taxonomy in terms of tool instances, roles, etc. to suit the development paradigm.

7. ACKNOWLEDGMENTS

The research described in this report is supported by grants from the National Science Foundation #IIS-0083075 and #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; and Margaret Elliott and Walt Scacchi at the UCI Institute for Software Research are also collaborators on the research project described in this paper.

8. REFERENCES

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

[2] Averill, E. Reference Models and Standards. Standardview 2, 2, (1994) 96-109.

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

[4] 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 Proceedings of Knowledge Discovery and Data Mining, (2000) 280-284.

[5] Dinkelacker, J., Garg, P. Corporate Source: Applying Open Source Concepts to a Corporate Environment. In Proceedings of the First ICSE Workshop on Open Source Software Engineering, (Toronto, Canada May 2001).

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

[7] 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).

[8] Lowe, D., Bucknell, A., and Webby, R. Improving hypermedia development: a reference model-based process assessment method. In Proceedings of the tenth ACM Conference on Hypertext and hypermedia (Darmstadt, Germany, 1999), ACM Press, 139-146.

[9] NetBeans IDE Development Downloads Page, http://www.netbeans.org/downloads/ide/development.html.

[10] NetBeans Q-Build Quality Verification Report, http://qa.netbeans.org/q-builds/Q-build-report-200308200100.html.

[11] NetBeans Test Results for NetBeans dev Build 200308200100, http://www.netbeans.org/download/xtest-results/netbeans_dev/200308200100/development-unit/index.html.

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

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

[14] Scacchi, W. Open Source Software Development Process Model Taxonomy, (2002). http://www.ics.uci.edu/~wscacchi/Software-Process/

[15] Scacchi, W. Understanding the Requirements for Developing Open Source Software Systems, IEE Proceedings- Software, 149, 1 (February 2002). 25-39.

[16] Srivasta, J., Cooley, R., Deshpande, M., Tan, P. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data, CM SIGKDD Explorations Newsletter, 1, 2 (2000). ACM Press, 12-23.

[17] Weske, M., Goesmann, T., Holten, and R., Striemer, R. A reference model for workflow application development processes. In Proceedings of the international joint conference on Work activities coordination and collaboration (San Francisco, CA 1999). ACM Press, 1-10.

[18] Ye, Y. and Kishida, K. Toward an understanding of the motivation Open Source Software developers In Proceedings of the 25th International Conference on Software Engineering (Portland, Oregon May 2003). 419-429.

Data Mining for Software Process Discovery in Open Source Software Development Communities

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

Abstract

Software process discovery has historically been an intensive task, either done through exhaustive empirical studies or in an automated fashion using techniques such as logging and analysis of command shell operations. While empirical studies have been fruitful, data collection has proven to be tedious and time consuming. Existing automated approaches have expedited collection of fine-grained data, but do so at the cost of impinging on the developer's work environment, few of who may be observed. In this paper, we explore techniques for discovering development processes from publicly available open source software development repositories that exploit advances in artificial intelligence. Our goal is to facilitate process discovery in ways that are less cumbersome than empirical techniques and offer a more holistic, task-oriented view of the process than current automated systems provide.

1. Introduction and Beginnings

Software process models represent a networked sequence of activities, object transformations, and events that embody strategies for accomplishing software evolution [10]. Software process discovery seeks to take artifacts of development (e.g. source code, communication transcripts, and so forth), as its input and elicit the networked sequence of events characterizing the tasks that led to their development. This process model may then be used as input to other process engineering techniques such as redesign and re-engineering.

Open source software development (OSSD) communities are a rich opportunity for software process discovery and analysis with the benefit that so much of their process-relevant data is publicly available. Though many researchers have sought non-automated means of software process modeling, often there is so much information that it becomes intractable to subsume unaided, thus motivating the push for tools to assist in process discovery. In our past efforts [6], we have shown the feasibility of automating the discovery of software process models by using manual simulation of how such automated techniques might operate as a basis to substantiate that discovery and modeling of software development processes in large OSSD communities such as Mozilla, Apache, NetBeans, and Eclipse (consisting of tens of thousands of developers continuously contributing software artifacts to the community repository) is both plausible and amenable to automation. In this paper, we explore techniques for searching OSSD Web repositories for process data, relating these data in the form of process events, and assigning them to meaningful orders as a process model in an attempt to reduce the manual effort necessary to discover and model software processes.

We take, as our process meta-model, that of Noll and Scacchi [8]. Software processes are composed of events: relations of agents, tools, resources, and activities organized by control flow structures dictating that sets of events execute in serial, parallel, iteratively, or that one of the set is selectively performed.

It has been shown [6] that OSSD community Web repositories encode process data in terms of the structure of the community repository, its content, and its usage and update patterns. OSSD artifacts vary along these three dimensions over time, and this variance is the source of process events. To effectively discover a software process, we must be able capture these data and their changes. This may be done through combined application of text and link analysis techniques, as described below. We propose the use of text analysis techniques for extracting instances of process meta-model entities from the content of the community repositories, followed by link analysis to assert relationships between the mined entities in the form of process events. Next, we apply usage and update patterns to guide integration of the results of text and link analysis together in the form of a process model (see Figure 1). Finally, we conclude with addressing the knowable validity of discovered software process models and future directions for continuing work.

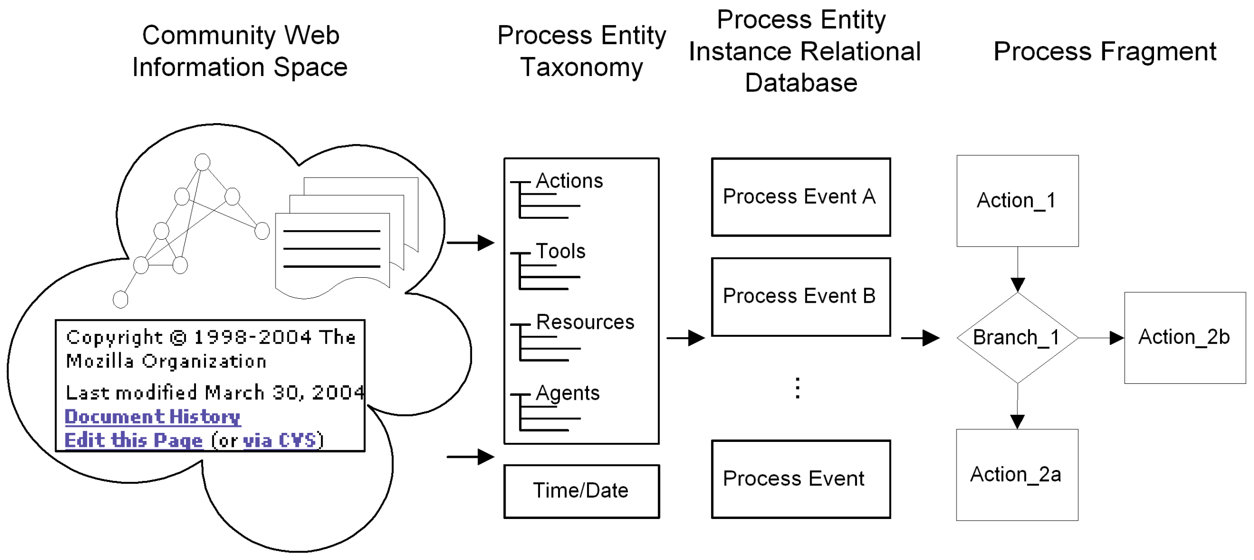

Community Web Information Space
Process Entity Taxonomy
Process Entity Instance Relational Database
Process Fragment
Copyright © 1998-2004 The Mozilla Organization
Last modified March 30, 2004
Document History
Edit this Page (or via CVS)
Actions
Tools
Resources
Agents
Time/Date
Process Event A
Process Event B
Process Event
Action_1
Branch_1
Action_2b
Action_2a

Figure 1: Web artifacts are filtered through a process entity taxonomy to extract atomic process action events, sequenced using temporal indications within the artifacts and reconstructed into a process using PRM

2. Text Analysis

The bulk of the process data is found within the content of Web artifacts. Much of the mapping consists of text extraction, matching between text strings in artifacts such as web pages and email messages and a taxonomy of process related keywords [5]. In the case of web content, we are especially looking for items like date stamps on email messages to place the associated events in time, document authors, and message recipients. This matching is done using a name recognizer.

An inherent challenge to name recognition is that many classes of lexical items we desire to recognize are open sets since we cannot enumerate all possible proper names they contain. Further, name classification suffers from synonymy and polysemy- the same concept represented using different terms, and different concepts represented using the same term, respectively. This frequently occurs between OSSD communities, using terms such as release *manager* rather than release *coordinator* to describe the same role. Fortunately, these are well known problems in text analysis and most text analysis systems provide some support for managing them. The SENSUS ontology system [3] is one such system that attempts to automate much of the domain modeling work allegedly covering most areas of human expertise. This automation is critical considering lexicographical differences across and evolution within communities.

Different types of content yield different opportunities for gathering data. Common to most open source communities are mailing lists and discussion forums, source repositories, community newsletters, issue repositories, and binary release sections, among others. The mere presence of these suggests certain activities in the development process. They also signal what types of data may be contained within. If we just look at source code repositories, we can derive a process specification of a limited set of activities- those that involve changes to the code. Similarly, issue and defect databases tell us that some testing is done on which the issue reports are based. In some communities, issue reports are also used to file feature requests. Such information may also be found within discussion forums or email lists.

Although it may seem tempting to attempt to tailor analysis of artifacts to their type (e.g. email message, defect report, etc) to capitalize on the structure of the artifact type thereby facilitating analysis. While this approach would potentially lead to increased performance in analysis of artifacts conforming to the structure expected by the artifact model, this structure varies widely between communities. To achieve high performance using artifact structure models requires development of models, not only for each artifact type in a community repository, but also for each artifact type used by all repositories under study.

It is interesting to note that we may uncover "how-to" guides or other partial process prescriptions in examining the community repository. Like all content, these may not accurately reflect the process as it is currently enacted, if they ever did. This suggests the need for probabilistic methods for modeling software development processes to filter noise within a process instance and accounting for variance across instances.

By itself, the result of text extraction gives us the raw ingredients of a process model. We look to link analysis to put these ingredients together into atomic process events.

3. Link Analysis

Text extraction allows us to ask questions such as who is collaboration with whom. From this information, we can construct a social network [Madey, et al] for the community. Social networks may identify developers that frequently collaborate, but they do not tell us what the developers are doing, and, more importantly, how they are doing it. One way to associate what and how information is through the use of probabilistic relational modeling (PRM).

Probabilistic relational modeling [4] is somewhat inspired by entity relationship modeling used to describe databases. In the classical example, we might have tables of movie actors, movies, and roles actors have played in movies and want to learn relationships between them. Conceptually, this is no different from linking process agents playing a role to complete an action (using various tools that consume and produce resources). Probabilistic relational modeling allows inference about individual process entities while taking into account the relational structure between them, unlike traditional approaches that assume independence between entities. Why is this the right approach? Software processes driven by the choice of tools used in development. Tools either dictate what and when activities are performed, or tools are selected to support desired activities, and to an extent, suggest methods of completing activities (i.e. enforce process compliance). Developer roles emerge to perform these activities and carry out supplemental work not performed by development tools. Further, process entity instances arising from text analysis have other relationships. They are related contextually to other entities in the artifacts in which they are found. They are also related to artifacts hyperlinked to those in which they are present. Such contextual relationships arising from the logical structure of the repository are also good candidates for probabilistic relational modeling. Indeed, doing so allows us to form process events whose entities span multiple artifacts.

To learn relationships between process entities, we must know the context of the entity with respect to others. This context can be represented in two ways. Extracting the URL of the artifact in which each entity is located allows us to cross-reference that entity with others in the same artifact, as well as other artifacts in which that entity is located. Additionally, if we look at the creation date of the artifact in which it was located, we may be able to intuit that those instances that are temporally distant may signal an activity of lengthy duration multiple instances of the same activity. This determination, however, is the work of usage and update pattern analysis.

4. Usage and Update Patterns

Usage patterns, like content size, are indicators of which areas of the Web space are most active, which reinforces the validity of the data found therein and also claims of what activities in the process may be occurring at a given time. Web access logs, if available, provide a rich source of data. Web page hit counters and last update statistics are also useful for this purpose.

Cadez [1] and Hong, et al [2] demonstrate two techniques for capturing Web navigation patterns, however neither can be done in a strictly noninvasive manner. The first uses server logs and cannot provide tours of the repository and the latter requires members to access the community Web through a proxy server used to track tours. Nevertheless, if we can map tours of the community Web to process events, we can get a sense of which activities are dependent on which other activities, which can be done in parallel, which sequences are done iteratively.

Fortunately, most large OSSD communities use content managing tools to perform versioning of not only product source code, but of other artifacts in the repository, as well. By analyzing changelogs we can learn the frequency of Web updates, in addition to the agent performing the update, and to some extent, the tools used to create the artifact, given its type. Work by Ripoche and Gasser [9] does this to an extent, studying defect resolution status in open source defect repositories. The approach may be generalized, extended with using the text and link analysis techniques given above, and applied to other types of artifacts, though with somewhat less precision due to the inferential nature of process entity relationship construction.

Unfortunately, revision histories are not always available. Since OSSD repositories are publicly accessible, it is possible to spider the Web repository periodically to track changes externally via *diff* tools, though information regarding the precise time of update and author would be lost. As an ethical matter, periodic spidering increases the load on the server that, for large repositories, is potentially burdensome.

By examining usage and update patterns, it is possible for us to detect process control flow structures. If we merely order then by time, the set of process events discovered is sequential. Iterations can be teased out of the sequence by considering patterns of repeated tours and updates of and to the Web. Activities being performed in parallel may also be discerned by examining non-intersecting concurrent usage and update patterns. Further, by analyzing the variance between iterations of the same task, we can identify sets of alternate activities, if the variance is small.

5. Process Model Verification

A critical question of software process discovery, regardless of automation, is how we may discern if the process discovered is a correct reflection of the process enacted by the community. The likelihood of arriving at an accurate model increases with the amount of data examined, within the limitations of the techniques applied. This is because the confidence of an asserted relationship between process entities increases with more positive instances of those relationships. Likewise, weak relationships are rejected due to insufficient evidence. At the same time, relationships between entities cannot be discovered if the entities are not in the list of process-relevant terms we look for during text extraction. Thus, the process model obtained is only as good as the taxonomy.

6. Conclusion

In this paper, we have presented a novel approach to discovering software processes from OSSD Web repositories, combining techniques for text analysis, link analysis, and of repository usage and update patterns. Though we have focused our discussion on open source repositories, given the availability of the artifacts, we believe that these techniques can be applied to closed source software repositories, and given the appropriate domain information, other types of processes, as well. Our hope is that in doing so, we may increase understanding of the process techniques that have led to their success.

7. Acknowledgments

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075 and #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; and Margaret Elliott at the UCI Institute for Software Research.

8. References

[1] 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. 2000 Knowledge Discovery and Data Mining Conference, 280-284. (2000).

[2] Hong, J. Heer, S. Waterson, and J. Landay, WebQuilt: A proxy-based approach to remote web usability testing, ACM Transactions on Information Systems, 19(3), 263-285. (2001).

[3] Hovy, E.H., A. Philpot, J.-L. Ambite, Y. Arens, J.L. Klavans, W. Bourne, and D. Saroz. 2001. Data Acquisition and Integration in the DGRC's Energy Data Collection Project. In *Proceedings of the dg.o 2001 Conference*. Los Angeles, CA.

[4] Getoor, L., Friedman, N., Koller, D., Taskar B. Learning Probabilistic Models of Link Structure, Journal of Machine Learning Research, 2002.

[5] Jensen, C. Applying a Reference Framework to Open Source Software Process Discovery. In Proceedings of the First Workshop on Open Source in an Industrial Context OOPSLA-OSIC03, Anaheim, CA October 2003.

[6] Jensen, C. and 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.

[7] Madey, G., Freeh, V., and Tynan, R. "Modeling the F/OSS Community: A Quantitative Investigation," in *Free/Open Source Software Development*, ed., Stephan Koch, Idea Publishing, forthcoming.

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

[9] Ripoche, G. and Gasser, L. "Scalable Automatic Extraction of Process Models for Understanding F/OSS Bug Repair", submitted to the 2003 International Conference on Software & Systems Engineering and their Applications (ICSSEA'03), CNAM, Paris, France, December 2003.

[10] Scacchi, W. Process Models in Software Engineering, in J. J. Marciniak (ed.), Encyclopedia of Software Engineering, 2nd. Edition, 2002.

Process Modeling

This section contains the following four chapters. The first proposes a scheme for modeling processes using low-fidelity process specifications in a form that is amenable to automated enactment. The second paper extends and elaborates the evolving structure and design of low fidelity process models. The third examines the modeling of a new class of hidden and knowledge-intensive processes associated with the recruitment and migration of people within a large OSSD project. The last provides a comprehensive examination of a multi-enterprise project network that spans and interconnects the processes of each project, as well as processes that can communicate and share knowledge-intensive work artifacts across project boundaries.

John Noll, Flexible Process Enactment Using Low-Fidelity Models. *Proceedings SEA '03*, Marina Del Rey, CA, USA, November, 2003.

Darren Atkinson, Daniel Weeks, and John Noll, The Design of Evolutionary Process Modeling Languages. *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, Korea, December 2004.

Chris Jensen and Walt Scacchi, Modeling Recruitment and Role Migration Processes in OSSD Projects, *Proc. 6th Intern. Workshop on Software Process Simulation and Modeling*, St. Louis, MO, May 2005.

Chris Jensen and Walt Scacchi, Process Modeling Across the Web Information Infrastructure, to appear in *Software Process--Improvement and Practice, 2005.*

Flexible Process Enactment Using Low-Fidelity Models

John Noll
Computer Engineering Department
Santa Clara University
500 El Camino Real
Santa Clara, CA 95053-0566
email:jnoll@cse.scu.edu

ABSTRACT
Attempts to extend process enactment to support dynamic, knowledge intensive activities have not been as successful as workflow for routine business processes. In part this is due to the dynamic nature of knowledge-intensive work: the tasks performed change continually in response to the knowledge developed by those tasks. Also, knowledge work involves significant informal communications, which are difficult to capture.

This paper proposes an approach to supporting knowledge-intensive processes that embraces these difficulties; rather than attempting to capture every nuance of individual activities, we seek to facilitate communication and coordination among knowledge workers to disseminate knowledge and process expertise throughout the organization.

KEY WORDS
Workflow Modeling, Cooperative Work Support

1 Introduction

The conventional workflow approach employs a server or execution engine that executes workflow specifications and stores documents produced by the workflows. Process participants (actors) interact with the engine through web browsers, environments, or task-specific tools, receiving guidance on what activities to perform, and how to perform them.

This approach has been successful for automating repetitive, routine processes, but attempts to extend it to support dynamic, knowledge intensive activities have not been as successful [1].

In part, this is due to the dynamic nature of such activities: actors in knowledge-intensive environments continually adapt their activities to reflect increasing understanding of the problem at hand, which results from performing the knowledge-intensive activities. Thus, the performance or enactment of knowledge-intensive work processes involves a cycle of planning, action, review, and refinement. This presents several problems for process management:

1. The activities performed in any cycle are difficult to describe in sufficient detail to be useful for conventional workflow enactment;

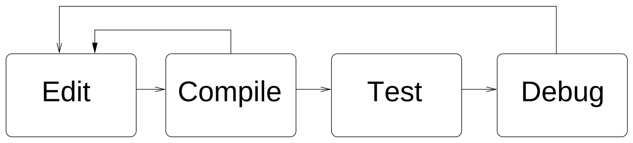


Figure 1. Edit-compile-debug process.

2. Experts perform these activities in a fluid, almost unconscious manner, rather than as discrete steps;

3. The cycle is repeated rapidly and continuously, so the set of activities evolves rapidly; therefore, any description of the process is immediately out of date.

Therefore, we propose an approach targeted to encouraging development of process expertise among knowledge workers. In this approach, actors are given high-level guidance about what activities to perform, and how to perform them, through the use of *low-fidelity* process models. These models specify a nominal order of tasks, but leave actors free to carry out their activities as their expertise and the situation dictates.

The approach comprises three key components:

1. Process specifications based on the notion of *low fidelity* process models;

2. A distributed process deployment and execution mechanism for enacting low fidelity process models;

3. A *Virtual Repository* of artifacts providing access to distributed collections, repositories, and databases of information objects related to the work to be performed;

The following sections discuss, in turn, the modeling approach based on low-fidelity models; coordination among concurrent processes; enactment based on low-fidelity models; related work; and, some conclusions.

2 Process Modeling

Our previous work with process modeling demonstrates the value of *low-fidelity* models for documenting and analyzing knowledge-intensive work [2, 3]. A low-fidelity

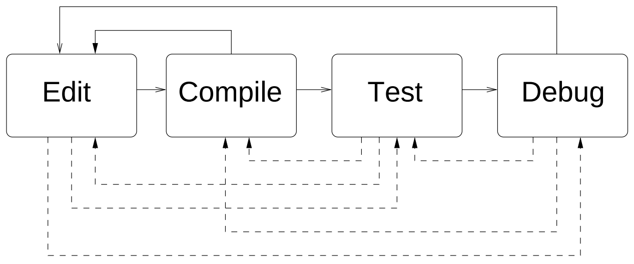


Figure 2. Edit-compile-debug process, augmented.

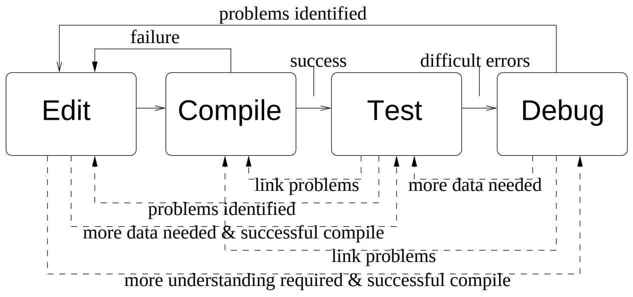


Figure 3. Edit-compile-debug process, augmented.

model does not seek to capture every detail and nuance of a knowledge-intensive process; rather, it documents the major activities of a process and the primary sequence in which they are performed.

An example of a model depicting software development is shown in Figure 1. This model shows the nominal sequence of activities involved in turning a design into working code: the programmer enters source code text using an editor, then compiles the code, iterating over these steps until the code compiles successfully. Then, he or she proceeds to test and debug, iterating over the whole sequence to fix the failures uncovered during testing.

This model captures both the important activities in code development, and the main sequence, and is thus useful for discussing the programming process. But it does not begin to capture all of the possible transitions between activities. Many experienced programmers switch frequently between debugging and editing, delaying the compilation step until several faults have been fixed. Occasionally, it is necessary to iterate over the compile-test-debug cycle; sometimes, a programmer will skip debugging and proceed directly back to editing. Figure 2 shows these additional transitions, represented by dashed edges.

While this depiction is more complete, in that it represents all of the plausible transitions between tasks, it is not entirely accurate. For example, although the graph shows a transition from “Edit” to “Debug”, it is not possible to take this transition until the “Compile” step has been successfully completed at least once: “Debug” requires an executable program, which is the output of the “Compile” step. Figure 3 shows these conditions as labels on the edges.

However, it’s not clear that the model depicted in Figure 3 is more useful than that in Figure 1; as a guidance tool, a novice programmer might find the numerous transitions confusing, while an expert would already know that these additional transitions are possible.

Our modeling approach is based on the notion of *low-fidelity* process models. A low-fidelity model seeks to capture the essence of a process, while abstracting away as many details as possible. The modeling language allows the modeler to capture both the nominal control flow (the solid edges in Figure 1), and the conditions that constrain transitions outside the nominal flow (the dashed lines in Figures 2 and 3).

Figure 4 shows a specification of the process of Figure 1, written in the PML process modeling language. The nominal control flow is represented explicitly by the *iteration* constructs and the ordering of *actions* in the specifications. The constraints on other transitions are expressed by the *provides* and *requires* statements. These are predicates that express the inputs and outputs of each step (*action*) in the process, and thus the pre- and post-conditions that exist at each step in the process. Note that this simple specification captures the constraint that testing and debugging cannot proceed until compilation is successful: the “Test” and “Debug” actions *require* a resource called “exec”, which is produced (*provided*) by the “Compile” action. Thus, until this action succeeds, “Test” and “Debug” are not possible.

Modeling processes using low-fidelity models yields several benefits:

- Low-fidelity models are easy to specify, and can be generated rapidly.
- A low-fidelity model still captures the essential facets of a process, especially the resources consumed and artifacts produced by a given set of activities.
- Because they seek to represent only high-level detail, low-fidelity models are relatively stable; that is, they continue to be accurate descriptions of the high-level process, even as the details of process activities evolve in response to knowledge and experience gained with the problem.

2.1 Modeling Coordinated Activities

The development process of Figure 1 does not exist in isolation. The input to the process is a set of requirements, and the output is debugged source code. Other processes produce and consume these resources, as depicted in Figure 5. This figure shows two cooperating processes: the development process of Figure 1, and a parallel test development process that ultimately applies a test suite to the debugged code.

These processes cooperate to produce a tested product: both start with requirements to develop their respective

```
process edit-code {
    iteration {
        iteration {
            action Edit {
                requires { design }
                provides { code }
            }
            action Compile {
                requires { code }
                provides { exec }
            }
        }
        action Test {
            requires { exec }
            provides {
                code.status == "unit tested"
            }
        }
        action Debug {
            requires { exec }
        }
    }
}
```

Figure 4. Software Development Process Fragment.

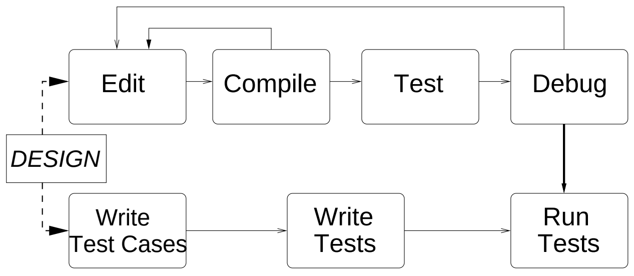


Figure 5. Coordinated processes.

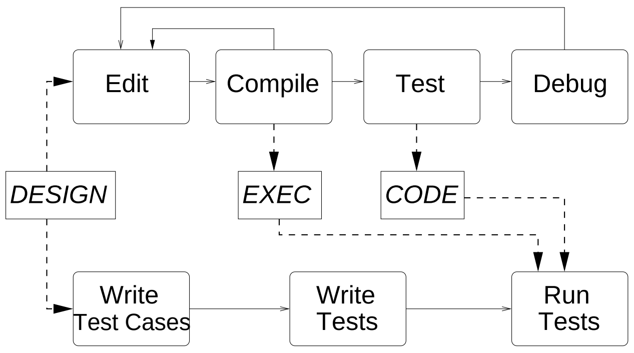


Figure 6. Resource flow between processes.

artifacts. In addition, the test process needs the output of the development process (the executable object) to run the tests. This dependency could be represented by an explicit link between the "Debug" and "Run Tests" actions (represented by the solid edge in Figure 5). But this approach has several difficulties.

First, it creates an explicit connection between the specific development process and the test process that does not always exist: developers could employ any of a number of different development processes (clean-room, test-first, even "chaotic" development) that could provide the object for testing.

Second, it requires both processes to be maintained as a single model, which is often not the case: different organizations are responsible for their respective processes which they develop and maintain independently.

Finally, it doesn't capture the true relationship between the processes: typically, testers require a compiled product to test, so that the actual relationship is between "Compile" and "Run Tests" as opposed to "Debug" and "Run Tests." But the compiled product must also have been debugged and tested. The essential relationship between the two processes is that the development process produces an executable product for the test process to test (Figure 6). It's not important to the test process how this product was developed, only that it exists in a state suitable for testing.

This relationship is easy to represent in PML, as shown in Figure 7. This specification shows that the beginning of the test process depends on the availability of the design document ("DESIGN"). More important, the "Run Tests" action cannot begin until the three conditions are met: the "Write Tests" action must complete, there must be an executable to run, and the code must be unit tested.

Note that because the processes are indirectly coupled, it is not necessary for all activity to be modeled, or enacted; enacted processes can be coordinated through a shared resource with ad-hoc work or activities in another organization. Thus, the "Run Tests" task can begin as soon as the "exec" and "code" resources are available; but these resources can be produced by any process, including a completely spontaneous ad-hoc process.

3 Flexible Enactment

Enactment is driven by events, which are classified as either *process* events or *resource* events. These are summarized in Table 1. A predicate in the process description specifies the state that the required resources must satisfy before the activity can proceed (the mechanism is described fully in [4]).

Process events signal action initiation and completion, and are generated directly by actors as a consequence of performing tasks. Resource events reflect changes in the environment, such as creation, deletion, and modification of resources, and time events such as deadlines or alarms.

The enactment mechanism enables flexible enactment through the way it handles process and resource events.

```
process test-code {
    action WriteTestPlan {
        requires { design }
        provides { test_plan }
    }
    action WriteTests {
        requires { test_plan }
        provides { test_suite }
    }
    action RunTests {
        requires { test_suite && exec
        && code.status == "unit tested" }
        provides { code.status == "tested" }
    }
}
```

Figure 7. Coordinated Test Process.

Process Events	
Create process	An actor requests instantiation of a new process instance.
Task Start	An actor has begun a task.
Task Suspend	The actor has suspended an active task.
Task Complete	The actor has completed a task.
Task Abort	The actor aborts a task that can't be completed.
Resource Events	
Object Creation	A new resource has been created (or detected).
Object Modification	An existing resource has been changed.
Object Deletion	An existing resource has been destroyed or removed.
Deadline	A time event (deadline, milestone, alarm) has passed.

Table 1. Process and Resource Events

Events affect process state in several ways:

1. Activation of the next task in the nominal control flow. The process description specifies the order in which tasks should be performed. A process event can trigger the transition from one task to the next; for example, the completion of the Edit task in Figure 1 causes transition to the Compile task.

2. Activation of other actions in the process. In Figure 4, completion of the “Compile” task *may* cause “Test” and “Debug” to become ready, if the “exec” object is successfully created.

3. Activation of actions in other processes. For example, when both the “Compile” and “Test” tasks of Figure 4 are successfully completed, resulting in products in the correct state, the “Run Tests” task in Figure 7 becomes ready.

The completion of an action can make a number of additional actions available for the actor to perform: a process event signals the completion of an action, which makes the next action in the nominal control flow available. In addition, the completion of an action may include the creation or modification of one or more resources as side-effects, generating resource events that make additional actions available. The enactment engine can then recommend that the next available action in the nominal control flow should be performed, and also show other actions available as the result of resource events.

As an example, suppose an actor has just completed the “Compile” action in Figure 6. This process event causes the “Test” action, which is the next action in the nominal control flow, to become available. The “Compile” action also creates an executable object (“EXEC” in Figure 6), a resource event that satisfies the “Debug” action's *requires* predicate (Figure 4). As a consequence, the “Debug” action also becomes available. Finally, the creation of the executable object satisfies part of the “Run Tests” action's *requires* predicate. This may cause “Run Tests” to become available as well. The result is several actions ready for the actor to perform, representing different process performance paths.

The enactment mechanism is depicted in Figure 8. The *User Interface* allows actors to generate process events, and invoke Tools to perform tasks. The *Process Engine* handles Process and Resource events, responding as described above. It includes a virtual machine to interpret PML process specifications to compute the state of actions in response to a process or resource event. The *Virtual Repository* is responsible for detecting resource events, and translating them into a representation-independent form suitable for interpretation by the *Process Engine*.

The Virtual Repository provides a logically centralized view of a set of distributed, heterogeneous information repositories. This enables the enactment mechanism to treat a variety of resources, such as email messages, Web

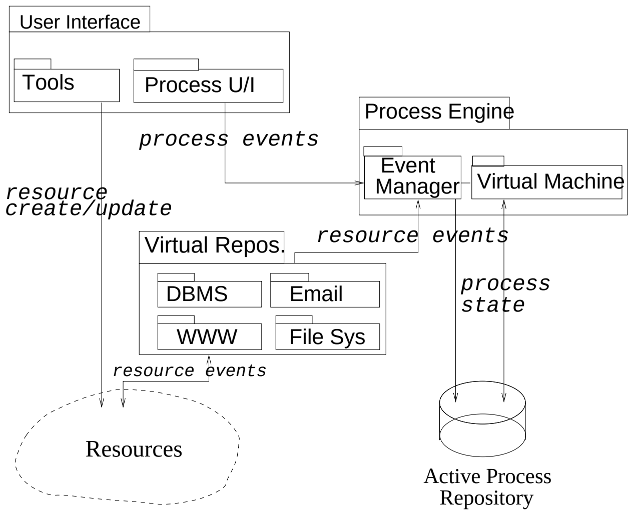


Figure 8. Process Enactment Architecture

pages, documents, etc. as information objects with a uniform format and access interface.

4 Related Work

There are three main approaches to enacting dynamic processes in a flexible manner.

The first treats deviations from the specified process as exceptions. This view supposes that there is a usual or "right" way to do things, from which deviation is occasionally required. For example, Milano augments enactment of Petri-net based models with the ability to jump forward or backward across several transitions in order to handle exceptions to the specified flow of control [5]. The philosophy of Milano is similar to the PML approach: use simple models to specify processes, and handle variation at runtime. However, Milano treats deviations as exceptions, rather than viewing them as alternate but normal variations as PML does.

The second approach models the process as a set of constraints; as long as the constraints are met, actors are free to perform tasks as they see fit, in the order that seems most appropriate. For example, Glance and colleagues propose a constraint specification language that can be used to specify the goals of a process, without specifying the order [6].

This approach provides a great degree of flexibility: as long as the goals are met, the actor is free to do any task in any order. However, the flexibility comes at the cost of guidance: it becomes difficult to advise the novice actor as to what step to perform at a given point in time. Dourish addresses this issue by adding constraints about the order in which tasks must be performed, if such an order is necessary [7].

Related to this approach are process-centered programming environments that employ rules to specify processes; examples include Merlin [8], which is based on Prolog, and Marvel [9].

The SPELL modeling language includes constraints on task pre- and post-conditions that are interpreted by the EPOS execution environment's planning mechanism to develop a sequence of activities to suit a specific situation [10]. While SPELL's constraints are similar to PML's *requires* and *provides* predicates, they are used to develop a specific sequence of tasks at runtime. In contrast, PML's predicates allow the actor to deviate from the nominal sequence when necessary.

The third approach attempts to model the process using specifications that inherently allow great flexibility in how tasks are performed.

For example, Bernstein proposes a hybrid model that combines constraints when processes are not well understood, with stronger control flow based specifications when the process matures [11]. This approach assumes that variation from the specification is a matter of process immaturity that will decrease over time as the process becomes better understood.

Jorgensen argues for enactment based on a dialog with the actor at runtime [12]. In this approach, processes are initially specified at a high level, and enactment takes place as a dialog between the actor and the enactment mechanism. Jorgensen also views variation as a side effect of process immaturity, so the goal of this dialog is gradual refinement of the process specification into a detailed model.

5 Conclusion

The approach described herein has several distinctive features.

First, low-fidelity process models specified using PML are both straightforward and enable flexible enactment.

Second, because the coupling between coordinating actors is indirect, through a shared resource, there is no requirement for trust (or even awareness) between coordinating peers. This enables extremely fluid, dynamic organizations in which participants can join at will without requiring administrative approval or action.

Finally, by relying resource events, process enactment can proceed in the background, out of the way of experienced users until they need explicit advice.

Acknowledgements

This work is supported in part by the National Science Foundation under Grant No. IIS-0205679, through subcontract from the University of California, Irvine; and by an IBM Faculty Research Grant awarded by Santa Clara University. No endorsement is implied.

References

[1] Paul Dourish. Process descriptions as organizational accounting devices: The dual use of workflow technologies. In *Proceedings of the 2001 International ACM SIGROUP Conference on Supporting Group Work*, pages 52–60, Boulder, Colorado, USA, October 2001. ACM Press.

[2] Walt Scacchi and John Noll. Process-driven intranets: Life-cycle support for process engineering. *IEEE Internet Computing*, September/October 1997.

[3] John Noll and Walt Scacchi. Specifying process-oriented hypertext for organizational computing. *Journal of Networking and Computer Applications*, 24(1), January 2001.

[4] John Noll and Bryce Billinger. Modeling coordination as resource flow: An object-based approach. In *Proceedings of the 2002 IASTED Conference on Software Engineering Applications*, Cambridge, Massachusetts, USA, November 2002.

[5] Alessandra Agostini and Giorgio De Michelis. A light workflow management system using simple process models. *Computer Supported Cooperative Work*, 9(3-4):335–363, August 2000.

[6] Natalie S. Glance, Daniele S. Pagani, and Remo Pareschi. Generalized process structure grammars (GPSG) for flexible representations of work. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, pages 180–189, Boston, Massachusetts, USA, 1996. ACM Press.

[7] P. Dourish, J. Holmes, A. MacLean, P. Marqvardsen, and A. Zbyslaw. Freeflow: Mediating between representation and action in workflow systems. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, pages 190–198, Boston, Massachusetts, USA, 1996. ACM Press.

[8] Burkhard Peuschel and Wilhelm Schäfer. Concepts and implementation of a rule-based process engine. In *Proceedings: 14th International Conference on Software Engineering*, pages 262–279. IEEE Computer Society Press/ACM Press, 1992.

[9] Israel Z. Ben-Shaul, Gail E. Kaiser, and G. Heineman. An architecture for multi-user software development environments. *Computing Systems, the Journal of the USENIX Association*, 6(2):65–103, 1993.

[10] R. Conradi, M. L. Jaccheri, C. Mazzi, M. N. Nguyen, and A. Aarsten. Design, use and implementation of SPELL, a language for software process modelling and evolution. In J.-C. Derniame, editor, *Software Process Technology*, number 635 in Lecture Notes in Computer Science, pages 167–177. Springer-Verlag, 1991.

[11] Abraham Bernstein. How can cooperative work tools support dynamic group process? Bridging the specificity frontier. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, pages 279–288, Philadelphia, Pennsylvania, USA, 2000. ACM Press.

[12] Havard D. Jorgensen. Interaction as a framework for flexible workflow modelling. In *Proceedings of the 2001 International ACM SIGROUP Conference on Supporting Group Work*, pages 32–41, Boulder, Colorado, USA, October 2001. ACM Press.

The Design of Evolutionary Process Modeling Languages

Darren C. Atkinson, Daniel C. Weeks, and John Noll
Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566 USA
{datkinson,dweeks,jnoll}@scu.edu

Abstract

To formalize a software process, its important aspects must be extracted as a model. Many processes are used repeatedly, and the ability to automate a process is also desired. One approach is to use a notation that already exists, such as a programming language, and extend it. However, the intricacies and restrictions the programming language places on the ability to succinctly and clearly describe a process can be problematic. An alternative approach is to develop a language specifically for describing processes. A significant disadvantage of this approach, however, is the lack of tool support for ensuring model correctness. We discuss a high-level language that encourages evolutionary model development and describe a tool for performing model verification. We have used our language and tool on the NetBeans model for distributed software development.

1. Introduction

Process descriptions [12] characterize the important aspects of processes from which models can be derived. One purpose of a model is to reflect the control-flow of the process without incorporating nonessential properties. The objective of modeling is not to recreate every minute aspect of the process, but instead to extract the meaningful properties of the process and imitate its behavior [1].

In addition to disambiguating a complicated process, having a written notation for process description provides the ability both to analyze the process by checking for errors and to automate it. Validating a process before enactment increases quality and ensures correctness. Automation increases efficiency and provides the facility to guide the process through its life-cycle, only stopping for human interaction when necessary.

In order to effectively check models for errors and to automate processes, a *formal* notation (i.e., language) is required to specify the model. The objective of the language is to be as expressive as an unstructured description, but changing the representation so it is unambiguous [4]. The design of the language constrains how and where the process model can be applied. If the language is too complicated or strict, it may not be expressive or flexible enough to be useful in a broad range of applications. If the language definition is too loose, it may not be amenable to meaningful analysis or automation.

In addition to finding problems in a process, modeling allows the process designers to explore many different designs before enactment. Complex processes may be too costly to actually implement and refine. Modeling allows the modeler to easily modify the process and determine if the changes are effective. Furthermore, processes are typically designed starting with abstract concepts and are iteratively refined into detailed descriptions. Therefore, the language used to describe a process needs to reflect this *evolutionary* development cycle, but still provide valuable information about the process at every level of abstraction. Finally, if the conceptual and procedural aspects of a process can be represented in a language, then tools can be designed to automatically check the models before enactment [8].

One common paradigm for modeling processes is rule-based or logical modeling [10, 13]. This method relies on rules to describe the tasks and then generates a model from the dependencies specified in the tasks. The main advantage of this approach is that the modeler need only specify individual tasks, and the associated tools will automatically generate a model with consistent dependencies.

The most obvious problem with this method is that the modeler has difficulty controlling the order of tasks in the process. If two steps are independent, but the modeler wants them to be performed in a sequence, then a false dependency must be introduced in order to achieve the desired results, which adds an unnecessary layer of complexity. We feel this method is also counterintuitive to how people think about processes. The order in which tasks are performed is a primary concern when defining a process, and the modeler should be able to control it.

The paradigm that we advocate is control-based [3, 19], which resolves many of the problems inherent in the rule-based paradigm. In this approach, the control is specified by the modeler, which allows her to describe the flow of control in the process. This method can be used to model abstract processes, detailed processes, and every layer of abstraction between the two [11]. At a high level of abstraction, the control is sequential, which allows the modeler to imply the dependencies without actually having to specify them. If it is later decided that the model should be more specific, the actual dependencies can be introduced. This method is more intuitive and reflects the steps that humans normally take.

2. Modeling language design

2.1. Goals and motivation

Although there are many different approaches to process modeling, there is a general consensus about the goals of modeling languages [15]. These goals embody how a language should capture the aspects of a process in order to represent the process properly. The most common goals are:

- *simplicity:* a non-technical person should be able to model a process without being encumbered by the syntactic or semantic requirements of the language
- *flexibility:* the language can be applied to a variety of applications such as business processes, software processes, or any other form of process equally
- *expressiveness:* the ability to accurately reflect a process is essential in order to extract useful information about the process
- *enactability:* the language should enable the model to mimic the actual execution of a process

Though these goals have been repeatedly defined and examined, many language implementations disregard these goals in an effort to achieve additional functionality [6].

The most common approach to designing a process modeling language is to build the language on top of an existing programming language, because of similar concepts and notations in both languages. A typical example of this approach is the language APPL/A [17, 18], which is designed as an extension to the programming language Ada.

There are many advantages to using this *bottom-up* approach to language design, most of which pertain to enactability. APPL/A was able to take advantage of features such as concurrency, iteration, modularity, information hiding, and exception handling that are integrated into Ada. In addition, the existing compilers provide type checking and error checking capabilities. Other modeling constructs such as relations and tasks were effectively implemented using Ada constructs of packages and tasks. Despite numerous advantages, there are some fundamental concerns raised by this language design approach.

Though the use of an underlying programming language has obvious benefits, it compromises many of the goals of modeling languages. Ada 95 has a rich and varied syntax, resulting in a modeling language that is complex, not simple as intended. A person modeling a process may not need to know the meaning of each keyword in Ada, but they must recognize them in order to avoid using them. Using a keyword unintentionally could result in checking errors caused by the lower level language that would confuse the modeler because they have no relevance to problems in the actual model. This places an additional burden on the modeler to understand aspects of both languages.

Building on a programming language also limits the expressiveness of the modeling language. Process-related activities may not be expressible in the underlying programming language and therefore cannot be expressed in the modeling language. Ada has a requirement that concurrent tasks that need to communicate must synchronously meet, which is called a *rendezvous*. This constrains the expressiveness of the modeling language (i.e., APPL/A) because asynchronous activities exist in processes. The primary concern with this limitation is that it is not a problem with the modeling language. Instead, it is a limitation imposed by the language on which it was developed. Additionally, how the process will be enacted depends on the underlying programming language and how the process will execute once compiled. What may seem intuitive at the modeling level, may not be reflected at the programming language level.

Existing tools that support the language can also be problematic. The error checking capabilities of the Ada compiler are designed for checking errors in computer programs. However, the errors that can occur in a process are based on a different criteria than those of programming languages. While compilers are designed to examine programs for static errors, processes are dynamic in nature and many of the useful features of static checking, such as type checking, are not essential for process models. This limitation is not due to an oversight in the modeling language design, but a conceptual difference between processes and programs.

The primary concern of this style of language design is that it relies on a language paradigm that is not explicitly designed for process modeling. Programming languages are designed for computation, and their target applications are not the same as those of process modeling languages. Though there are many similar concepts between programming and process modeling, there are subtle differences that separate the two. For example, although a program may be evolved, at each step the program must be semantically correct. In process modeling, the modeler may want to experiment with partially correct models to obtain feedback. Therefore, a new language design is needed that focuses on and represents the concepts of process models rather than relying on programming languages, which are a product of a different research domain.

2.2. Our approach

Instead of using existing languages to reflect processes, we examine the requirements of process modeling languages and design a language based on those principles. The result of our approach is the modeling language PML [2, 14], which intrinsically supports process-related concepts rather than implementing them in terms of concepts from a programming language.[1]

This *top-down* approach to language development has many advantages that address problems inherent in the bottom-up approach. PML is a simple language with only thirteen keywords. This design decision has many implications: the language is much easier to learn, which makes it more attractive to those who do not have a background in programming. Another positive aspect of PML is that the syntax is very straightforward and not impacted by that of a programming language. In PML, statements all follow a simple form that helps to eliminate the confusion of complicated grammars. The only consideration is about what statements can be nested inside other statements, but nesting is generally shallow.

Another problem that this language design addresses is the difficulty in achieving the right level of expressiveness. Modeling languages built on programming languages are restricted by the underlying language, but not having that restriction allows for a much more adaptable grammar. PML incorporates a language construct called a *qualifier*, which is not a keyword in the language, but is a user-defined specification that enumerates the characteristics or qualities of a resource, which allows the modeler to emphasize, constrain, or modify resources in the process model.

This design approach also makes the process model easier to enact. Instead of relying on a compiler to generate code that is later executed to enact the process, the actual process model can be enacted. PML employs an enactment environment to interpret and enact the process model and is designed specifically for execution of process models. Therefore, it understands how to handle process related activities as opposed to a program that is designed for computation. For example, the environment can rely on the user in making decisions regarding which action to perform next.

PML is also quite flexible in that it supports evolutionary model development. A high-level process model can be written easily and modified iteratively until the desired level of detail is obtained. To illustrate this ability, we present the syntax for PML in the context of the evolutionary development of a process model to describe the traditional waterfall model of software development.

1 The name PML is an acronym, originally for "Process Markup Language." The language quickly evolved to resemble a programming language, but the PML name was retained for historical reasons.

3. The PML language

3.1. Language fundamentals

The most fundamental component of a process is a task or action, which are terms that can be used interchangeably. The PML syntax for an action is:

```
action identifier { ... }
```

With just the process and action statements it is possible to make a non-trivial model of the waterfall process:

```
process waterfall {
  action analyze { }
  action design { }
  action code { }
  action test { }
}
```

This high-level description provides information about what steps need to be completed and the order in which they should be performed. Though there is little detail about any of these steps, the model has enough information for a basic understanding of the waterfall development process.

3.2. Resources and attributes

Resources are an essential component to creating a process model that does more than just reiterate the steps in a process. The ability to describe the flow of resources allows the modeler to recreate a variety of dependencies that occur within a process. The only postulate for an action is that the resource is available when the process enters or exits the action. PML allows actions to require and provide resources, which reflects the action's need for or the production of a resource, but gives no indication of its origin or destination. Using these constructs, we can modify our model to provide more information about the internals of an action:

```
action analyze {
  requires { function && behavior && interface }
  provides { requirements && analysis_documentation }
}
```

This statement illustrates the conditions that must be met for this action to be performed and to terminate. Entrance to the action is not possible unless the function, behavior, and interface are available and exiting is not possible without requirements and analysis documentation. Using these predicates, a modeler can reconstruct the dependencies that exist within a process by specifying its pre- and postconditions.

In most cases, resources alone are not enough to provide the detail needed for an accurate model. While many actions in a process may require a resource, there are specific qualities or characteristics of the resource that are essential and cannot be described by the resource's name. We previously stated that the action analyze:

```
provides { analysis_documentation }
```

However, introducing a new resource to describe the fact that the analysis portion of the documentation is complete complicates the process. Without being able to modify the properties of a resource, a new resource needs to be created to describe any change in the process. Therefore, we provide attributes to solve this problem by describing the state of a resource and thus it would be more clear to state:

```
provides { documentation.analysis }
```

While analysis_documentation is an abstract resource created to describe the result of an action, documentation is a concrete resource that will persist throughout the process as new sections of the documentation are added. Attributes provide a means to describe changes to resources without having to create spurious resources.

Finally, attributes alone cannot always adequately describe specific qualities and states of resources or their properties. Actions often rely on attributes having specific values and as the model evolves and detail is added, constraining the state of resources and attributes provides more explicit control. By adding expressions the model transitions to another level of detail and can represent state:

```
provides { documentation.analysis == ''complete'' }
```

This statement is an assertion regarding the state of the attribute of a resource, and does not affect the value of the attribute. The enactment environment simply ensures that the attribute has the correct state when the action terminates. Such level of detail can be gradually added to further specify or constrain the model.

3.3. Control constructs

PML has four mechanisms for describing the control of a process. These control-flow constructs reflect process-related activities and describe the ordering of steps in a process.

3.3.1. Sequence. A sequence is the most basic form of control and is the default control mechanism when nothing else is specified. The actions in a **sequence** construct are performed in the order that they are specified:

```
sequence {
  action first { }
  action second { }
  action third { }
}
```

This construct is the most natural and intuitive form of control for a process. When one thinks about performing any process, a simple sequence of steps to accomplish the final goal is often the easiest representation.

3.3.2. Iteration. A condition that occurs quite frequently within processes is the need to repeat certain steps. While iterating over these steps, there are two concerns that must be addressed: when to go back and repeat the steps, and when to stop repeating and continue the process. Generally, this decision is handled by an expression that is evaluated to determine if the steps need to be repeated. This method works well if the number of repetitions is known when the loop begins, but the dynamic nature of a process often results in this information being unavailable. An example of this non-deterministic nature processes is making a cake where the instructions state: add flour, stir mixture, test for consistency, and *repeat* until mixture is thick and consistent, which is clearly subjective. The syntax for an **iteration** follows the same structure as a sequence:

```
iteration {
  action first { }
  action second { }
  action third { }
}
action post { }
```

When determining which path to take in PML, the predicates of the first action in the loop, first, and the first action following the loop, post, are the points of interest. When the last action in a loop is complete, the loop determines how to proceed based on whether or not the requirements in the first action of the loop and the first action following the loop are satisfied.

At first this decision procedure may appear to be inconvenient because processes may need to wait for a human to choose the proper path, but it actually allows the process to be more dynamic by providing multiple options when they exist and suppressing them when only one path is available. Also, there are many conditions in processes that are based on human judgment and cannot be evaluated by a machine.

3.3.3. Selection. Selecting one of many paths requires that a decision be made about which direction to take. The **selection** construct in PML defines possible paths of execution with only one being performed:

```
selection {
  action choice_1 { }
  action choice_2 { }
  action choice_3 { }
}
```

The decision procedure for determining which path to take is handled in a similar manner to iterations. In this case, the predicates of the first actions in each possible path are the focus.

This type of decision in process models cannot always be automatically determined and therefore may rely on human interaction to choose which path to take. Though it is possible to simply choose the first available path, therefore avoiding human interaction, there might be external considerations about which path should be taken that an automatic procedure cannot foresee.

3.3.4. Branch.

The **branch** construct specifies a set of concurrent actions within a process:

```
branch {
  action path_1 { }
  action path_2 { }
  action path_3 { }
}
```

Concurrency is usually employed as an optimization, which is generally performed implicitly and does not have a decision procedure associated with it. Each path must be performed, which removes any need for human interaction related to control. Unlike APPL/A, PML does not restrict the way that a rendezvous is handled. Instead, a PML interpreter must decide whether a synchronous or asynchronous rendezvous is used. We realize that this introduces an ambiguity as to what will actually happen at a rendezvous, but processes do not adhere to the strict nature of programming languages and the dynamic nature of processes requires that the decision be left to the modeler. Of course, artificial constraints in an asynchronous implementation can be introduced to recreate a synchronous rendezvous.

The waterfall model states that testing should be done after the code is written, but writing tests is often started at the same time as coding, so that tests can be prepared as the code is written rather than having to wait until the code is complete. To represent this we can change our model to:

```
branch {
  action code { }
  action write_tests { }
}
```

3.4. Advanced language features

Though attributes and expressions provide methods for describing properties and states of resources, not every quality of a resource can be expressed in this manner. There are aspects of a resource that are extrinsic to the resource and apply to how the resource is handled, modified, and restricted. For example, consider an action in a model that requires both design and funding. This action has two requirements that consist of some tangible resource. The design is an inexhaustible resource in that it can conceivably be used over and over again without losing any of its substance or quality. However, funding is exhaustible and can only be used until the funding is gone. Some languages provide keywords associating a resource with being consumed by an action [13]. Though adding keywords will make modeling a specific situation, such as this one, much easier, there are many possible situations that cannot be conceived of while designing the language. Furthermore, adding a language construct to clarify how each situation should be handled explicitly violates our goal of simplicity and the expressiveness of the language would rely on how many situations we could envision.

A similar problem occurs when creating a new resource. Providing more information about how a resource was created is not possible with the basic language constructs of PML. For example, code does not spontaneously appear in the coding stage, but is derived from the design, but it is not possible to illustrate this quality of the code without additional levels of specification.

To alleviate these problems, PML has a construct called a *qualifier*. The qualifier is used to describe characteristics or qualities of a resources that are beyond the scope of the language's regular syntax. With this construct we can state:

```
(partially_consumed) funding
```

In this example, partially_consumed is a user-defined quality of the resource funding. This language feature also supports multiple layers of qualifiers, such as:

```
(new) (generated) executable
```

With this construct, the model can better represent the process, but there are some difficulties associated with using a qualifier. For the process to be enacted, the environment must understand how to handle the qualifier if it has a direct impact on the execution of the process. This means that additional functionality must be provided to interpret the meaning of a qualifier, otherwise it will be ignored.

Using the language features of PML, we present a detailed, idealized waterfall process model in Figure 1. This example is one of many possible models of the waterfall development process. Even this model can be refined to include more detail to meet the needs of the person performing the process, such as adding scheduling, funding, and project-specific information. However, this model can be applied to any waterfall development process without modification because it is at a high enough level to describe the general process, but low enough to capture the essential control and resources.

We started with a simple model to describe the waterfall process. By adding resources, attributes, expressions, and finally qualifiers, we gradually introduced more and more detail to make the original model more specific. At any point in this evolution, we could have stopped and used the existing model. For example, even the first model presented that consisted solely of actions is enactable. We feel that this type of evolutionary development of models reflects the top-down way in which people reason about and describe most processes, at least at an initial, conceptual level.

4. Model verification

4.1. Tool motivation

Using a process modeling language to recreate an actual process is a complex procedure because the modeler must extract important information about tasks, resources,

```
process waterfall {
  action analyze {
    requires { function && behavior && interface }

    provides { requirements }
    provides { documentation.analysis == ''complete'' }
  }
  action design {
    requires { requirements }
    requires { documentation.analysis == ''complete'' }

    provides { design }
    provides { documentation.design == ''complete'' }
  }
  branch {
    action code {
      requires { design }
      requires { documentation.design == ''complete'' }

      provides { documentation.code == ''complete'' }
      provides { (derived) code && (new) executable }
    }
    action write_tests {
      requires { requirements && design }
      requires { documentation.design == ''complete'' }

      provides { test_cases }
    }
  }
  action test {
    requires { code && test_cases && executable }

    provides { code.tested }
  }
}
```

Figure 1. An elaborated waterfall model.

and control in such a way that the model will properly reflect the process. Consequently, the resulting model often contains errors that can be attributed to two sources: the process and the modeler. Errors that are contained within the process are problematic in that they represent some inefficiency or mistake in the process that could result in any number of problems including slow performance or even preventing the process from continuing after it reaches a certain point. Problems introduced by the modeler represent human error by either improperly representing the process, or making a typographical error that has repercussions throughout the model. With the help of tools that look for these errors, models can be more efficient and accurate.

We noted that modeling languages implemented using programming languages have inherited tool support for checking errors in models, but these tools are not specifically designed for process-related errors. Compilers perform type-checking, look for undeclared variables, and check for other syntactic errors. The problem with using these methods is that they do not represent the kind of errors that occur in a process model. Therefore, we need to explore the types of errors that might occur in a process and how they would be reflected in a model.

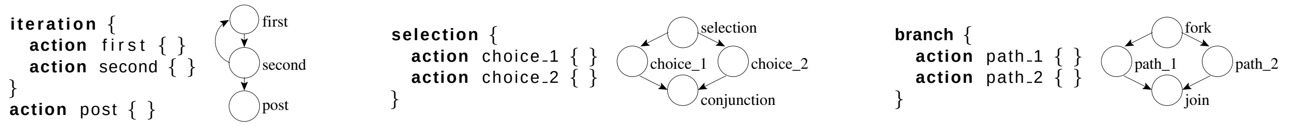
In evolutionary process modeling, any errors are usually related to the many levels of abstraction that the model must pass through before arriving at a detailed representation of the process. The first level of abstraction in a model is a list of tasks that must be performed. However, at this level, the errors that can be introduced by a modeler are simple and include problems such as syntax errors or typographical mistakes.

Transitioning to a lower level of abstraction incorporates adding resources to the model which begins the development of dependencies and may result in a considerable number of errors related to modeling. If the name of a resource is misspelled and another step in the model needs that resource, the dependency will be broken because the task was expecting the resource to have a different name. A modeler might also forget to state that a step has requirements or that it provides something. These types of errors manifest themselves as broken dependencies and extraneous steps in the model. Similarly, if a modeler fails to note what a step requires, but does note what it produces, then it appears that the step is creating some resource out of nothing. Though some steps in a process may only rely on abstract concepts or ideas that would not be properly represented by a requirement, this type of mistake is generally a problem that is introduced as an oversight. The same type of concern is raised when a step requires resources but a product for the task is not specified.

Dependencies at low levels of abstraction have a direct impact on the control of the process, which can lead to difficulties in trying to satisfy both control flow and dependencies put in place by the modeler. If the modeler wants to specify that two steps in the process are concurrent, but unintentionally creates a dependency that would prevent concurrency, such as having the first concurrent thread rely on a product of the second concurrent thread, then the model would not represent the real process. This type of error is the result of either not understanding the dependencies of the process or over-specification of concurrency within the process.

Other control-flow aspects of a model are compromised by common modeling errors. If there are many possible paths in the process, but only one can be taken, then fulfilling dependencies is critical for the modeler. If the modeler notes that a step after a path selection depends on a product that is produced during the path selection, then all possible paths must produce that resource or the modeler has introduced the potential for a stall in the process. As possible paths become more numerous and more complicated, it becomes difficult to track what is produced and where it will be available.

Once a process model has been effectively implemented at a level of resource specification, it is possible to transition to a lower level of abstraction that will illustrate constraints on the state of objects within the process. This level of abstraction is the most detailed and also the most error prone.



```
iteration {
  action first { }
  action second { }
}
action post { }
```

```
selection {
  action choice_1 { }
  action choice_2 { }
}
```

```
branch {
  action path_1 { }
  action path_2 { }
}
```

Figure 2. Graph representations of PML control constructs.

When transitioning to a detailed specification, the modeler must keep track of dependencies between the properties of resources as well as the resources themselves. The addition of properties to the model can disrupt the dependencies that were in place at higher levels of abstraction. For example, if the requirements for a step in the model are altered to include the state of a property, but the model fails to specify that the property was introduced by an earlier step, then the dependency between the two steps is broken.

The primary objective of a tool designed to analyze a process model is to examine the model for the types of errors mentioned. In order to fulfill this objective, there are a number of requirements that a tool must meet and failing to meet these requirements is detrimental to the tool's usefulness:

- *meaningful feedback:* The tool should attempt to constructively map the errors in the model to conceptual errors in the real process.
- *analysis refinement:* The evolutionary nature of process modeling languages requires that supporting tools operate at each level of refinement in the development of the process model. If the analysis tool is reporting resource and dependency errors when the model is at a higher level of abstraction, then the analyzer has failed to meet the evolutionary requirements of the language.
- *ease of use:* If the analysis tool is cryptic, slow, or difficult to use, then it will deter users from utilizing it to aid their model development.

4.2. Tool design and implementation

Our tool is designed to translate a process model into a format that incorporates all aspects of the model and based on the structure of processes, the most intuitive representation is a graph. The procedure for mapping from a PML model to a graph is relatively simple; the nodes of a graph represent actions constructs and the edges represent the flow of control. The language constructs designed for describing control flow are interpreted and constructed into a graph in a syntax-directed, bottom-up manner as shown in Figure 2. Each action node describes the resources that are used and produced through the *provides* and *requires* properties. A tree structure is used to describe resources and expressions.

One of the objectives of an automated analysis tool the ability to check a process at many levels of abstraction. Our tool, PMLCHECK, currently provides four conceptual levels of checking: syntax checking only, resource specification, resource dependencies, and expression satisfiability.

PMLCHECK is not strictly limited to providing information at these levels of refinement and within each conceptual level there are a variety of checks that are performed and PMLCHECK can focus analysis on a particular point of interest. This flexibility was intentionally designed to reflect the evolutionary nature of process specification and the PML language while providing the modeler with control over information gathered by the tool.

We noted that inconsistencies may be introduced into a model because of a failure to specify requirements for a task. In PML, this translates to the failure to require or provide a resource in an action. These types of errors fall into four categories: those requiring and providing no resources ("*empty*"), those only requiring resources ("*black holes*"), those only providing resources ("*miracles*"), and those that provide resources other than those that they require ("*transformations*").

Each of these scenarios is an indicator that something has been left out of the process model and is a projection of problems discussed previously. Since all of these cases are local to an action, PMLCHECK simply examines each action in turn in order to find errors. However, there are legitimate cases where a new resource is created and we want to explicitly state that it is not an error. Using qualifiers provides the ability to state that a transformation should in fact occur. We provide a predefined qualifier, derived, that will suppress a warning in the case of a transformation, but this is only one of many uses for a qualifier.

In contrast, tracing dependencies through a model is much more complicated than simple specification checks. Control-flow constructs and the level of specification of a resource play an important role in determining whether or not resources are available. PMLCHECK implements two types of resource-based dependency checks: assuring resources required by an action are provided, and provided resources are required by an action.

To implement both of these checks, PMLCHECK uses standard graph propagation algorithms to propagate the availability of resources through the control-flow graph. A resource available along only one path of a **selection** construct is marked as only possibly available. A similar check is performed for resources that are produced or consumed in concurrent actions in a **branch**.

Error Type	Initial	Revised	Final
Empty	2	0	0
Miracle	2	0	0
Black hole	6	0	0
Transformation	32	1	0
Unprovided	24	7	0
Unconsumed	20	12	0

Table 1. Summary of errors reported by PMLCHECK.

Finally, to implement the checks for expression satisfiability, PMLCHECK uses logical equalities to first rewrite the expressions into a canonical form to eliminate negations and many of the relational operators, thereby reducing total number of cases that need to be examined. Since expressions are limited only to logical and relational operations on resources and literals, satisfiability is simple to implement using a straightforward exhaustive algorithm as a unification-based approach is not required. (For full details, see [21].)

5. Experimental results

NetBeans is an IDE for Java, whose requirements and release process is based on a distributed open source development model, and is therefore different than a traditional software process. In open source projects such as this, the actual coding of the system is external to the requirements and release of the product and the software development process is not concerned with how the code is written because the authors develop in a variety of environments.

The development process for NetBeans has two components: eliciting requirements and releasing the next version of the software. The first stage entails detailing what features should be included in the next version of the software and the second is based on establishing that the code is ready for release and generating a deliverable. The NetBeans development process is not self-contained because it relies on the previous revision of the process to continue. Though many software projects are terminated when the product is finalized, a release for NetBeans signifies a specific level of achievement of the software, but development continues to proceed.

Using the model from [7], analysis consisted of two levels of refinement in order to capture inconsistencies at different levels of abstraction. On first inspection of the model it is clear that the model is in a very basic state in that it includes control and resources, but no attributes or expressions. Through verification using PMLCHECK, we improved the quality and consistency of the model by removing errors without adversely affecting the underlying process.

The first application of PMLCHECK revealed a significant number of errors in the process and are summarized in Table 1. Empty actions generally indicate that resources are missing from the specification. For example, the empty action CompleteStabilization is the final action in the model, but it does not require anything and does not produce anything. However, this action is clearly included to finalize the product and make it available, but any information about what resources are required was omitted. The action WaitForVolunteer also does not contain resources, but for a different reason. This action is an artificial action created to represent what the process is doing in preparation for the next action to take place. It is not essential for the process because the next action must be ready before the process can continue, so it can be removed without adversely affecting the rest of the model.

"Black holes" pose a problem similar to empty actions. Though actions such as ReviewNetBeans and SendMessageToCommunityForFeedback were initially specified as not providing anything, they do contribute to the process. ReviewNetBeans may not provide anything new, but it does affect a property of the road-map and should reflect those changes by providing NetBeansRoadmap.Reviewed. In Figure 3, the action SendMessageToCommunityForFeedback would intuitively imply that feedback is gathered from the community and thus should provide CommunityFeedback as a resource, as Figure 4 shows.[2] These types of oversights are a misrepresentation of the process, and PMLCHECK helped locate the cause of these inconsistencies.

PMLCHECK reports that there are a significant number of transformations being performed in the process, but this report has two possibilities: the transformation is correct and the tool should not consider the created resource as an error, or the transformation is indicative of a change to a resource that was not specified as a requirement to the action. The only possible way to determine the actual meaning is to carefully inspect the process model. Action SetReleaseDate is an obvious situation where the tool is improperly reporting an inconsistency because the release date is derived from the road-map. By qualifying the created resource as (derived) ReleaseDate, PMLCHECK will understand that the resource is intended to be available at this point in the process. Action ReviseProposalBasedOnFeedback is an example of where a transformation is improper, as shown in Figure 3. This action is modifying two resources PotentialRevisionsToDevelopmentProposal and RevisedDevelopmentProposal, but these relate to a single resource: DevelopmentProposal. As shown in Figure 4, by consolidating these resources to a single resource and using attributes, we can

2 Due to space considerations, only extracts from the models are shown here. The complete models are approximately two hundred lines each and can be found in [21].

```
iteration EstablishFeatureSet {
  action CompileListOfPossibleFeaturesToInclude {
  requires { ProspectiveFeaturesGatheredFromIssuezilla &&
    ProspectiveFeaturesFromPreviousReleases }
  provides { FeatureSetForUpcomingRelease }
  }
  action CategorizeFeaturesProposedFeatureSet {
    requires { FeatureSetForUpcomingRelease }
    provides { WeightedListOfFeaturesToImplement }
  }
  action SendMessageToCommunityForFeedback {
    requires { WeightedListOfFeaturesToImplement }
    /*provides { }*/
  }
  action ReviewFeedbackFromCommunity {
    requires { FeebackMessagesOnMail }
    provides { PotentialRevisionsToDevelopmentProposal }
  }
  action ReviseProposalBasedOnFeedback {
    requires { PotentialRevisionsToDevelopmentProposal }
    provides { RevisedDevelopmentProposal }
  }
}
```

Figure 3. Extract from original NetBeans model.

reduce the total number of resources. In addition to clarifying the model, this change brings forth a more critical problem: nowhere in the specification of the process is the development proposal created. The first indication of a development proposal is in action ReviewFeedbackFromCommunity which provides PotentialRevisionsToDevelopmentProposal, but prior to this action there is no development proposal, so it is difficult to discuss potential revisions to a nonexistent proposal.

Though a report of an unprovided resource can mean a misrepresentation of process, it can also be indicative of a resource that should preexist the process. Action ReviewNetBeans requires the NetBeansRoadmap, but this is the first action in the process which means the resource cannot be specified prior to its use. PMLCHECK also reports resources that are provided by an action but are not used later in the process. One possible cause for this error is that a task later in the process has been misspecified and does not note that it requires a certain resource. For example, action ReportIssuesToIssuezilla provides IssuezillaEntry, but this resource is never used in the process. The following action looks at standing issues, but does not explicitly require this resource. PMLCHECK provides a simple mechanism for specifying the inputs and outputs of a process to suppress these types of errors.

After applying the types of changes described along with some cosmetic changes of names throughout the process, we arrive at a revised model with the number of errors shown in Table 1. Examining these remaining errors revealed that many were the result of changes made to the process including trivial errors resulting from case-sensitivity and misspellings. Applying the same techniques to our revised model resulted in a final model with no errors.

```
iteration EstablishFeatureSet {
  action CompileListOfPossibleFeaturesToInclude {
    requires { ProspectiveFeatures.Issuezilla &&
         ProspectiveFeatures.PreviousVersions }
    provides { (derived) ReleaseFeatureSet }
  }
  action CategorizeFeaturesProposedFeatureSet {
    requires { ReleaseFeatureSet }
    provides { ReleaseFeatureSet.Weighted }
  }
  action CreateDevelopmentProposal {
    requires { ReleaseFeatureSet.Weighted }
    provides { (derived) DevelopmentProposal }
  }
  action SendMessageToCommunityForFeedback {
    requires { ReleaseFeatureSet.Weighted &&
         DevelopmentProposal && CommunityMailingList }
    provides { (derived) CommunityFeedback }
  }
  action ReviewFeedbackFromCommunity {
    requires { CommunityFeedback && DevelopmentProposal }
    provides { DevelopmentProposal.PotentialRevisions }
  }
  action ReviseProposalBasedOnFeedback {
    requires { DevelopmentProposal.PotentialRevisions }
    provides { DevelopmentProposal.Revised }
  }
}
```

Figure 4. Extract from revised NetBeans model.

6. Related work

APPL/A [17] is a process enactment language designed as a superset of Ada to maximize automation. Features specific to modeling that are not implemented in Ada are constructed as extensions to the language. The modeling language JIL [20] aims to recreate many of the functionalities of languages such as APPL/A, but without the underlying programming language. JIL is designed with a combination of proactive and reactive control constructs allowing the modeler to define the control flow, or have it determined by the interpreter. While JIL is designed toward complete automation, PML supports user interaction to allow more dynamic models. Also, whereas JIL provides high-level constructs for modeling software processes, PML is not restricted to just the software process domain.

Cook and Wolf [5] discuss a method for validating software process models by comparing specifications to actual enactment histories. This technique is applicable to downstream phases of the software life-cycle, as it depends on the capture of actual enactment traces for validation. As such, it complements our technique, which is an upstream approach. Similarly, Johnson and Brockman [9] use execution histories to validate models for predicting process cycle times. The focus of their work is on estimation rather than validation, and is thus concerned with control flow rather than resource flow.

Scacchi's research uses a knowledge-based approach to analyzing process models. Starting with a set of rules that describe a process setting and models, processes are diagnosed for problems related to consistency, completeness, and traceability. Conceptually, this work is closely related to ours; many of the inconsistencies uncovered by PMLCHECK are also revealed by Scacchi and Mi's *Articulator* [16].

7. Conclusion

We have presented a philosophy of modeling based on the fundamental elements of processes with the intention of highlighting the essential components of processes in order to create informative models for analysis and enactment. We utilized this philosophy as a framework for designing a high-level language, PML, that has the expressive capability to model processes at abstract and concrete levels of specification. This language has a number of features such as qualifiers that allows flexible development and specification. However, the consequence of constructing this new language is lack of tool support and modeling for the purpose of improvement requires verification of the model.

To provide support for PML, we implemented a new method of process checking based on our research into process structure. The resulting tool, PMLCHECK, examines process models looking for common errors that result from process development and design. The flexibility of the language and the tool allow for specification and verification at many levels of abstraction. Using a general approach to process modeling and analysis allows for the concepts presented in this paper to be applied to a variety of modeling languages and analysis tools. Finally, the model of the NetBeans process that we examined and refined illustrates many benefits of tool-guided analysis. Understanding the resource flow of a process provides useful information to improve the specification of a process and to note areas of ambiguity. Examining the interaction of resources in the process can also improve the enactability of a model by ensuring that resource flow is consistent throughout.

References

[1] P. Armenise, S. Bandinelli, C. Ghezzi, and A. Morzenti. A survey and assessment of software process representation formalisms. *Int. J. Softw. Eng. Knowl. Eng.*, 3(3):401–426, Sept. 1993.

[2] D. C. Atkinson and J. Noll. Automated validation and verification of process models. In *Proc. 7th IASTED Int. Conf. on Softw. Eng. Appl.*, pages 587–592, Cambridge, MA, Nov. 2003.

[3] A. G. Cass, B. S. Lerner, E. K. McCall, L. J. Osterweil, S. M. Sutton, Jr., and A. Wise. Little-JIL/Juliette: A process definition language and interpreter. In *Proc. 22nd Int. Conf. on Softw. Eng.*, pages 754–757, Limerick, Ireland, June 2000.

[4] R. Conradi and C. Liu. Process modelling languages: One or many? In *Proc. 4th Eur. Work. on Softw. Process Tech.*, pages 98–118, Noordwijkerhout, The Netherlands, Apr. 1995.

[5] J. E. Cook and A. L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, Apr. 1999.

[6] G. Cugola and C. Ghezzi. Software processes: A retrospective and a path to the future. *Softw. Process Improv. Pract.*, 4(3):101–123, Sept. 1998.

[7] C. Jensen, W. Scacchi, M. Oza, E. Nistor, and S. Hu. A first look at the NetBeans requirements and release process. Technical report, Institute for Software Research, Feb. 2004.

[8] G. Joeris and O. Herzog. Towards flexible and high-level modeling and enacting of processes. In *Proc. 11th Int. Conf. on Adv. Inf. Syst. Eng.*, pages 88–102, Heidelberg, Germany, June 1999.

[9] E. W. Johnson and J. B. Brockman. Measurement and analysis of sequential design processes. *ACM Trans. Des. Autom. Electron. Syst.*, 3(1):1–20, Jan. 1998.

[10] G. Junkermann, B. Peuschel, W. Schäfer, and S. Wolf. Merlin: Supporting cooperation in software development through a knowledge-based environment. In *Software Process Modelling and Technology*, pages 103–129. Research Studies Press Ltd., 1994.

[11] G. E. Kaiser, S. Popovich, and I. Z. Ben-Shaul. A bi-level language for software process modeling. In *Proc. 15th Int. Conf. on Softw. Eng.*, pages 132–143, Baltimore, MD, May 1993.

[12] C. D. Klingler. A STARS case study in process definition. Technical Report F19628-88-D-0031, DARPA, 1994.

[13] C. D. Klingler, M. Neviaser, A. Marmor-Squires, C. M. Lott, and H. D. Rombach. A case study in process representation using MVP-L. In *Proc. 7th Annual Conf. on Comp. Assur.*, pages 137–146, Gaithersburg, MD, June 1992.

[14] J. Noll and W. Scacchi. Specifying process-oriented hypertext for organizational computing. *J. Netw. Comput. Appl.*, 24(1):39–61, Jan. 2001.

[15] R. F. Paige, J. S. Ostroff, and P. J. Brooke. Principles for modeling language design. *Inf. Softw. Technol.*, 42(10):665–675, July 2000.

[16] W. Scacchi and P. Mi. Process life cycle engineering: A knowlege-based approach and environment. *Int. J. Intell. Syst. Account. Financ. Manage.*, 6(2):83–107, June 1997.

[17] S. M. Sutton, Jr. *APPL/A: A Prototype Language for Software-Process Programming*. PhD thesis, University of Colorado, Department of Computer Science, Aug. 1990.

[18] S. M. Sutton, Jr., D. Heimbinger, and L. J. Osterweil. APPL/A: A language for software-process programming. *ACM Trans. Softw. Eng. Methodol.*, 4(3):221–286, July 1995.

[19] S. M. Sutton, Jr., B. S. Lerner, and L. J. Osterweil. Experience using the JIL process programming language to specify design processes. Technical Report UM-CS-1997-068, University of Massachusetts, 1997.

[20] S. M. Sutton, Jr. and L. J. Osterweil. The design of a next-generation process language. In *Proc. 6th Euro. Softw. Eng. Conf. and 5th ACM Symp. on Found. Softw. Eng.*, pages 142–158, Zurich, Switzerland, Sept. 1997.

[21] D. C. Weeks. Process modeling language design and model verification. Master's thesis, Santa Clara University, Department of Computer Engineering, June 2004.

Modeling Recruitment and Role Migration Processes in OSSD Projects

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

Abstract

Socio-technical processes have come to the forefront of recent analyses of the open source software development (OSSD) world. Though there many anecdotal accounts of these processes, such narratives lack the precision of more formal modeling techniques, which are needed if these processes are going to be systematically analyzed, simulated, or re-enacted. Interest in making these processes explicit is mounting, both from the commercial side of the industry, as well as among spectators who may become contributors to OSSD organization. Thus, the work we will discuss in this paper serves to close this gap by analyzing and modeling recruitment and role transition processes across three prominent OSSD communities whose software development processes we've previously examined: Mozilla.org, the Apache community, and NetBeans.

Keywords: Project recruitment, membership, process modeling, open source, Mozilla, Apache, NetBeans

Introduction

In recent years, organizations producing both open and closed software have sought to capitalize on the perceived benefits of open source software development methodologies. This necessitates examining the culture of prominent project communities in search of ways of motivating developers. Although the ensuing studies have provided much insight into OSSD culture, missing from this picture was the process context that produced the successes being observed. Ye and Kishida (2003) and Crowston and Howison (2005) observe that community members gravitate towards central roles over time represented with "onion" diagrams such as in figure 1. These depictions indicate a similar number of layers in organizational hierarchies across communities, but do not suggest how one might transition between layers and what roles are available at each layer. Much like their development processes, OSSD communities typically provide little insight into role migration processes. What guidance is provided is often directed at recruitment- initial steps to get people in the door. Guidance for attaining more central roles is often characterized as being meritocratic, depending on the governance structure of the community. Nevertheless, these development roles and how developers move between them seems to lie outside of the traditional view of software engineering, where developers seem to be limited to roles like requirements analyst, software designer, programmer, or code tester, and where there is little/no movement between roles (except perhaps in small projects).

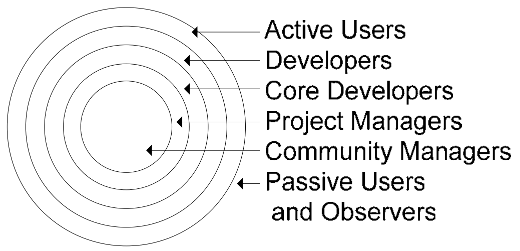


Figure 1. An "onion" diagram representation of an open source community organizational hierarchy

Christie and Staley (2000) argue that social and organizational processes, such as those associated with moving between different developer roles in a project, are important in determining the outcome of software development processes. In previous studies, we have examined software development processes within and across OSSD communities (Jensen and Scacchi, 2005, Scacchi 2002, 2004, 2005). Here, we take a look at two related socio-technical processes used in OSSD as a way of merging the social/cultural and technical/developmental OSSD activities. Specifically, we'll focus on the recruitment and migration of developers from end-users or infrequent contributors towards roles more central to the community, like core developer, within projects such as the Mozilla, Apache community, and NetBeans projects. Such processes characterize both the hierarchy of roles that OSS developers play (cf. Gacek and Arief 2004), as well as how developers move through or become upwardly mobile within an OSSD project (Sim and Holt 1998). While anecdotal evidence of these processes exists, the lack of precision in their description serves as a barrier to community entry, continuous improvement, and process adoption by other organizations. The goal of our work here thus serves to provide process transparency through explicit modeling of such processes in ways that may enable increased community participation, more widespread process adoption, and process improvement.

In the remaining sections, we outline details about recruitment and role migration as membership processes as found while examining each of these three OSSD project communities. At the ProSim'05 Workshop we will present a variety of semi-structured and formal models that enable more rigorous analysis and simulated re-enactment using tools and techniques we have previously developed and employed (cf. Noll and Scacchi 2001, Jensen and Scacchi 2005)

Membership Processes in Mozilla.org

Developer recruitment in Mozilla was difficult at the start. The opening of the Netscape browser source code offered developers a unique opportunity to peek under the hood of the once most dominant Web browser in use. Nevertheless, the large scale of the application (millions of lines of source code) and the complex/convoluted architecture scared developers away. These factors, combined with the lack of a working release and the lack of support from Netscape led one project manager to quit early on (Mockus, *et. al*, 2002). However, with the eventual release of a working product, the Mozilla project garnered users who would become developers to further the cause.

The Mozilla Web site lists several ways for potential developers and non-technical people to get involved with the community (Getting Involved with Mozilla.org, 2005). The focus on quality assurance and documentation reflects a community focus on maturing, synchronizing, and stabilizing updates to the source code base. Technical membership roles and responsibilities currently listed include bug reporting, screening, confirming, and fixing, writing documentation, and contacting sites that do not display properly under Mozilla. Compared to more central roles, these activities do not require deep knowledge of the Mozilla source code or system architecture, and serve to allow would-be contributors to get involved and participate in the overall software development process.

When bugs are submitted to the Bugzilla, they are initially assigned to a default developer for correction. It is not uncommon for community developers and would-be developers to become frustrated with an outstanding issue within the bug repository and submit a patch, themselves.

The next task is to recruit others to accept the patch and incorporate it into the source tree. Recruitment of patch review is best achieved through emailing reviewers working on the module for which the patch was committed or reaching out to the community via the Mozilla IRC chat. By repeatedly demonstrating competency and dedication writing useful code within a section of the source, would-be developers gain a reputation among those with commit access to the current source code build tree. Eventually, these committers recommend that the developer be granted access by the project drivers. In rare cases, such a developer may even be offered ownership of a particular module if s/he is the primary developer of that module and it has not been blocked for inclusion into the trunk of the source tree[1].

Once a project contributor is approved as a source code contributor, there are several roles available to community members. Most of these are positions requiring greater seniority or record of demonstrated accomplishments within the community. As module developers and owners establish themselves as prominent community members, other opportunities may open up. In meritocratic fashion (cf. Fielding 1999), developers may transition from being a QA module contact to a QA owner. Similar occasions exist on the project level for becoming a module source reviewer.

Super-reviewers attain rank by demonstrating superior faculty for discerning quality and effect of a given section of source on the remainder of the source tree. If a reviewer believes that s/he has done this appropriately, s/he must convince an existing super-reviewer of such an accomplishment. This super-reviewer will propose the candidate to the remainder of the super-reviewers. Upon group consensus, the higher rank is bestowed on the reviewer (Mozilla Code Review FAQ, 2005). The same follows for Mozilla drivers, who determine the technical direction of the project per release.

Community level roles include smoke-test coordinator, code sheriff, and build engineer, although no process is prescribed for such transitions. As individual roles, they are held until vacated, at which time, the position is filled by appointment from the senior community members and Mozilla Foundation staff. Role hierarchy and a flow graph of the migration process for transitioning from reviewer to super-reviewer are provided in figure 2 as an example of those we have modeled for this community. In the flow graph, rectangles refer to actions, whereas ovals

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=18574

refer to resources created or consumed by the associated action, as determined by the direction of the arrow linking the two. Transitions from one role to another are depicted with a dashed arrow from an action performed by one role to the title of another. We have also used dashed lines to differentiate social or role transitioning activities and resources from strictly technical, developmental resources.

Membership Processes in the Apache Community

Role migration in the Apache community is linear. The Apache Software Foundation (ASF) has laid out a clear path for involvement in their meritocracy. Individuals start out as end-users (e.g., Web site administrators), proceed to developer status, then committer status, project management committee (PMC) status, ASF membership, and lastly, ASF board of directors membership (How the ASF Works, 2005). Much as in advancement in the Mozilla community, Apache membership is by invitation only. As the name suggests, the Apache server is comprised of patches submitted by developers. These patches are reviewed by committers and either accepted or rejected into the source tree.

In addition to feature patches, developers are also encouraged to submit defect reports, project documentation, and participate on the developer mailing lists. When the PMC committee is satisfied with the developer's contributions, they may elect to extend an offer of "committership" to the developer, granting him/her write access to the source tree. To accept committership, the developer must submit a contributor license agreement, granting the ASF license to the intellectual property conveyed in the committed software artifacts.

PMC membership is granted by the ASF. To become a PMC member, the developer/committer must be nominated by an existing ASF member and accepted by a majority vote of the ASF membership participating in the election (Fielding, et. al, 2002). Developers and committers nominated to become PMC members have demonstrated commitment to the project, good judgment in their contributions to the source tree, and capability in collaborating with other developers on the project. The PMC is responsible for the management of each project within the Apache community. The chair of the PMC is an ASF member elected by his/her fellow ASF members who initially organizes the day-to-day management infrastructure for each project, and is ultimately responsible for the project thereafter. ASF membership follows the same process as PMC membership- nomination and election by a majority vote of existing ASF members.

ASF members may run for office on the ASF board of directors, as outlined by the ASF bylaws (Bylaws of the Apache Software Foundation, 2005). Accordingly, the offices of chairman, vice chairman, president, vice president, treasurer (and assistant), and secretary (and assistant) are elected annually. A flow graph of the role migration process appears in figure 3.

Although, there is one path of advancement in the Apache community, there are several less formal committees that exist on a community (as opposed to project) scale. These include the conference organizing committee, the security committee, the public relations committee, the Java Community Process (JCP) committee, and the licensing committee. Participation in these committees is open to all committers (and higher ranked members) and roles are formalized on an as-needed basis (e.g. conference organization). Non-committers may apply for inclusion in specific discussion lists by sending an email to the board mailing alias explaining why access should be granted. Thus, processes associated with these committees are ad hoc and consist of one step.

Membership Processes in the NetBeans.org Community

Roles in the NetBeans.org community for developing the Java-based NetBeans interactive development environment are observable on five levels of project management (Oza, et. al 2002) just as in Apache. These range from users to source contributors, module-level managers, project-level managers, and community-level managers. The NetBeans community's core members are mostly Sun Microsystems employees, the community's primary sponsor, and are subject to the responsibilities set on them by their internal organizational hierarchy. As such, (and unlike the cases of Apache and Mozilla), not all roles are open to volunteer and third-party contributors. Non-Sun employed community members wanting to participate beyond end-usage are advised to start out with activities such as quality assurance (QA), internationalization, submitting patches, and documentation (Contributing to the NetBeans Project, 2005). As in the case with Mozilla, until they have proven themselves as responsible, useful, and dedicated contributors, developers must submit their contributions to developer mailing lists and the issue repository, relying on others with access to commit the source. However, unlike Mozilla, developers are also encouraged to start new modules.

While the community was more liberal with module creation early in the project's history, as the community has matured, additions to the module catalogue have become more managed to eliminate an abundance of abandoned modules. Also as in Mozilla, developers are subjected to the proving themselves before being granted committer status on a portion of the source tree. Additionally, they may gain module owner status be creating a module or taking over ownership of an abandoned module that they have been the primary committer for. With module ownership comes the responsibility to petition the CVS manager to grant commit access to the source tree to developers, thereby raising their role status to "committer."

Rising up to the project-level roles, the Sun-appointed CVS source code repository manager is responsible for maintaining the integrity of the source tree, as well as granting and removing developer access permissions. In contrast, the release manger's role is to coordinate efforts of module owners to plan and achieve timely release of the software system. Theoretically, any community member may step in at any time and attempt to organize a release. In practice, this rarely occurs. Instead, most community members passively accept the roadmap devised by Sun's NetBeans team. In the latter case, the previous release manager puts out a call to the community to solicit volunteers for the position for the upcoming cycle. Assuming there are no objections, the (usually veteran) community member's candidacy is accepted and the CVS manager prepares the source tree and provides the new release manager permissions accordingly. Alternatively, a member of Sun may appoint a member of their development team to head up the release of their next development milestone.

At the community-management level, the community managers coordinate efforts between developers and ensures that issues brought up on mailing lists are addressed fairly. At the inception of the NetBeans project, an employee of CollabNet (the company hosting the NetBeans Web portal) originally acted as community manager and liaison between CollabNet and NetBeans. However, it was soon transferred to a carefully selected Sun employee (by Sun) who has held it since. As community members have risen to more central positions in the NetBeans community, they tend to act similarly, facilitating and mediating mailing list discussions of a technical nature, as well as initiating and participating in discussions of project and community direction.

Lastly, a committee of three community members, whose largely untested responsibility is to ensure fairness within the community, governs the NetBeans project.. One of the three is appointed by Sun. The community at large elects the other two members of the governance board. These elections are held every six months, beginning with a call for nominations by the community management. Those nominees that accept their nomination are compiled into a final list of candidates to be voted on by the community. A model of the product development track role migration process is shown in figure 4.

Discussion

In both NetBeans and Mozilla, recruitment consists of listing ways for users and observers to get involved. Such activities include submitting defect reports, test cases, source code and so forth. These activities require a low degree of interaction with other community members, most notably decision makers at the top of the organizational hierarchy. Our observation has been that the impact of contributions trickles up the organizational hierarchy whereas socio-technical direction decisions are passed down. As such, activities that demonstrate capability in a current role, while also coordinating information between upstream and downstream (with respect to the organizational hierarchy) from a given developer are likely to demonstrate community member capability at his/her current role, and therefore good candidates for additional responsibilities.

Recruitment and role migration processes aren't something new; since they describe the actions and transition passages involved in moving along career paths. Like career paths described in management literature (e.g., Lash and Sein 1995), movement in the organizational structure may be horizontal or vertical. Most large OSSD project communities are hierarchical, even if here are few layers to the hierarchy and many members exist at each layer.

In the communities we have examined, we found different paths (or tracks) towards the center of the developer role hierarchy as per the focus of each path. Paths we've identified include project management (authority over technical issues) and organizational management (authority over social/infrastructural issues). Within these paths, we see tracks that reflect the different foci in their software processes. These include quality assurance roles, source code creation roles, and source code versioning roles (e.g. cvs manager, cvs committer, etc), as well as role paths for usability, marketing, and licensing. There are roles for upstream development activities (project planning--these are generally taken up by more senior members of the community. This is due in part that developers working in these roles can have an impact on the system development commensurate with the consequences/costs of failure, and require demonstrated skills to ensure the agents responsible won't put the software source code into a state of disarray).

In comparison to traditional software development organizations, tracks of advancement in open source communities are much more fluid. A developer contributing primarily to source code generation may easily contribute usability or quality assurance test cases and results to their respective community teams. This is not to suggest that a module manager of a branch of source code will automatically and immediately gain core developer privileges, responsibilities, and respect from those teams. However, industrial environments tend towards rigid and static organizational hierarchies with highly controlled growth at each layer.

The depiction of role hierarchies in open source communities as concentric, onion-like circles speaks to the fact that those in the outer periphery have less direct control or knowledge of the community's current state and its social and technical direction compared to those in the inner core circle. Unlike their industrial counterparts, open source community hierarchies are dynamic. Although changes in the number of layers stabilizes early in the community formation, the size of each layer (especially the outer layers) is highly variable. Evolution of the organizational structure may cause or be caused by changes in leadership, control, conflict negotiation, and collaboration in the community, such as those examined elsewhere (Jensen and Scacchi 2005b). If too pronounced, these changes can lead to breakdowns of the technical processes.

As a general principle, meritocratic role migration processes such as those we have observed consist of a sequence of establishing a record of contribution in technical processes in collaboration with other community members, followed by certain "rights of passage" specific to each community. For Apache, there is a formal voting process that precedes advancement. However, in the Mozilla and NetBeans communities, these are less formal. The candidate petitions the appropriate authorities for advancement or otherwise volunteers to accept responsibility for an activity. These authorities will either accept or deny the inquiry.

Conclusion

Social or organizational processes that affect or constrain the performance of software development processes have had comparatively little investigation. This is partially because some of these processes may be well understood (e.g., project management processes like scheduling or staffing), while others are often treated as "one-off" or *ad hoc* in nature, executing in a variety of ways in each instantiation. The purpose of our examination and modeling study of recruitment and role migration processes is to help reveal how these socio-technical processes are intertwined with conventional software development processes, and thus constrain or enable how software processes are performed in practice. In particular, we have examined and modeled these processes within a sample of three OSSD projects that embed the Web information infrastructure. Lastly, we have shown where and how they interact with existing software development processes found in our project sample.

References

Bylaws of the Apache Software Foundation, available online at http://www.apache.org/foundation/bylaws.html accessed 7 February 2005

Christie, A. and Staley, M. "Organizational and Social Simulation of a Software Requirements Development Process" *Software Process Improvement and Practice* 2000; 5: 103–110 (2000)

Contributing to the NetBeans Project, available online at http://www.netbeans.org/community/contribute/ accessed 7 February 2005

Coward, Anonymous. "About Firefox and Mozilla" Comment on Slashdot.org forum "Firefox Developer on Recruitment Policy," available online at http://developers.slashdot.org/comments.pl?sid=137815&threshold=1&commentsort=0&tid=154&tid=8&mode=thread&cid=11527647, 31 January, 2005.

Crowston, K. and Howison, J. 2005. The Social Structure of Free and Open Source Software Development, First Monday, 10(2). February. Online at http://firstmonday.org/i8ssues/issue10_2/crowston/index.html

Elliott, M., The Virtual Organizational Culture of a Free Software Development Community, *Proceedings of the Third Workshop on Open Source Software*, Portland, Oregon, May 2003.

Fielding, R., Shared Leadership in the Apache Project. *Communications ACM*, 42(4), 42-43, 1999.

Fielding, R., Hann, I-H., Roberts, J and Sandra Slaughter, S. “Delayed Returns to Open Source Participation: An Empirical Analysis of the Apache HTTP Server Project,” Presented at the Conference on Open Source: Economics, Law, and Policy, Toulouse, France June 2002.

Gacek, C. and Arief, B., The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40, January/February 2004.

Getting Involved with Mozilla.org, Web page available online at http://www.mozilla.org/contribute/ 3 November 2004

How the ASF works, available online at http://www.apache.org/foundation/how-it-works.html, accessed 7 February 2005

Jensen, C. and Scacchi, W., Process Modeling Across the Web Information Infrastructure, *Software Process Improvement and Practice*, to appear, 2005.

Jensen, C. and Scacchi, W. Collaboration, Leadership, Control, and Conflict Negotiation Processes in the NetBeans.org Open Source Software Development Community. working paper, Institute for Software Research, March 2005

Lash, P.B. and Sein, M.K. Career Paths in a Changing IS Environment: A Theoretical Perspective, Proc. SIGCPR 1995, 117-130. Nashville, TN

Mockus, A., Fielding, R., and Herbsleb, J. “Two Case Studies of Open Source Software Development: Apache and Mozilla,” ACM Transactions on Software Engineering and Methodology, 11(3):309-346, 2002

Mozilla Code Review FAQ, available online at http://www.mozilla.org/hacking/code-review-faq.html, accessed 7 February 2005

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

Oza, M. Nistor, E. Hu, X., Jensen, C., Scacchi, W. “A First Look at the NetBeans Requirements and Release Process.” June 2002, updated February 2004 available online at http://www.isr.uci.edu/~cjensen/papers/FirstLookNetBeans/.

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

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

Scacchi, W., Socio-Technical Interaction Networks in Free/Open Source Software Development Processes, in S.T. Acuña and N. Juristo (eds.), *Software Process Modeling*, 1-27, Springer Science+Business Media Inc., New York, 2005.

Sim, S.E. and Holt, R.C., The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize, *Proc. 20th Intern. Conf. Software Engineering*, Kyoto, Japan, 361-370, 1998.

Ye, Y. and Kishida, K. Towards an Understanding of the Motivation of Open Source Software Developers, *Proc. 25th Intern. Conf. Software Engineering*, Portland, OR, 419-429, 2003.

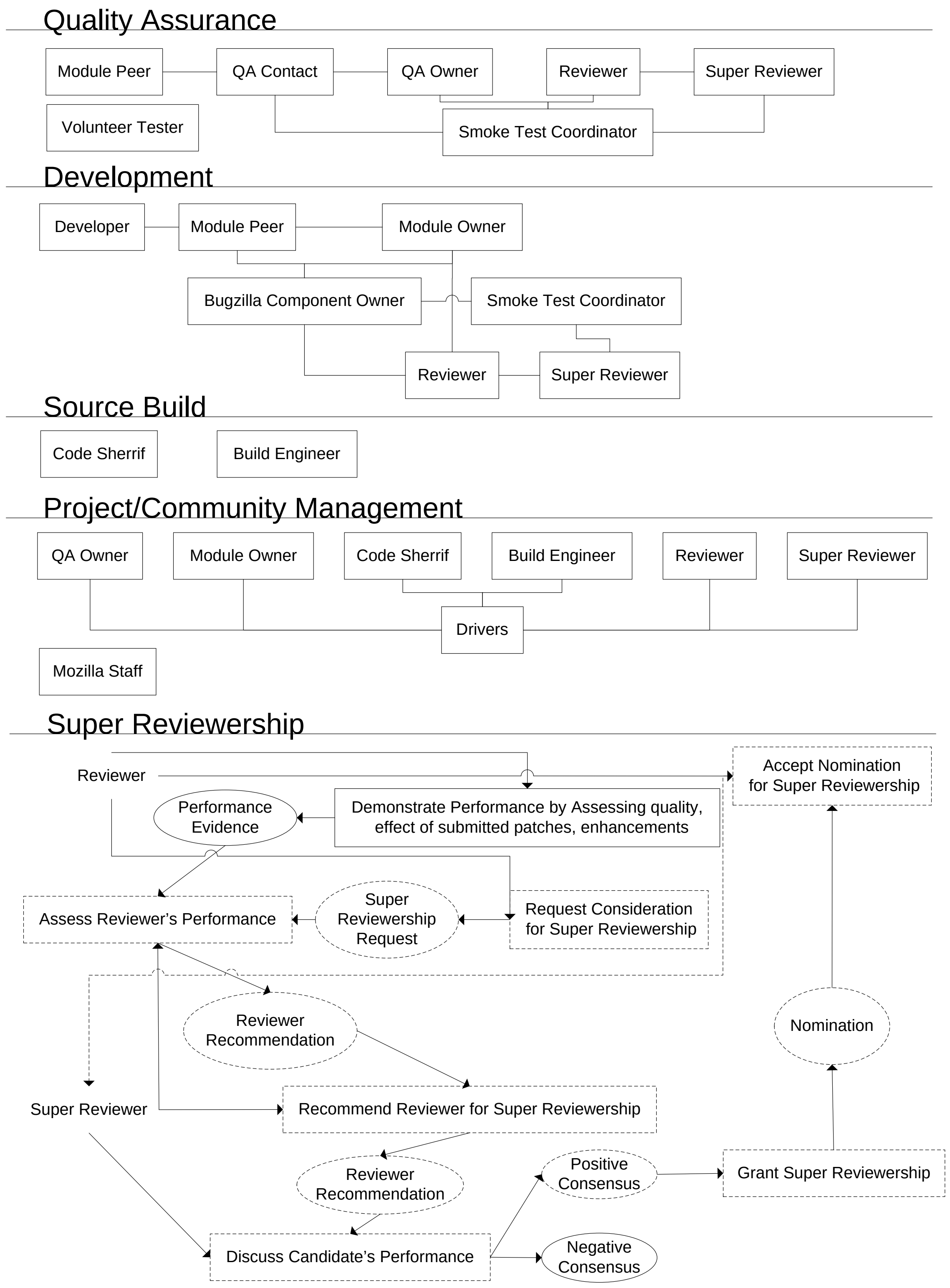


Figure 2. Role hierarchy and super reviewership migration in the Mozilla community

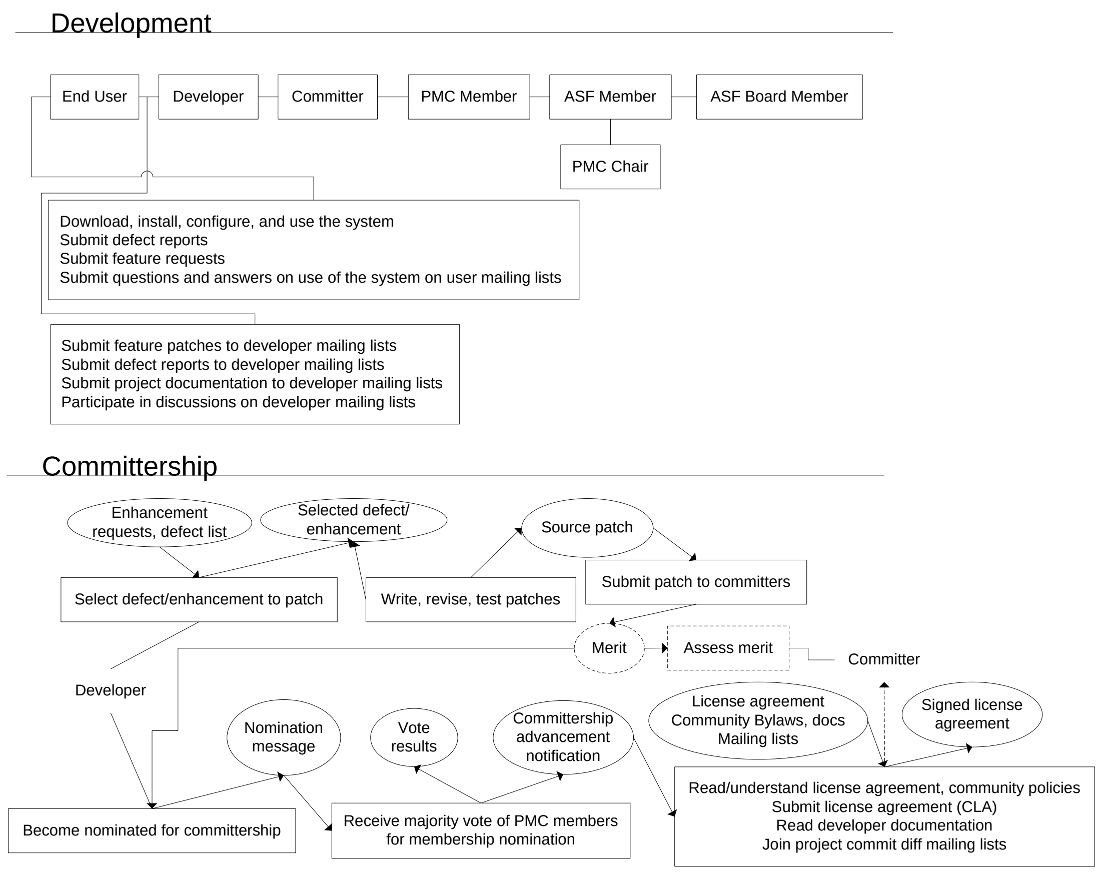
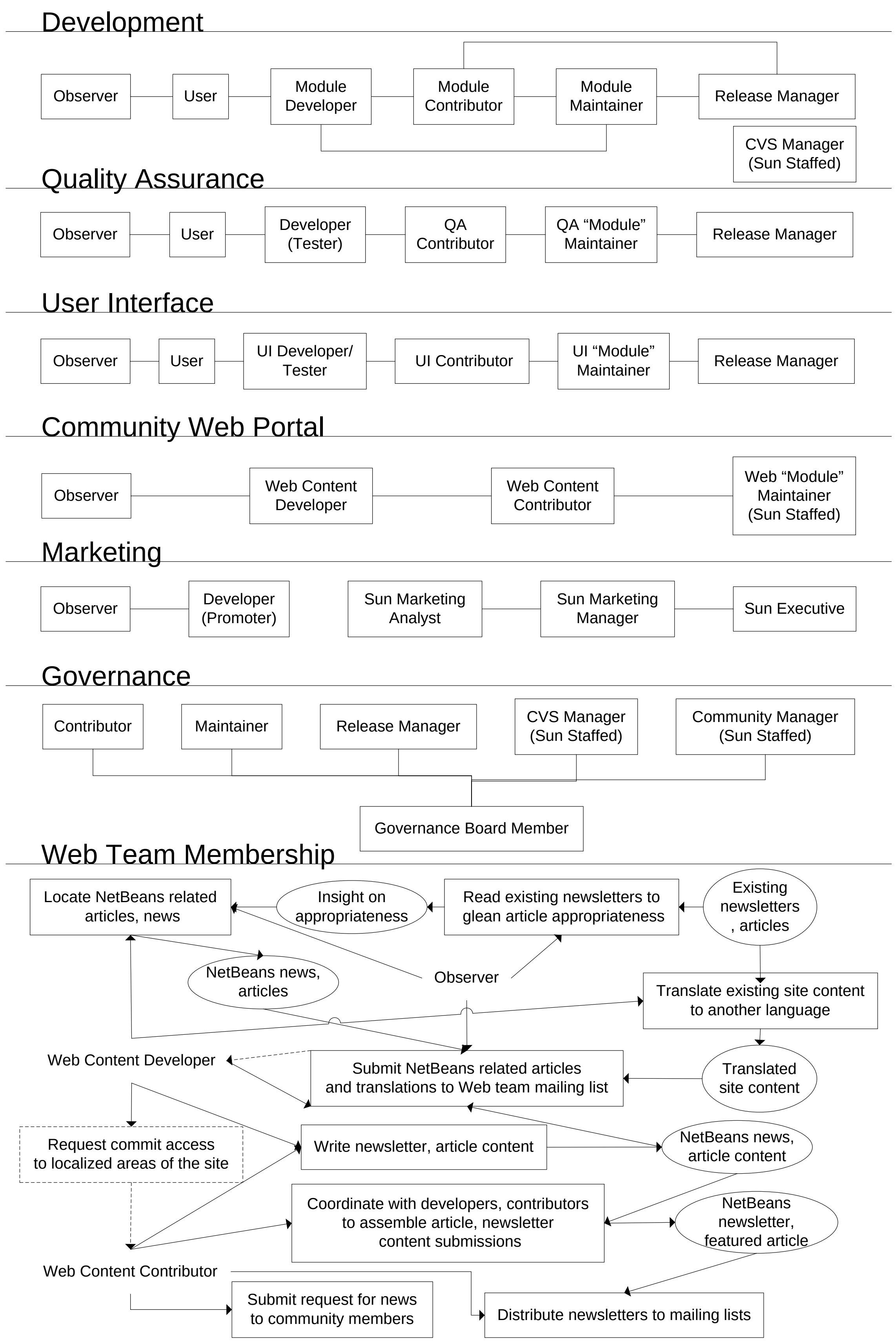

Development
End User
Developer
Committer
PMC Member
ASF Member
ASF Board Member
PMC Chair
Download, install, configure, and use the system
Submit defect reports
Submit feature requests
Submit questions and answers on use of the system on user mailing lists
Submit feature patches to developer mailing lists
Submit defect reports to developer mailing lists
Submit project documentation to developer mailing lists
Participate in discussions on developer mailing lists
Committership
Enhancement requests, defect list
Selected defect/ enhancement
Source patch
Select defect/enhancement to patch
Write, revise, test patches
Submit patch to committers
Merit
Assess merit
Committer
Developer
Nomination message
Vote results
Committership advancement notification
License agreement Community Bylaws, docs Mailing lists
Signed license agreement
Become nominated for committership
Receive majority vote of PMC members for membership nomination
Read/understand license agreement, community policies
Submit license agreement (CLA)
Read developer documentation
Join project commit diff mailing lists

Figure 3. Role hierarchy and committership migration in the Apache community, highlighting the sequence of a developer becoming a committer



Development
Observer
User
Module Developer
Module Contributor
Module Maintainer
Release Manager
CVS Manager (Sun Staffed)
Quality Assurance
Observer
User
Developer (Tester)
QA Contributor
QA “Module” Maintainer
Release Manager
User Interface
Observer
User
UI Developer/ Tester
UI Contributor
UI “Module” Maintainer
Release Manager
Community Web Portal
Observer
Web Content Developer
Web Content Contributor
Web “Module” Maintainer (Sun Staffed)
Marketing
Observer
Developer (Promoter)
Sun Marketing Analyst
Sun Marketing Manager
Sun Executive
Governance
Contributor
Maintainer
Release Manager
CVS Manager (Sun Staffed)
Community Manager (Sun Staffed)
Governance Board Member
Web Team Membership
Locate NetBeans related articles, news
Insight on appropriateness
Read existing newsletters to glean article appropriateness
Existing newsletters , articles
NetBeans news, articles
Observer
Translate existing site content to another language
Web Content Developer
Submit NetBeans related articles and translations to Web team mailing list
Translated site content
Request commit access to localized areas of the site
Write newsletter, article content
NetBeans news, article content
Coordinate with developers, contributors to assemble article, newsletter content submissions
NetBeans newsletter, featured article
Web Content Contributor
Submit request for news to community members
Distribute newsletters to mailing lists

Figure 4. Role hierarchy and Web Team membership migration in the NetBeans open source community

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

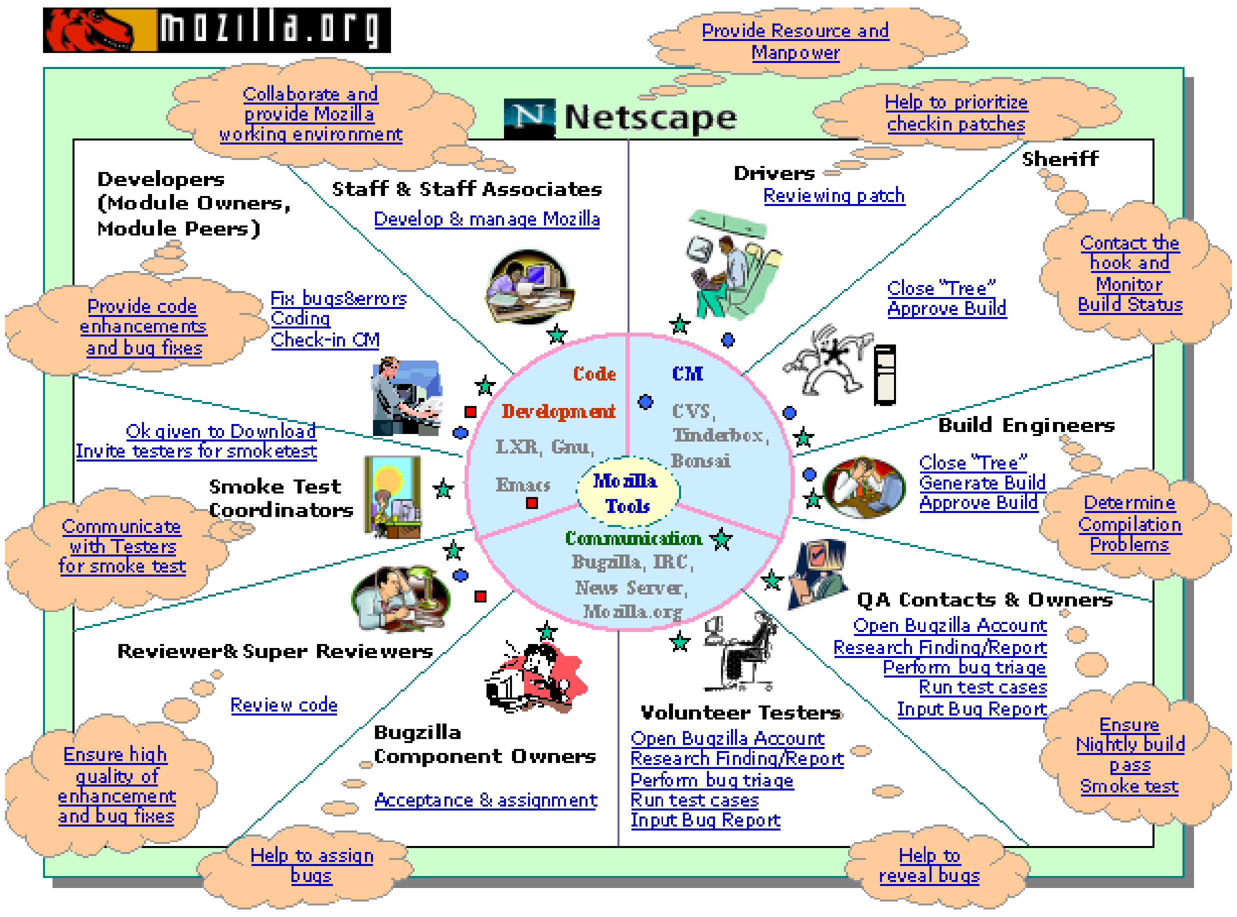


Figure 1: Mozilla quality assurance process rich picture [cf. Carder, et al 2002]

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.

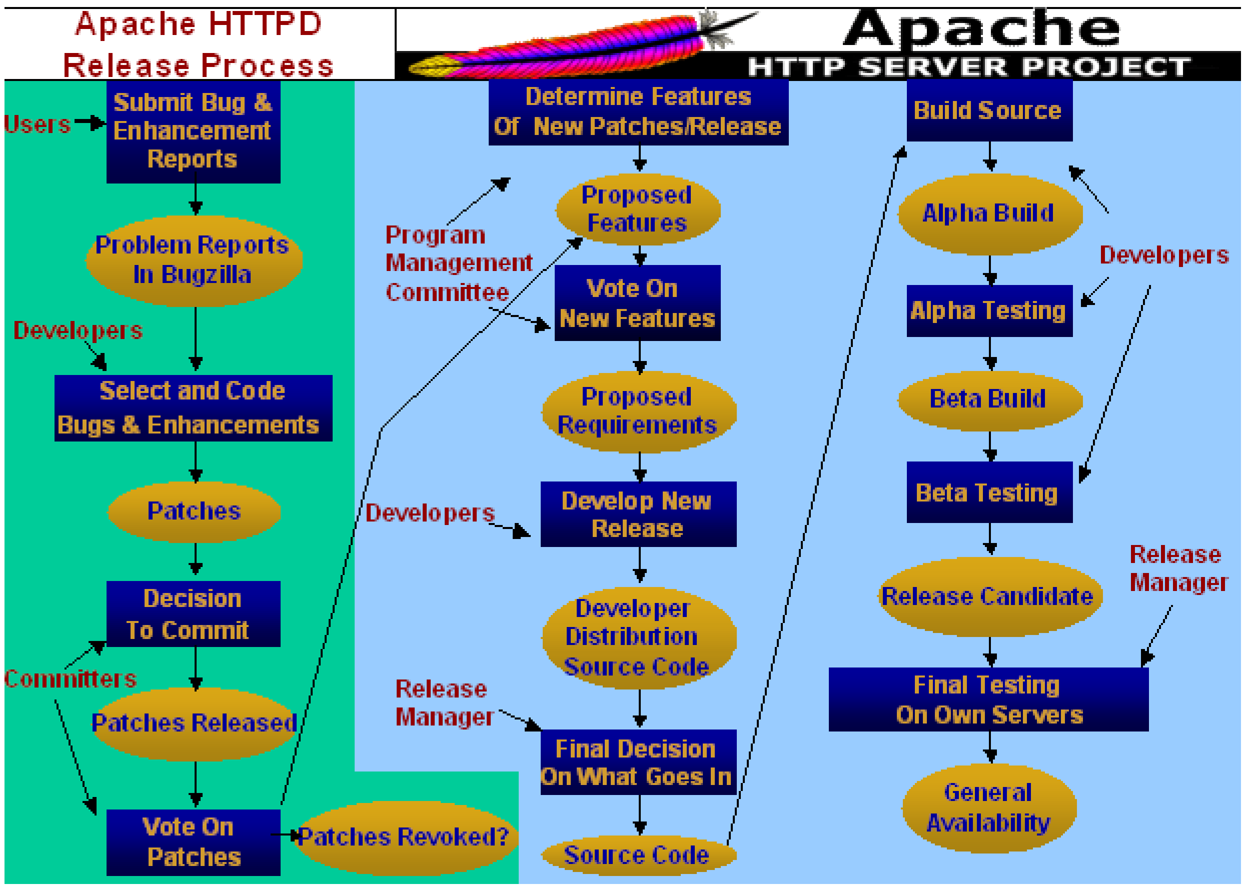


Figure 2: Apache HTTP server release process flow graph [cf. Ata, et al 2002]

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

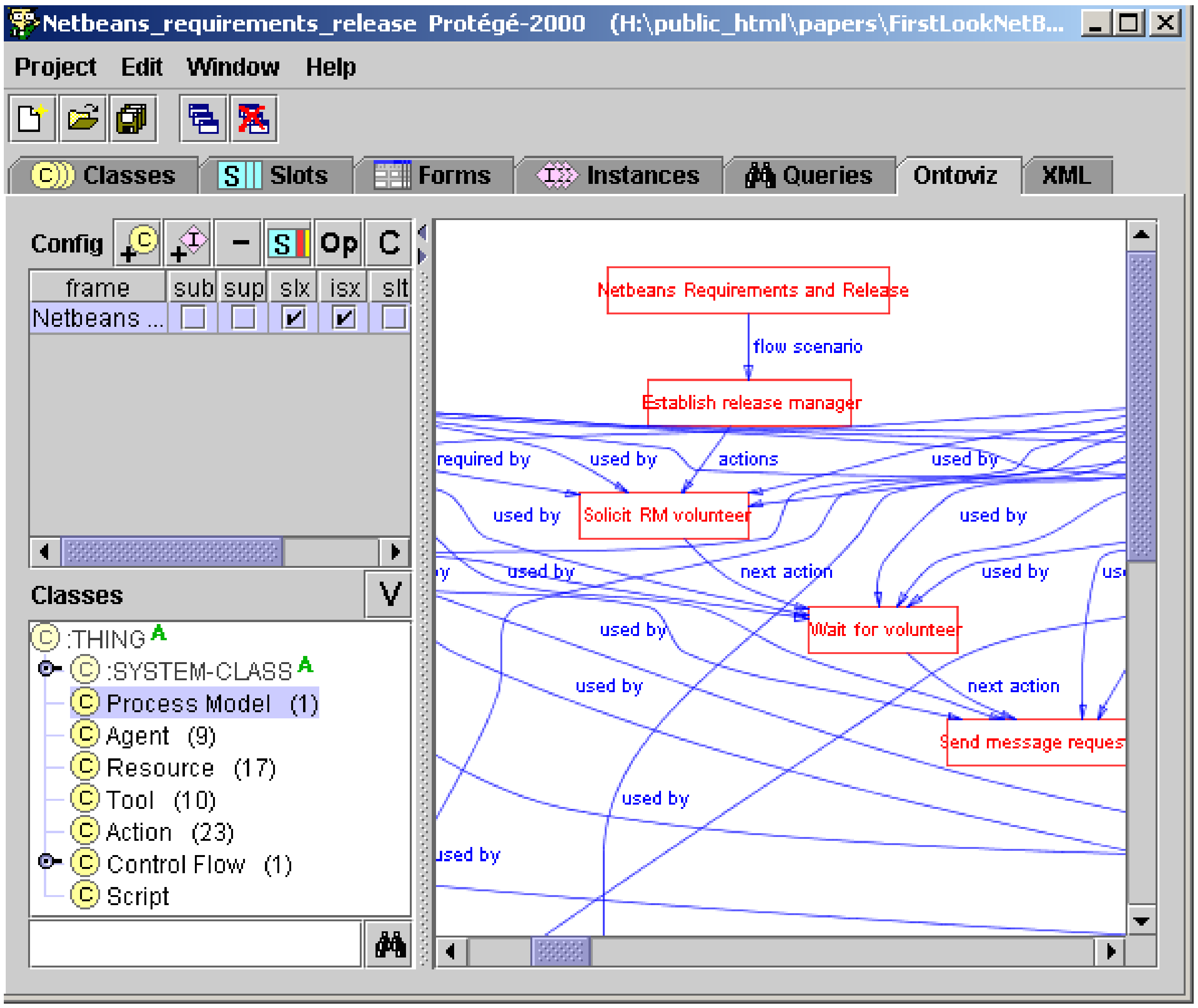
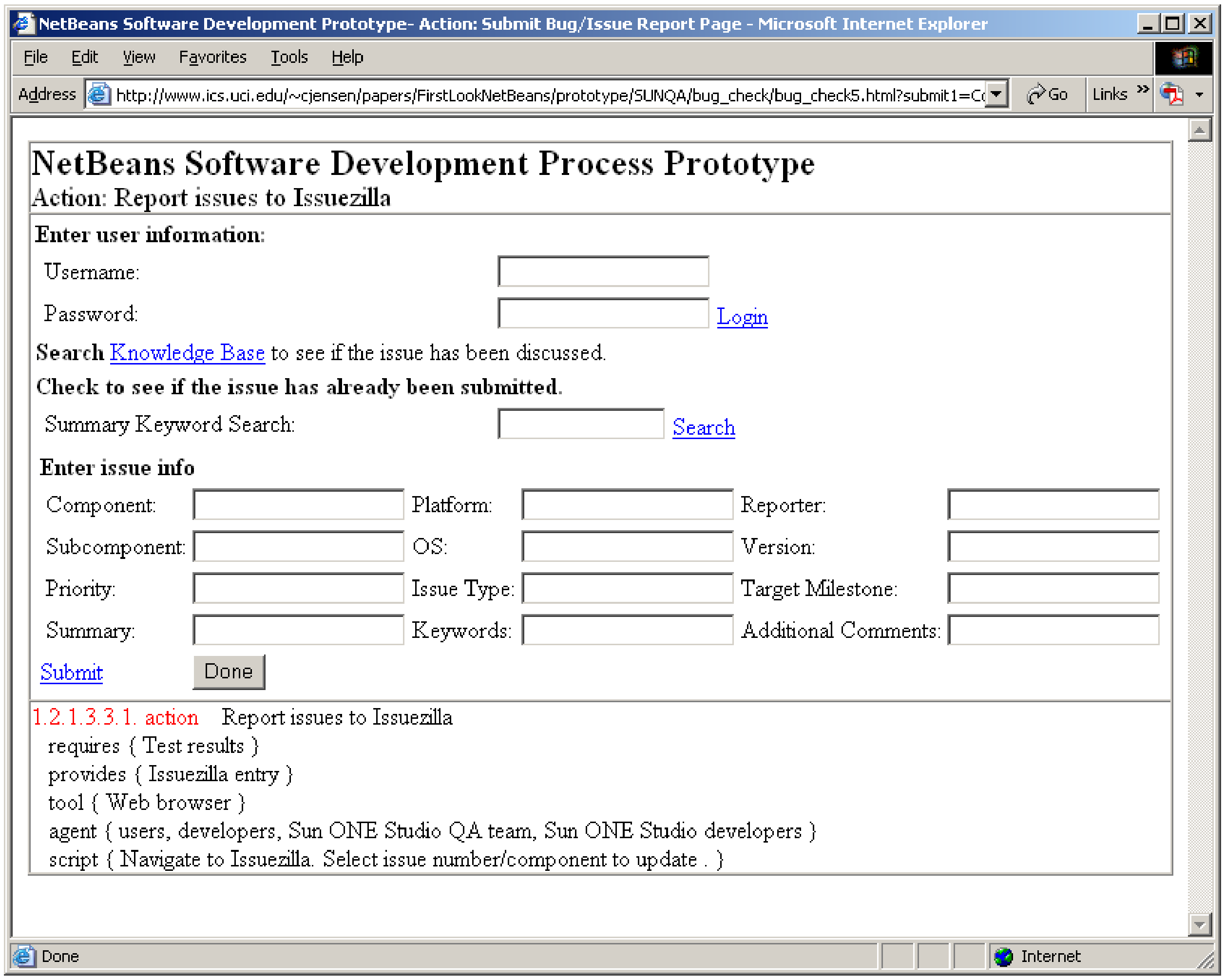


Figure 3: NetBeans requirements and release process formal model rendering with Protégé-2000

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

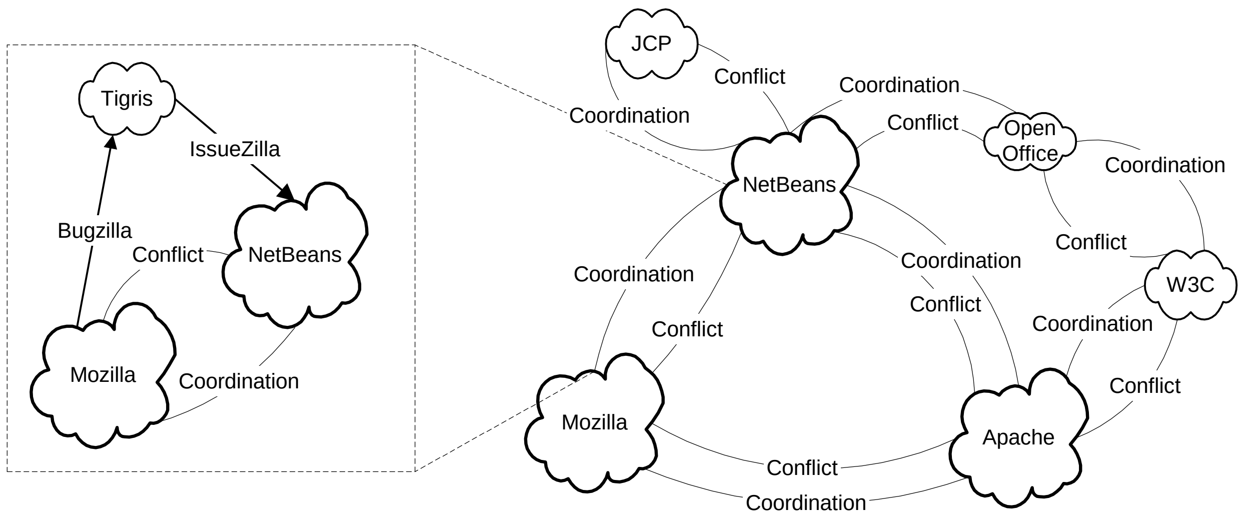


Figure 5: Intercommunity interprocesses communication in the Web information infrastructure

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.

Process Analysis and Enactment

This section contains the following two chapters that examine new methods and techniques for automatically analyzing process models, and for inferring the state of a process during its enactment.

Darren Atkinson and John Noll, Automated Validation and Verification of Software Process Models. *Proceedings SEA '03,* Marina Del Rey, CA, USA, November, 2003.

John Noll and Jigar Shah, Process State Inference for Support of Knowledge Intensive Work. *Proceedings SEA '04,* Cambridge, MA, USA, November, 2004.

Automated Validation and Verification of Process Models

Darren C. Atkinson
Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566
atkinson@engr.scu.edu

John Noll
Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566
jnoll@engr.scu.edu

ABSTRACT
In process programming, processes are modeled as pieces of software, and a process programming language is used to specify the process. Such a language resembles a conventional programming language, providing constructs such as iteration and selection. This approach allows models to be simulated and enacted easily. However, it also suffers from the same problems that plague traditional programming, such as the question of whether the program itself is semantically correct or contains errors. We present an automated approach for detecting errors in such process models. Our approach is based on static code analysis techniques. We have developed a tool to analyze processes modeled using PML and have subsequently successfully redesigned models using our tool.

KEY WORDS
Process Programming, Modelling Languages, Modelling and Simulation, Static Analysis

1 Introduction

1.1 Motivation

In 1987, Osterweil asserted that "software processes are software too" [1], and thus could (and should) be developed, analyzed, and managed using the same software engineering methods and techniques that are applied to software. This idea implies there is a software process life-cycle that resembles the software life-cycle, involving analysis, design, implementation, and maintenance of software processes [2]. One of the outgrowths of this line of research is the notion of *process programming*: the specification of process models using process programming languages that resemble, and in some cases are derived from, conventional programming languages [3].

One advantage of process programming is that a process model can be coded and simulated or enacted easily. An enactment engine can, for example, automatically notify actors when they should begin execution of a particular task. However, process programming is also subject to all of the pitfalls of traditional programming and software engineering. In particular, there is the possibility of errors in the program and, more importantly, errors in the design and in the capturing of the requirements.

There is a large body of knowledge comprising techniques for analyzing programs written in conventional programming languages. These techniques enable programmers to assess the correctness of their programs, identify potential faults, and, as in the case of optimizing compilers, automatically redesign the implementation of a program to improve execution performance. We would therefore like to apply these techniques to the analysis of process programs in order to help the process engineer find errors before simulation or enactment of a model. Specifically, we would like to use these techniques to validate the correctness of process programs as models of real-world processes and aid in process redesign.

1.2 Approach

In this paper we present a technique for analyzing the flow of *resources* through a process, as specified by a process program. This technique, derived from research into data-flow analysis of conventional programming languages, enables a process designer to answer important questions about a process model, including:

- Does a process actually produce the product that it is supposed to produce?
- Are intermediate products consumed by later steps in a process actually produced by earlier steps?
- Does the flow of resources through a process match the flow of control?

The answers to these questions can result from errors in the specification, indicating a need for further capture and modeling activities; or, they may highlight flaws in the underlying process, indicating a potential for process improvement. To validate our hypotheses, we have developed a tool to analyze specifications written in the PML process programming language [4].

We have used our tool to analyze software process models and present an in-depth analysis of the redesign of one model, used by students for their senior projects. Our tool detected 63 errors in the model, which consisted of only 204 lines of PML code. Through iterative use of the tool, we were able to successfully redesign the model.

We begin with a brief overview of PML, to provide a context for discussing our technique. Then, we present our analysis technique and discuss the implementation of our tool. We discuss our results of applying the tool to actual PML specifications. We conclude with our assessment of the technique and potential directions for future work.

2 The PML Language

PML is a simple process programming language that is intended to model organizational processes at varying levels of detail [4]. PML was designed specifically for rapid, incremental process capture, to support both process modeling and analysis, and process enactment [5]. Using PML, a process can be specified initially at a very high level that contains only major process steps and control flow.

PML reflects the conceptual model of process enactment developed by Mi and Scacchi [6]. This model views a process as a situation in which agents use tools to perform tasks that require and produce *resources*. PML models processes as collections of actions that represent atomic process tasks. PML specifies the order in which actions should be performed using conventional programming language control flow constructs such as sequencing, iteration, and selection, as well as concurrent branching of process flows:

- Sequence—A series of tasks to be performed in order:

```
sequence {
    action first {}
    action second {}
}
```

- Iteration—A series of tasks to be performed repeatedly:

```
iteration {
    action first {}
    action second {}
}
action go_on {}
```

- Selection—A set of tasks from which the actor should choose *one* to perform:

```
selection {
    action choice_1 {}
    action choice_2 {}
}
```

- Branch—A set of tasks that can be performed concurrently (all tasks in a branch must be performed before the process can continue):

```
branch {
    action path_1 {}
    action path_2 {}
}
```

The *provides* and *requires* fields of an action specify how resources are transformed as they flow through a process. As such, they capture several important facts about a process, namely what conditions must exist before an action can begin, and what conditions will exist after an action is completed. As a result, the *requires* and *provides* predicates specify the purpose of an action, in terms of how the action affects the products under development. The simplest form of a resource predicate simply names the resource:

```
provides { resourceName }
```

This predicate states that the output of an action is a resource bound to the variable *resourceName*. Resource specifications may also be predicates that constrain the state of the resource:

```
requires { resourceName.attributeName op value }
```

Here, *op* is any relational operator. Predicates may also be joined using conjunction and disjunction. In short, resource predicates allow process designers to specify in some detail how a product evolves as a process progresses, as well as what resources are required to produce a product, and the state those resources must have before the process can proceed.

3 Analysis of Resource Flow

What can we learn from analysis of syntactically correct process programs? Analysis helps in two phases of the process life-cycle. First, by analyzing the flow of resources through a process specification, we can identify situations where provided and required resources do not match. This information is useful for validating process specifications against reality; such inconsistencies may indicate gaps in process capture and understanding.

Second, resource analysis can also point out potential areas of improvement in the process being modeled. Inconsistencies between provided and required resources signal a potential for re-engineering to make the process more effective. For example, if a sequence of actions does not have a resource flowing from one action to the next, it may be possible to perform those actions concurrently.

In the following sections, we examine in detail the kinds of inconsistencies that can exist in a specification and their potential impact on a process. Then, we discuss the design of a tool for detecting these inconsistencies in PML specifications.

3.1 Categories of Resource Inconsistencies

Inconsistencies can be classified into several situations:

1. A resource is provided by an action that does not require any resources. This situation (termed a “miracle”) could represent a modeling error where the modeler failed to capture an action’s inputs; or, it could represent a real situation where the actor generates something like a document from (intangible) ideas:

```
action describe_problem {
    /* requires inspiration */
    provides { problem_description }
}
```

2. A resource is required by an action that does not provide any resources. This situation (termed a "black hole") could represent a legitimate activity, such as a task that requires the actor to read certain documents and develop an "understanding" of their contents; the action produces no tangible results, but is worthwhile nevertheless:

```
action understand_problem {
    requires { problem_description }
    /* Provides nothing tangible */
}
```

3. A resource is required, but a different resource is provided. Occasionally, this situation (termed a "transformation") represents a modeling error, but is more often the desired result: an action consumes some resources in the production of another. A simple example happens when a document is assembled from different sections: the action requires each section, and provides the completed document:

```
action submit_design_report {
    requires { use_cases && architecture }
    provides { design_report }
}
```

4. Required resource not provided. In this situation, an action requires a resource that is not provided by any preceding action:

```
action a { provides { r } }
action b { requires { s } }
```

5. A provided resource is never used. An action might provide a resource that is never required by a subsequent action:

```
action a { provides { r && s } }
action b { requires { r } }
```

   Inconsistencies due to unprovided or unrequired resources are not necessarily errors: an unrequired resource could indicate an action that represents an output of a process; an unprovided resource could indicate a point where the process receives input from another process.

6. A provided resource does not match a subsequent resource requirement. Here, the resource is not missing, but rather in the wrong state:

```
action a { provides { r.status == 1 } }
action b { requires { r.status == 2 } }
```

3.2 Analysis Tool Design

Our analysis tool, called `pmlcheck`, is designed to complement the PML compiler. The compiler generates executable models, useful for simulation and enactment, and `pmlcheck` can tell the process engineer interesting things about these models.

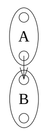
```
function check-if-provided (node,resource,start)
  visited [node] := true
  status [node] := unknown

  if $node \neq start$ and $resource \in provided$ [node] then
    status [node] := true
    necessary [node] [resource] := true
  else
    for pred in predecessors [node] do
      if visited [node] = false then
        check-if-provided (pred,resource,start)
      end if
      status [node] := $\phi_{node}$ (status [node], status [pred])
    end for
  end if
end function
```

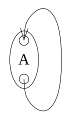
Figure 1. Basic algorithm used by `pmlcheck`.

To compute the flow of resources though a PML program, `pmlcheck` constructs a *process graph* similar to a control-flow graph in conventional languages. Each atomic *action* becomes a graph node. The graphs for other constructs are easily constructed in a syntax-directed manner:

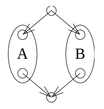
```
sequence {
    A { }
    B { }
}
```



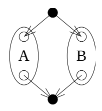
```
iteration {
    A { }
}
```



```
selection {
    A { }
    B { }
}
```



```
branch {
    A { }
    B { }
}
```



The colored nodes in the *branch* graph distinguish it from the *selection* graph, since in the former all paths are always executed and in the latter only one path is executed.

The first three inconsistencies described in the Section 3.1 are *local* to an action node and are easily checked without traversal of the graph. However, the latter three inconsistencies require *global* knowledge of resources and therefore require a graph traversal.

The basic algorithm used to check if a required resource is provided is given in Figure 1. The algorithm performs a depth-first search of the process graph looking for a node that provides the required resource. The function ϕ_{node} is a decision function that updates the status of a node given its current status and the status of a predecessor. Effectively, ϕ_{node} performs a boolean `and` for a *selection* since a resource must be provided on all paths to be definitely provided, and performs a boolean `or` for a *branch* since it is enough that the resource be provided on any path since all paths are guaranteed to be executed. The algorithm also records those provided resources that were found during the search. This information is used to determine which resources are provided, but never required.

3.3 Further Design Considerations

Rather than using a separate analyzer, we could require that global consistency be enforced at compile time, as many modern programming languages do. However, such a policy is generally not desirable. First, it is not necessary: useful process analysis and enactment are possible without global consistency. Second, it is not always possible. Process capture is an iterative process that uncovers hidden activities over time, as process understanding emerges. Thus it is desirable to allow specifications that are incomplete or inconsistent. Finally, valid models can be inconsistent, because the underlying process being modeled is inconsistent. An organization's processes may contain useless steps, missing steps, or sequences of activities that do not produce desired results. Nevertheless, it is important to document these processes accurately, to establish a baseline for process redesign. Therefore, the process engineering environment must be tolerant of inconsistencies that exist in the real world.

4 Examples and Results

To assess the effectiveness of `pmlcheck`, we analyzed two software development processes: the development process used to conduct Computer Engineering Senior Design projects at Santa Clara University, and a graduate Software Engineering course software development process.

4.1 SCU Senior Design Process

Our first experiment employed `pmlcheck` to aid in the creation of a model of the Santa Clara University Computer Engineering department's senior design project process. The process spells out a set of milestones and deliverables roughly based on Boehm's Anchoring Milestones [7].

We first did an initial capturing of the process in which we simply translated the narrative specification into PML. Then, we used the analysis provided by `pmlcheck` to improve the accuracy of the model by correcting specification errors and elaborating resource specifications.

The first version of the model was a simple translation of the narrative specification into a PML specification. We modeled each milestone as a sequence of actions, each action producing a single deliverable.

The tool reported 63 potential inconsistencies in this initial model (see Table 1). How many of these were actual errors? To determine the answer, we analyzed the reported inconsistencies in detail, categorizing them as follows:

- Specification error—The modeler made a mistake in the program specification such as misspelling a resource name.

- Modeling error—The model did not match the underlying process. For example, an action was out of order or was missing.

- Process error—The model was correct, but the underlying process contained an inconsistency.

- Spurious error—The tool correctly identified an error, but the error was triggered by a previous error.

- No error—The tool incorrectly reported an inconsistency.

Of the 63 reported inconsistencies, three were specification errors where a resource name was misspelled, and two were spurious errors, caused by the specification errors. An additional two were not errors as they represented process output.

The remaining 56 errors were the result of incorrectly modeling some aspect of the process, such as omitting a required or provided resource from an action (42 inconsistencies). These conclusions are summarized in Table 2.

Perhaps most interesting from a process engineering viewpoint, 13 errors were the result of omitting actions to capture and deliver document components as a single document; for example, one sequence was missing a "submit design report" action to assemble the document parts and deliver them as a completed "design report" resource. We used our analysis of the initial version of the model to correct the errors uncovered by `pmlcheck`. In the new model, 12 inconsistencies were reported, none of which were errors, as they represented process input or output.

4.2 Graduate Software Processes

We also used to `pmlcheck` to analyze twenty-four process models developed by graduate software engineering students to describe the class project development process. The intent was to develop formal models of the processes specified by the instructor as narrative text in assignments and lectures, augmented by the students' personal experience. The analysis results are shown in Table 1.

4.3 Discussion

It appears from these experiments that the majority of inconsistencies reported by `pmlcheck` are unprovided or unrequired resources. This is its chief limitation: since PML does not distinguish between provided and required resources and process inputs and outputs, `pmlcheck` takes a conservative approach and reports process inputs as unprovided resources, and outputs as unrequired resources.

Curiously, the Graduate Software Development processes contained only two miracles and no black holes among 95 actions; in contrast, the initial Senior Design model had 28 miracles and 15 black holes. This appears to be the due to careful attention to detail on the part of the three modelers who wrote these specifications. Also, `pmlcheck` reported 67 transformations in the Graduate Software Development processes; 31 of these proved to be specification errors that caused the provided resource to appear to be a new resource rather than a modification of the required resource. This was a surprise: we had anticipated that most actions identified as transformations would

Model	Lines	Actions	Resources	Empty	Unprovided	Unrequired	Miracles	Black Holes	Trans.
Senior Design									
senior_design.pml	204	35	69	1	3	16	28	15	36
senior_design2.pml	290	37	122	0	6	6	0	0	42
Graduate S/W Development									
Architecture.pml	130	16	26	3	5	5	0	0	13
Checkout.pml	11	1	2	0	1	1	0	0	1
Commit.pml	12	1	2	0	1	1	0	0	1
Edit.pml	27	3	6	0	2	2	0	0	3
PostMortem.pml	44	7	4	4	0	2	2	0	3
Update.pml	18	2	4	0	2	2	0	0	2
checkin.pml	13	1	2	0	1	1	0	0	1
checkout.pml	16	1	2	0	1	1	0	0	1
make.pml	13	1	2	0	1	1	0	0	1
milestone1.pml	172	18	36	0	13	13	0	0	8
milestone5.pml	141	15	30	0	10	10	0	0	6
updateANDresolve.pml	22	2	4	0	2	2	0	0	1
Analysis.pml	10	1	3	0	2	1	0	0	1
FunctionalRequirements.pml	10	1	3	0	2	1	0	0	1
Milestone2.pml	59	7	21	0	7	7	0	0	7
Milestone3.pml	45	5	15	0	5	5	0	0	5
NonFunctionalRequirements.pml	10	1	3	0	2	1	0	0	1
OperationalConcept.pml	10	1	3	0	2	1	0	0	1
ProjectLog.pml	10	1	3	0	2	1	0	0	1
RepositoryCheckIn.pml	10	1	3	0	2	1	0	0	1
RepositoryCheckOut.pml	20	2	5	0	3	2	0	0	2
RepositorySynchronize.pml	20	2	5	0	3	2	0	0	2
RiskIdentification.pml	10	1	3	0	2	1	0	0	1
SourceCodeEdit.pml	33	4	6	1	3	3	0	0	3
TOTAL	866	95	193	8	74	67	2	0	67

Table 1. Detailed analysis of the errors reported for all models.

Model	Total	Spec.	Model.	Proc.	Spurious	No Error
original	63	3	56	0	2	2
revised	12	0	0	0	0	12

Table 2. Classification of analysis results for the original and the revised senior design models.

actually transform resources into new resources. Thus, it appears to be useful to optionally flag actions that transform resources for closer examination.

5 Related Work

5.1 Program Analysis

Many of the checks performed by our tool are analogous to those checks performed by optimizing compilers such as `gcc` and static checkers such as `lint`. Optimizing compilers typically warn the user regarding possibly uninitialized variables. Our analysis tool informs the user regarding resources that are required without possibly being provided. As another example, register allocation [8], the process of effectively assigning registers to variables to increase execution speed, requires knowledge of the lifetimes of variables in a program. Such knowledge is obtained by computing when a variable is first and last possibly referenced, which is analogous to determining when a resource is first provided and last required.

Algorithms for analyzing programs described as graphs are well-known [9, 10] as are algorithms for computing properties of the graphs [11]. Finally, other tools to aid the programmer in finding errors in programs include assertion checkers [12] and program slicing tools [13, 14].

5.2 Process Validation

Cook and Wolf [15] discuss a method for validating software process models by comparing specifications to actual enactment histories. This technique is applicable to downstream phases of the software life-cycle, as it depends on the capture of actual enactment traces for validation. As such, it complements our technique, which is an upstream approach.

Similarly, Johnson and Brockman [16] use execution histories to validate models for predicting process cycle times. The focus of their work is on estimation rather than validation, and is thus concerned with control flow rather than resource flow.

Scacchi's research employs a knowledge-based approach to analyzing process models. Starting with a set of rules that describe a process setting and models, processes are diagnosed for problems related to consistency, completeness, and traceability [2]. Conceptually, this work is most closely related to ours; many of the inconsistencies uncovered by `pmlcheck` are also revealed by Scacchi and Mi's *Articulator* [17]. Although PML and the *Articu-*

lator share the same conceptual model of process activity, there are important differences. Their approach is based on knowledge-based techniques, with rule-based process representations and strong use of heuristics. This is a different approach than PML's, which closely resembles conventional programming. Thus, our analysis technique is derived from programming language research.

6 Conclusion

What can we conclude about data-flow analysis of process programs? Data-flow analysis can uncover specification errors, such as misspelled resource names, that can exist in otherwise syntactically correct process specifications. Without analysis, these errors would not be detectable until the process is executed. Also, resource flow analysis can identify inconsistencies between a specification and the process it models. This was shown in Section 4, where our initial Senior Design process model was missing several resource dependencies that were important to the process. Further, data-flow analysis can validate that a process produces the products it was intended to produce. By verifying that the resource flow specified by the process program proceeds correctly from beginning to end, the process designer can validate that the process does in fact transform its inputs into the desired outputs. Finally, in addition to identifying potential errors in a process specification, resource flow analysis can suggest opportunities for redesign of a valid process.

For example, our revised Senior Design model contains six actions that require the "problem statement" resource. Where does this resource come from? At present, the process assumes that the problem statement exists prior to the beginning of the process. But the intent of the process is for professors to provide problems for student teams to solve; so the process should include a phase where students and professors negotiate the problem statement, which then serves as the input to the Conception phase.

6.1 Future Work

A sequence specifies a temporal dependency between actions: a predecessor must be completed before the successor can begin. This implies that the predecessor does something that the successor needs; in other words, the predecessor provides something that the successor requires. If the resources analysis shows that no resource flows between sequential actions, however, it may indicate an opportunity for concurrency. In this case, the process specification indicates a dependency among actions that does not exist.

The opposite situation occurs when the control-flow specification indicates that actions can be performed concurrently, but the resource flow among them requires that they performed in a certain order. This situation may indicate either an error in process capture, or a problem with the process itself.

This suggests a tool for automatically transforming a specification into an equivalent specification based on the resource flow graph. Such a tool would analyze the resource dependencies among actions, then re-arrange their ordering so that the control flow matches the resource flow.

Finally, the analysis of actual PML programs discussed in Section 4 revealed certain deficiencies in PML. Specifically, since PML makes no distinction between resources provided by or required from actions within the process and resources provided by or to the external environment, `pmlcheck` cannot distinguish between an unprovided resource and a process input, and likewise between an unrequired resource and a process output. This suggests the need for an enhancement to PML to allow the process modeler to specify the process inputs and outputs.

References

[1] L. J. Osterweil, Software processes are software too, *Proc. 9th Intl. Conf. Soft. Eng.*, Monterey, CA, 1987, 2–13.

[2] W. Scacchi, Understanding software process redesign using modeling, analysis and simulation, *Softw. Process. Improv. and Pract.*, 5(2–3), 2000, 183–195.

[3] S. M. Sutton, Jr., D. Heimbigner, and L. J. Osterweil, APPL/A: A language for software process programming, *ACM Trans. Softw. Eng. Meth.*, 4(3), 1995, 221–286.

[4] J. Noll and W. Scacchi, Supporting software development in virtual enterprises, *J. Digit. Inf.*, 1(4), 1999.

[5] W. Scacchi and J. Noll, Process-driven intranets: Life-cycle support for process reengineering, *IEEE Inter. Comput.*, 1(5), 1997, 42–49.

[6] P. Mi and W. Scacchi, A knowledge-based environment for modeling and simulating software engineering processes, *ACM Trans. Knowl. Data Eng.*, 2(3), 1990, 283–289.

[7] B. W. Boehm, Anchoring the software process, *IEEE Softw.*, 13(4), 1996, 73–82.

[8] F. C. Chow and J. L. Hennessy, A priority-based coloring approach to register allocation, *ACM Trans. Prog. Lang. Syst.*, 12(4), 1990, 501–536.

[9] A. V. Aho, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools* (Reading, MA: Addison-Wesley, 1986).

[10] J. Ferrante, K. J. Ottenstein, and J. D. Warren, The program dependence graph and its use in optimization, *ACM Trans. Prog. Lang. Syst.*, 9(3), 1987, 319–349.

[11] T. Lengauer and R. E. Tarjan, A fast algorithm for finding dominators in a flowgraph, *ACM Trans. Prog. Lang. Syst.*, 1(1), 1979, 121–141.

[12] D. Jackson, ASPECT: An economical bug-detector, *Proc. 13th Intl. Conf. Soft. Eng.*, Austin, TX, 1991, 13–22.

[13] M. Mock, D. C. Atkinson, C. Chambers, and S. J. Eggers, Improving program slicing with dynamic points-to data, *Proc. 10th ACM Sym. Foun. Soft. Eng.*, Charleston, SC, 2002, 71–80.

[14] M. Weiser, Program slicing, *IEEE Trans. Softw. Eng.*, SE-10(4), 1984, 352–357.

[15] J. E. Cook and A. L. Wolf, Software process validation: Quantitatively measuring the correspondence of a process to a model, *ACM Trans. Softw. Eng. Meth.*, 8(2), 1999, 147–176.

[16] E. W. Johnson and J. B. Brockman, Measurement and analysis of sequential design processes, *ACM Trans. Des. Autom. Electron. Syst.*, 3(1), 1998, 1–20.

[17] W. Scacchi and P. Mi, Process life cycle engineering: A knowlege-based approach and environment, *Int. J. Intell. Syst. Account. Financ. Manage.*, 6(2), 1997, 83–107.

PROCESS STATE INFERENCE FOR SUPPORT OF KNOWLEDGE INTENSIVE WORK

John Noll
Computer Engineering Department
Santa Clara University
500, El Camino Real,
Santa Clara, CA-95053, USA.
email: jnoll@cse.scu.edu

Jigar Shah
Computer Engineering Department
Santa Clara University
500, El Camino Real,
Santa Clara, CA-95053, USA.
email: jdshah@scu.edu

ABSTRACT
Different actors may do the same work in different ways, depending on their preferences and level of expertise. The nature and amount of process support required also varies with the knowledge level of the actors: novice actors may require guidance at each and every stage of the process, while experts like to have a free hand and need guidance only when in doubt.

We describe a descriptive enactment approach, whereby guidance is provided only when asked, rather than actively prescribing a list of actions at every stage of the process. The enactment mechanism also infers the state of the process by examining the state of products created or modified during the execution of the process; therefore, the actor does not have to notify the system of every action he does while performing the process, but the system can still keep track of process progress so that appropriate guidance can be provided when needed.

KEY WORDS
Software Engineering Applications, Cooperative Work Support, Workflow Modeling

1 Introduction

Traditional workflow support approaches are based on pre-defined, formal descriptions of work processes. Also, these approaches are based on the paradigm that the system has to guide actors through each and every stage of the process execution. Hence, the success of these systems has been limited to domains which have routine and highly repetitive processes [7].

Knowledge intensive work is different from routine work in that actors may perform knowledge intensive tasks in different ways, depending on their intuition, preferences, and expertise. For example, novice actors who are performing the work for the first time may not have any knowledge about how to do the work. More experienced actors who have done the work before have some insight about how things should be done. Finally, there are experts, who know the process thoroughly and can readily improvise new solutions to problems. Due to this difference in their respective knowledge levels, different actors may do the same work in different ways. Consequently, the amount and nature of guidance required while doing the work is different. Thus, a system for supporting knowledge intensive work must be flexible, in order to provide support workers with varying expertise.

In this paper, we describe a process support system which is more explanatory in nature than enforcing. Rather than prescribing a list of actions to be performed at each and every stage of the process, we adopt a reactive approach, whereby the actor is provided with guidance only when he explicitly asks for it. The process support system is ‘descriptive’, in the sense that the actor does not even need to inform the system about the activities he has performed while executing a process. Rather, the process enactment engine infers the state of the process by examining the state of products created or modified during the performance of the process’s tasks. Then, if and when an actor requires guidance as to what tasks should (or may) be performed next, the system can use the inferred state to determine the next action to be taken, according to the underlying process model.

To enable this, we use a product centric modeling approach. A process specification lists tasks along with a nominal sequence in which these tasks could be performed. Each task also has a specification of the pre-conditions and post-conditions for its performance, expressed in terms of the state of artifacts used and produced when the task is performed. Using this specification, the current state of the process can be inferred by observing the current state of the products in the environment. Then, when the actor asks for advice, the process support system uses this inferred state and the process specification to provide guidance on what to do next. Since there is no enforcement of the nominal flow of tasks specified in the process model, deviations can be easily supported.

2 Product Based Modeling

A critical feature to the approach presented in this paper is the ability to view a knowledge-intensive processes as a series of actions that use or consume some artifact and produce some artifact on completion.

We model processes using the PML Process Modeling Language [3]. PML provides familiar programming language constructs such as selection, branch, iteration, and sequence to model the recommended order in which actions should be performed.

The PML feature central to the approach in this paper is the ability to view the process as a series of actions that use or consume some artifact and when they are done, produce some artifact. To capture this information, the specification of an action in the process model may be accompanied with predicates that specify the *resources* (products, artifacts) the action produces uses.

The *requires* predicate specifies the state of the resources which are required for the action to be done. As an example, consider,

```
requires { document }
```

The *provides* predicate, on the other hand, specifies the state of a product produced or modified as a side-effect of performing the action:

```
provides { document.spell_checked == "true" }
```

Thus, a PML process model specifies the evolution of the products created during the execution of that process. Hence, while enacting a process, at any given stage, by observing the state of products, we can easily infer the state of process.

As an example, Figure 1 shows a PML model describing a process a student might follow to submit homework via email or hardcopy.

To understand how PML enables flexible process support, we will consider how two actors – one novice, one experienced – might perform this process. This process has several steps: first, create a PDF file from the homework document; then, either submit the PDF file as a hard copy in person, or send it as an email attachment. Email submissions will be acknowledged by a message from the professor.

Consider a student who has never submitted homework via email. This student might interact with a process guidance system at each and every stage of the process, as depicted in Figure 2, and depend on the system completely to guide him through the process. This novice actor informs the system of each and every action he performs while executing the process. The system is able to respond immediately by suggesting the next course of action, by following the process control flow from the last action completed.

In contrast, a student that has submitted homework this way before is familiar with the process and may need little guidance. Consider Figure 3, which shows an expert student interacting with a process support system while submitting his homework.

Since this student knows the process, he knows how to start, and hence does not need (or want) to interact with the system for guidance. He starts the job straightaway and creates a PDF file of his homework document. However, if, after creating the PDF file, he is not sure of how to proceed,

```
process SubmitHomework {
  action create_pdf_file {
    requires { document }
    provides { pdf_file }
    script {
      "Create a PDF formatted version of your
       document. }
  }
  selection {
    action submit_hardcopy {
      requires { pdf_file }
      script {
"Print a copy of your document and turn
it in at the beginning of class." }
    }
    sequence {
       action submit_email {
requires { pdf_file }
provides { ack_message }
script {
  "Create an email message with subject
  identifying the course and assignment
  to which this document applies. Attach
  the pdf_file to this message." }
      }
      action verify_ack {
requires { ack_message }
script {
  "Examine the ack_message to be sure that
  the professor has received your homework
  file." }
      }
    }
  }
}
```

Figure 1. PML Model Process for Submitting Homework

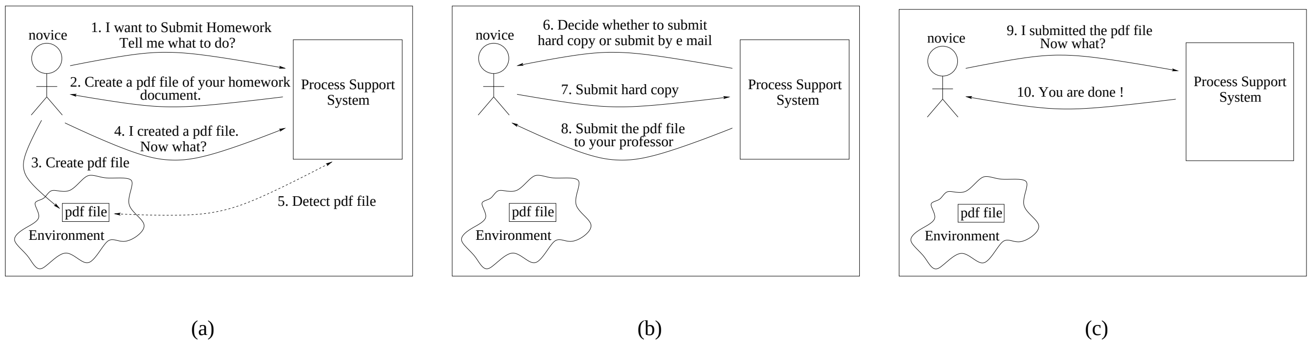

novice
1. I want to Submit Homework
Tell me what to do?
2. Create a pdf file of your homework document.
Process Support System
4. I created a pdf file. Now what?
3. Create pdf file
pdf file
5. Detect pdf file
Environment
(a)
6. Decide whether to submit hard copy or submit by e mail
7. Submit hard copy
8. Submit the pdf file to your professor
(b)
9. I submitted the pdf file
Now what?
10. You are done !
(c)

Figure 2. Novice Interacting with a Process Support System

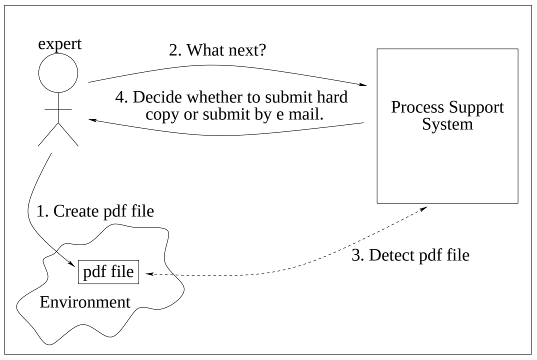


Figure 3. Expert Interacting with a Process Support System

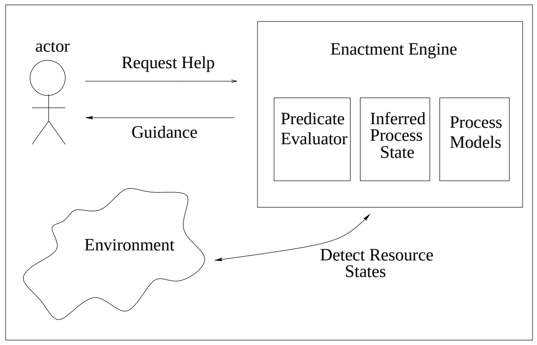


Figure 4. Enactment Architecture

he could consult the system to ask for advice on what to do next. At this point, the system can detect that he has already created a PDF file; using this information, it can respond with the suggestion that he may perform any action that *requires* the PDF file, such as submit the homework by email or submit a hard copy.

The key observation from this scenario is that an expert actor does not want advice at each and every stage of the process, but he may want advice when he is not sure of what to do next. The novice actor, on the other hand, requires assistance with each step.

3 Enactment Mechanism

A process model specifies the evolution of products during the performance of that process. Since the actor is not required to inform the system about his activities, the enactment mechanism has to keep track of the process by monitoring the evolution of products specified in the process model. Since the process model also specifies a order in which the actions could be performed, the enactment engine should can use the process model to provide guidance to the actor, if asked to do so. Figure 4 shows the architecture of the enactment mechanism. In the following sections, we describe the components of this architecture.

3.1 Enactment Engine

The enactment engine parses the PML process model and interprets the process specification. This interpretation involves the following:

- Get the required and provided resources for a given action.
- Determine whether the *requires* and *provides* predicates are satisfied. The predicate evaluator component is used to evaluate the predicates.
- Update the process state to reflect the state of the resources in the environment. To do this, a the engine converts the PML process model into a graph representation, which can be traversed for interpreting the process description. Details on this graph representation can be found in [3].
- When asked, use the process model and the inferred process state to advise on what to do next.

Thus, the main functionality of the enactment engine is to update the process state to reflect the state of resources in the environment and to provide guidance based on this process state and the PML model.

A change in resource state may be the result of the actor completing an action. This may also result in the *requires* predicate of another action to evaluate to true, thus making that action ready to be executed. At this point, if an actor asks for guidance, the enactment engine can suggest to do the action whose *requires* predicate just evaluated to true. Hence to infer the process state based on the state of resources, the enactment engine evaluates all the *requires* and *provides* predicates (using the predicate evaluator) and marks the actions accordingly. This then represents the updated process state.

3.2 Process Guidance

The guidance given by the enactment engine contains a list of all the actions in the process, presented with the same structure as the nominal flow specified in the process model. In addition, all the actions are annotated with a specific state. An action can be in any of the following states:

READY The previous action is DONE, and all the resources *required* by this action are available.

DONE The action has been performed, it's *provides* predicate is true.

BLOCKED The previous action is DONE, but its required resources are not available.

AVAILABLE An action in the AVAILABLE state means that the resources it *requires* are available, but the previous action is not yet DONE.

NONE The previous action is not DONE, and the *requires* predicate is not yet true.

By marking actions as *READY* or *AVAILABLE*, the enactment mechanism provides the following guidance to the actor: "ideally, you *should* do the actions in the *READY* state; you *could* do the actions in the *AVAILABLE* state; or if you really know what you are doing, you can do any action you want (there is no enforcement)".

Figure 5 shows a screenshot of the user interface, which the actor sees when he asks for guidance after completing the action *create_pdf_file* in our example scenario. The user interface exhibits the following features:

- The left pane shows all the actions in the process, presented in the structure specified in the process model. This enables the actor to get a sense of how different actions are structured within the process.

- Each action is accompanied by a color coded icon representing the state of that action. Blue indicates actions which are *done*; red indicates actions which are *blocked*; green indicates actions which are *ready*; and yellow indicates actions which are *available*. For example, in Figure 5, actions *submit_hardcopy* and *submit_email* are *ready*, while action *create_pdf_file* is *done*.

- The engine has determined that action *create_pdf_file* has been done and marked it accordingly.

- The right hand side pane shows the details of *submit_hardcopy* – the resources it requires, the resources it provides, and also a script which informally describes what is to be done to do this action.

- The buttons 'start', 'finish', 'suspend', and 'abort' on the right hand side are for novice actors who want to interact with the system at each and every stage of the process. They can use these buttons to indicate what actions they are doing. The enactment engine updates the process state based on these notifications and shows the updated process state in the left panel.

Thus, when asked for help, the enactment engine can provide the actor with adequate help on what to do next depending on far he is done and on what the process model indicates.

4 Related Work

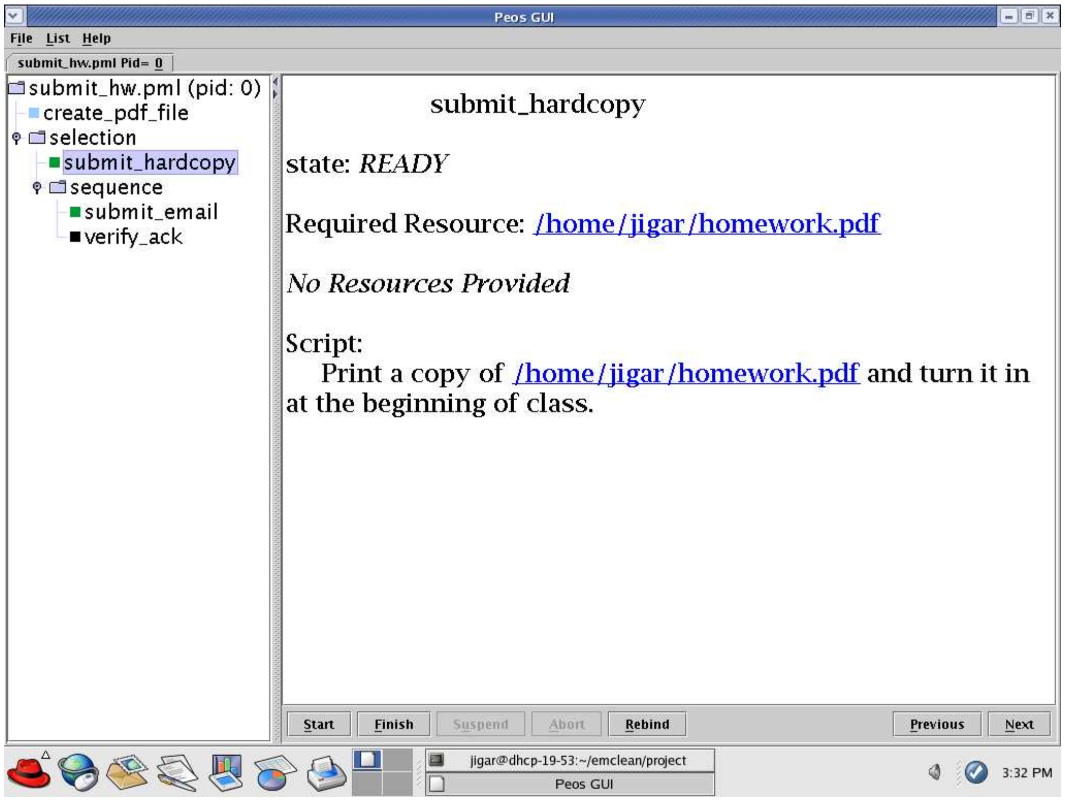
A significant amount of research has been done toward development of adaptive, flexible and dynamic workflow systems. Based on the approaches taken, we can broadly classify the research done so far into five categories.

The first approach views deviations from the normal flow of work as 'exceptions'. In this view, a process has a normal sequence of tasks, and occasional exceptions to the normal sequence where tasks are skipped, performed out of order, An example of this approach is the MILANO system [1], which augments execution of processes modeled as simple Petri-nets with the ability to skip or change the order of places in the net. Another example of the exception handling approach is the PROSYST system [5]. Just as is the case in our approach, users are not forced to satisfy the constraints stated in the process model.

Another approach to handling deviations as exceptions focuses on a consistent and effective evolution of the workflow model as a basic step toward making workflow systems flexible. Such systems allow modification of the workflow models at runtime. For example, Casati and colleagues [4] allow modifications to the flow structure of a running process instance; another approach uses the concept of inheritance to achieve flexible and reusable workflow models [13].

Implicit in the exception-handling view of flexibility is the notion that there is a "right" or "normal" sequence of tasks, and exceptional deviations. In contrast, we view deviations from the nominal sequence as both normal an unexceptional.

Processes are not always well enough understood to be fully specified as a detailed model. To cope with this situation, Jorgensen proposes an approach allowing initially ambiguous process models to be deployed, calling on the actor to interpret the ambiguous parts [9]. Thus, enactment



submit_hardcopy

state: *READY*

Required Resource: /home/jigar/homework.pdf

No Resources Provided

Script:
Print a copy of /home/jigar/homework.pdf and turn it in at the beginning of class.

Figure 5. User Interface

takes place as a dialog between the actor and the enactment mechanism, ultimately leading to a refinement of the model into a complete, unambiguous specification. Sadiq and colleagues [10] also define flexibility as the ability of the process to execute on the basis of a partially defined model where the full specification is made at runtime and maybe unique for each instance.

Many researchers attribute the lack of flexibility in workflow to process model specifications that are too rigid. As an alternative, Glance and colleagues [8] propose a constraint specification language that can be used to specify the goals of the process, without having to specify the order of the activities to be performed. This gives actors maximum flexibility in achieving the process goal: they can select the most appropriate sequence of tasks as long as they do not violate the constraints specified in the process model. Dourish and colleagues [11] also use a constraint based process modeling formalism. Its focus is on mediation between process and action rather than enactment of a process. However, the increased flexibility comes at the cost of providing guidance to the user.

Process mining and plan inference are two research areas that also attempt to infer process data from the state of the environment. Cook and Wolf have applied process mining techniques on software engineering processes [6] They describe three approaches to discover processes from event streams - algorithmic, using neural networks, and a Markovian approach. Application of process mining in the context of workflow management is presented in [2]. This work deals with the problem of generating workflow graphs from workflow events recorded in a workflow log and presents an algorithm to construct such graphs.

Though not directed explicitly to workflow support, plan inference is conceptually similar to our approach of inferring process state from user actions. The goal is to infer intent by observing the actions of the actor. One such application of this idea is discussed in [12], which uses plan inference techniques for providing context sensitive help. However, the goal of this approach is to deduce what plan is being followed, rather than the state of a previously identified plan (or process).

Unlike process mining or plan inference, we start with an existing process model and then infer the state of an instance of this model, thus avoiding some of the computational complexity involved in constructing a model from scratch.

5 Conclusion

We have proposed a process support system, which is more explanatory in nature, rather than enforcing, and which is flexible enough to support actors with varying degrees of expertize, ranging from novices to experts. While the system can support actors interacting with it at each and every stage of the process, it can also support actors who like to have a free hand while doing the work and need guidance only when in doubt. The system is truly ‘descriptive’ in the sense that, the actor does not even need to inform the system about the activities he has performed while executing the process. The system infers the state of the process by inferring the state of products evolved during the execution of the process, and uses this inferred state and the process specification to provide guidance when asked.

We believe that this approach is well suited for supporting knowledge intensive work, where the expertise level of actors ranges from novices to experts. Since the system does not prescribe any flow of work, experts have a free hand in doing the actual work in a way that seems best to them. They are not forced into doing anything. Also, since the system provides guidance when asked for, it is useful for novice actors who may want guidance at every stage of process execution.

Our initial experience with the system has been positive. We feel that the approach we have proposed has potential and plan to conduct more extensive experiments using the proof of concept system we have developed during the course of this research.

Acknowledgments

This work is supported in part by the National Science Foundation under Grant No. IIS-0205679, through subcontract from the University of California, Irvine; and by IBM Faculty Research and Arthur Vining Davis grants awarded through Santa Clara University. No endorsement is implied.

References

[1] A. Agostini and G. D. Michelis. A light workflow management system using simple process models. *Computer Supported Cooperative Work*, 9(3-4):335–363, Aug. 2000.

[2] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. *Lecture Notes in Computer Science*, 1377:469–483, 1998.

[3] D. C. Atkinson and J. Noll. Automated validation and verification of process models. In *Proceedings of the 2003 IASTED Conference on Software Engineering Applications*, Marina Del Rey, CA, USA, November 2003.

[4] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow Evolution. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, pages 438–455, 1996.

[5] C.Cugola. Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models . *IEEE Transactions on Software Engineering*, 24(1), 1998.

[6] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[7] P. Dourish. Process descriptions as organizational accounting devices: The dual use of workflow technologies. In *Proceedings of the 2001 International ACM SIGROUP Conference on Supporting Group Work*, pages 52–60, Boulder, Colorado, USA, Oct. 2001. ACM Press.

[8] N. S. Glance, D. S. Pagani, and R. Pareschi. Generalized process structure grammars (GPSG) for flexible representations of work. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, pages 180–189. ACM Press, 1996.

[9] H. D. Jorgensen. Interaction as a framework for flexible workflow modelling. In *Proceedings of the 2001 International ACM SIGROUP Conference on Supporting Group Work*, pages 32–41, Boulder, Colorado, USA, Oct. 2001. ACM Press.

[10] P. Mangan and S. Sadiq. On Building Workflow Models for Flexible Processes. In X. Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, Melbourne, Australia, 2002. ACS.

[11] P.Dourish, J.Holmes, A.MacLean, P.Marqvardsen, and A.Zbyslaw. Freeflow: Mediating between representation and action in workflow systems. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work*, pages 190–198. ACM Press, 1996.

[12] K.-J. Quast. Plan recognition for context sensitive help. In *Proceedings of the 1st international conference on Intelligent user interfaces*, pages 89–96. ACM Press, 1993.

[13] G. Yang. Process inheritance and instance modification. In *Proceedings of the 2003 International ACM SIGROUP Conference on Supporting Group Work*, pages 229–238, Sanibel Island, Florida, USA, Nov. 2003. ACM Press.

Process Breakdown, Recovery and Articulation

This section contains the following two chapters that examines issues that arise when complex processes or hidden workflows that span one or more enterprises breakdown, and must be repaired or re-articulated in order for the process to complete.

Margaret Elliott, and Walt Scacchi, Free Software Development: Cooperation and Conflict in a Virtual Organizational Culture, in S. Koch (ed.), *Free/Open Source Software Development,* 152-172, Idea Publishing, Pittsburgh, PA, 2005.

Chris Jensen and Walt Scacchi, Collaboration, Leadership, Control, and Conflict Negotiation in the NetBeans.org Software Development Community, *Proc. 38th. Hawaii Inter. Conf. Systems Science,* Waikoloa Village, HI, January 2005.

Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture

Margaret S. Elliott
Institute for Software Research
University of California, Irvine
Irvine, CA 92697
949 824-7202
melliott@ics.uci.edu

Walt Scacchi
Institute for Software Research
University of California, Irvine
Irvine, CA 92697
949 824-4130
wscacchi@ics.uci.edu

August 2003
Previous version: May 2003

Revised version submitted to:
S. Koch (ed.), *Free/Open Source Software Development*, IDEA Publishing, 2004.

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. Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at the Santa Clara University; Chris Jensen, Mark Bergman, and Xiaobin Li at the UCI Institute for Software Research, and also Julia Watson at The Ohio State University are collaborators on the research project that produced this chapter.

Free Software Development: Cooperation and Conflict in A Virtual Organizational Culture

1 Introduction

Free/open source software development (F/OOSD) projects are growing at a rapid rate. The SourceForge Web site estimates 600,000+ users with 700 new ones joining every day and a total of 60,000+ projects with 60 new ones added each day. Thousands of F/OOSD projects have emerged within the past few years (DiBona, *et al*., 1999; Pavlicek, 2000) leading to the formation of globally dispersed virtual communities (Kollock and Smith, 1999). Examples of open software projects are found in the social worlds that surround computer game development; X-ray astronomy and deep space imaging; academic software design research; business software development; and Internet/Web infrastructure development (Elliott, 2003; Elliott and Scacchi, 2002; Elliott and Scacchi, 2003; Scacchi 2002a, 2002b). Working together in globally distributed virtual communities, F/OSS developers communicate and collaborate using a wide range of web-based tools including Internet Relay Chat (IRC) for instant messaging, CVS for concurrent version control (Fogel, 1999), electronic mailing lists, and more (Scacchi, 2002b).

Proponents of F/OSS claim advantages such as improved software validity, simplification of collaboration, and reduced software acquisition costs. While some researchers have examined F/OOSD using quantitative studies exploring issues like developer defect density, core team size, motivation for joining free/open source projects, and others (Koch and Schneider, 2000; Mockus *et al.*, 2000, 2002), few researchers have explored the social phenomena surrounding F/OOSD (Berquist, M. and J. Ljungberg, 2001; Mackenzie *et al.,* 2002). While the importance of understanding the culture of FOSS developers has been discussed in popular literature (Pavlicek, 2000; Raymond, 2001), no researchers have articulated the work culture of F/OOSD in a virtual organization. In this chapter, we present the results of a virtual ethnography to study the work culture and F/OOSD work processes of a *free software* project, GNUenterprise (GNUe) (http://www.gnuenterprise.org). We identify the beliefs and values associated with the free software movement (Stallman, 1999a) which are manifested into the work culture of the GNUe community and we show the importance of computer-mediated communication (CMC) such as chat/instant messaging and summary digests in facilitating teamwork, resolving conflicts, and building community.

The free software movement promotes the production of free software that is open to anyone to copy, study, modify, and redistribute (Stallman, 1999b). The Free Software Foundation (FSF) was founded by Richard M. Stallman (known as RMS in the F/OSS community) in the 1970s to promote the ideal of freedom and the production of free software, based on the concept that source code is fundamental to the furthering of computer science, and that free source code is necessary for innovation to flourish in computer science (DiBona *et al*., 1999). It is important to distinguish between the terms free software (Stallman, 1999a) and open source (DiBona *et al*., 1999). Free software differs from open source in its philosophical orientation. RMS feels that the difference is in their values, their ways of looking at the world.

> "For the Open Source movement, the issue of whether software should be open source is a practical question, not an ethical one. As one person put it, `Open source is a development methodology; free software is a social movement.' For the Open Source movement, non-free software is a suboptimal solution. For the Free Software movement, non-free software is a social problem and free software is the solution.
> http://www.fsf.org/philosophy/free-software-for-freedom.html

A popular expression in the free software culture is "Think free speech, not free beer." The FSF promotes the use of the General Public License (GPL) for free software development as well as other similar licenses (http://www.gnu.org/licenses/license-list.html). While the majority of open source projects use the GPL, alternative licenses suggested by the Open Source Inititative (OSI) are also available (see http://www.opensource.org).

The free software movement has spawned a number of free software projects all adhering to the belief in free software and belief in freedom of choice (http://www.gnu.org) as part of their virtual organizational culture. As with typical organizations (Martin, 1992, Schein, 1992), virtual organizations develop work cultures, which have an impact on how the work is completed. Each of these free software projects basically follow the suggested work practices outlined on the FSF Web site (see http://www.fsf.org) for initiating and maintaining a free software project. However, each project may also have cultural norms generic to their particular virtual organization. Subsequently, there is a need for better articulation of how these free software beliefs and values may influence F/OOSD. Managers and developers of F/OOSD projects would benefit from an understanding of how the culture of the free software movement influences work practices. In this chapter, we present empirical evidence from the GNUe case study of the influence that beliefs and values of the free software movement have on teamwork, tool choices, and conflict resolution in a free software development project. The results show a unique picture of one free software community and how they rely on CMC for software and documentation reviews, bug fixes, and conflict resolution. As with all qualitative research (Yin, 1992; Strauss and Corbin, 1990), we do not intend to portray a generalized view of all free software development projects. However, research has shown that many F/OOSD projects follow similar procedures (Scacchi 2002b). Future research will show how closely the GNUe work culture resembles that of other free software projects.

In the section 2 we present the GNUe project, followed by research methods in section 3. In section 4, we present background and in section 5 we discuss the GNUe virtual organizational culture with a conceptual diagram followed a description of the cases in section 6. Next we present a discussion of the data in section 7 followed by recommendations in section 8. We finish the chapter with section 9 on future research and section 10 as conclusions.

2 GNUe Project

GNUe is a meta-project of the GNU (http://www.gnu.org) Project. GNUe is organized to collect and develop free electronic business software in one location on the Web. The plans are for GNUe to consist of:

1. a set of tools that provide a development framework for enterprise information technology professionals to create or customize applications and share them across organizations;
2. a set of packages written using the set of tools to implement a full Enterprise Resource Planning system; and
3. a general community of support and resources for developers writing applications using GNUe tools. The GNUe Web site advertises it as a "Free Software project with a corps of volunteer developers around the world working on GNUe project."

GNUe is an international virtual organization for software development (Crowston and 2002; Noll and Scacchi, 1999) based in the U.S. and Europe. This organization is centered about the GNUe Web portal and global Internet infrastructure that enables remote access and collaboration. As of the writing of this paper, GNUe contributors consist of 6 core maintainers (co-maintainers who head the project); 18 active contributors; and 18 inactive contributors. The 6 core maintainers share various tasks including the monitoring of the daily IRC, accepting bug fixes to go into a release, testing software, documentation of software, etc. Another task for these core maintainers appears to be that of trying to resolve conflicts and answering questions regarding GNUe. For the duration of the IRC logs that we studied, several core maintainers were on the IRC almost the entire day. Companies from Austria, Argentina, Lithuania, and New Zealand support paid contributors, but most of the contributors are working as non-paid participants.

3 Research Methods

This ongoing ethnography of a virtual organization (Hine, 2000; Olsson, 2000) is being conducted using the grounded theory approach (Strauss and Corbin, 1990) with participant-observer techniques. The sources of data include books and articles on OSSD, instant messaging (Herbsleb and Grinter, 1999, Nardi *et al.*, 2000) transcripts captured through IRC logs, threaded email discussion messages, and other Web-based artifacts associated with GNUe such as Kernel Cousins(summary digests of the IRC and mailing lists – see http://kt.zork.net). This research also includes data from email and face-to-face interviews with GNUe contributors, and observations at Open Source conferences. The first author spent over 100 hours studying and perusing IRC archives and mailing list samples during open and axial coding phases of the grounded theory. During open coding the first case study presented here was selected as representative of the strong influence of cultural beliefs on GNUe software development practices. The selection of cases was aided by the indexing of each Kernel Cousin into sections labeled with a topic. For example, we read through all Kernel Cousins looking mainly at the indices only and found the following title "Using Non-Free Tools for Documentation" in (http://kt.zork.net/GNUe/gnue20011124_4.html). Hyperlinks from this cousin pointed us to a similar case where non-free tools were being used for documentation of code. The third case was found by coding the last file in the three day series for the case two debate. In the third case, a newcomer asks for help regarding the use of GNUe and we show how cooperation and community building are facilitated by the use of IRC.

The initial research questions that formed the core of the grounded theory are:

1) How do people working in virtual organizations organize themselves such that work is completed?
2) What social processes facilitate open source software development?
3) What techniques are used in open source software development that differ from typical software development?

We began this research with the characterization of open source software communities as communities of practice. A community of practice (COP) is a group of people who share similar goals, interests, beliefs, and value systems in a common domain of recurring activity or work (Wenger, 1998). An alternative way of viewing groups with shared goals in organizations is to characterize them as organizational subcultures (Trice and Beyer, 1993; Schein, 1992; Martin, 2002). As the grounded theory evolved, we discovered rich cultural beliefs and norms influencing "geek" behavior (Pavlicek, 2000). This led to us to the characterization of the COPs as virtual organizations having organizational cultures.

We view culture as both objectively and subjectively constrained (Martin, 2002). In a typical organization, this means studying physical manifestations of the culture such as dress norms, reported salaries, annual reports, and workplace furnishings and atmosphere. In addition, subjective meanings associated with these physical symbols are interpreted. In a virtual organization, these physical cultural symbols are missing, so we focus on unique types of accessible manifestations of the GNUe culture, such as Web site documentation and downloadable source code. We use the grounded theory approach to build a conceptual framework and develop a theory regarding the influence of organizational culture on software development in a free software project (Strauss and Corbin, 1990). Data collection includes the content analysis of Web site documents; IRC archives; mailing lists; kernel cousins; email interviews; and observations and personal interviews from open source conferences.

During the open coding, we interpreted books and documents as well as Web site descriptions of the OSSD process. We discovered strong cultural overtones in the readings and began searching for a site to apply an analysis of how motivations and cultural beliefs influenced the social process of OSSD. We selected GNUe as a research site because it exemplified the essence of free software development providing a rich picture of a virtual work community with a rapidly growing piece of downloadable free software. The GNUe Web site offered access to downloadable IRC archives and mailing lists as well as lengthy documentation - all facilitating a virtual ethnography. We took each IRC and kernel cousin related to the three cases and applied codes derived from the data (Strauss and Corbin, 1990). We used a text editor to add the codes to the IRC text logs using [Begin and End] blocks around concepts we identified such as "belief in free software". In this way, we discovered the relationships shown in Figure 1. During the axial coding phase of several IRC chat logs, mailing lists and other documentation, we discovered relationships between beliefs and values of the work culture and manifestations of the culture. In the next section we discuss the organizational culture perspective and studies relating to conflict resolution in cyberspace.

4 Background

In this section, we discuss the organizational culture perspective that is used to characterize the work culture of the virtual organization, GNUe. Next we discuss literature related to conflict resolution in virtual communities.

4.1 Organizational Culture Perspective

Popular literature has described open source developers as members of a “geek” culture (Pavlicek, 2000) notorious for nerdy, technically savvy, yet socially inept people, and as participants in a “gift” culture (Berquist and Ljungberg, 100; Raymond, 2001) where social status is measured by what you give away. However, no empirical research has been conducted to study FOSS developers as virtual organizational cultures (Martin, 2002; Schein, 1992) with beliefs and values that influence decisions and technical tool choices. Researchers have theorized the application of a cultural perspective to understand IT implementation and use (Avison and Myers, 1995), but few have applied this to the workplace itself (Dube´ and Robey, 1999; Elliott, 2000).

Much like societal cultures have beliefs and values manifested in norms that form behavioral expectations, organizations have cultures that form and give members guidelines for “the way to do things around here.” An organizational culture perspective (Martin, 2002; Schein, 1992; Trice and Beyer, 1993) provides a method of studying an organization’s social processes often missed in a quantitative study of organizational variables. Organizational culture is a set of socially established structures of meaning that are accepted by its members (Ott, 1989).

The substances of such cultures are formed from ideologies, the implicit sets of taken-for-granted beliefs, values, and norms. Members express the substance of their cultures through the use of cultural forms in organizations -- acceptable ways of expressing and affirming their beliefs, values and norms. When beliefs, values, and norms coalesce over time into stable forms that comprise an ideology, they provide causal models for explaining and justifying existing social systems. In a virtual organization, cultural beliefs and values are manifest in norms regarding communication and work issues (if a work-related community like OSSD) and in the form of electronic artifacts – IRC archives, mailing list archives, and summary digests of these archives as Kernel Cousins. Most organizational culture researchers view work culture as a consensus-making system (Ott, 1989; Trice and Beyer, 1993; Schein, 1992). In the GNUe study, we apply an integration perspective (Martin, 2002) to the GNUe community to show how beliefs and values of the free software movement tie the virtual organization together in the interests of completing the GNUe free software project (See Elliott and Scacchi, 2003 for a detailed report of the GNUe study). We present the GNUe virtual organization as a subculture of the FSF inculcating the beliefs and values of the free software movement into their everyday work.

4.2 Conflict Resolution in Virtual Communities

Researchers have attempted to understand conflict resolution in virtual communities (Kollock and Smith, 1996; Smith, 1999) in the areas of online communities and in the game world. Many others have studied conflict resolution in common work situations such as computer-supported cooperative work (CSCW) (Easterbrook, 1993). For our purposes, we are interested in virtual communities and how they resolve conflicts so this discussion does not include studies on conflict management tools.

Smith (1999) studied conflict management in MicroMUSE, a game world dedicated to the simulation and learning about a space station orbiting the earth. There were two basic classes of participants: users and administrators. Disputes arose in each group and between the two groups regarding issues like harassment, sexual harassment, assault, spying, theft, and spamming. These problems emerged due to the different meanings attributed to MicroMUSE by its players and administrators and due to the diverse values, goals, interests, and norms of the group. Smith concluded that virtual organizations have the same kinds of problems and opportunities brought by diversity as real organizations do, and that conflict is more likely, and more difficult to manage than in real communities. Factors contributing to this difficulty are: wide cultural diversity; disparate interests, needs and expectations; nature of electronic participation (anonymity, multiple avenues of entry, poor reliability of connections and so forth); text-based communications; and power asymmetry among users. On the contrary, in our GNUe study, we found that text-based communications via the archival text (IRC and Kernel Cousins) enabled the conflict resolution.

Kollock and Smith (1996) explored the implications of cooperation and conflict in Usenet groups emphasizing the importance of recognizing the free-rider problem. In a group situation where one person can benefit from the product or resource offered by others, each person is motivated not to contribute to the joint effort, instead free-riding on others' work. The authors do a detailed analysis of this free-rider problem and give suggestions for how to avoid it in Usenet groups. For example, they suggest bandwidth be used judiciously, posting useful information and refraining from posting inappropriate information as a way to better manage bandwidth. Success on a Usenet group also depends on its members following cultural rules of decorum. We explore the topic of following cultural rules in the next section by presenting the conceptual framework of the GNUe study.

5 Conceptual Diagram of GNUe Virtual Organizational Culture

The substance of a culture is its ideology – shared, interrelated sets of emotionally charged beliefs, values and norms that bind people together and help them to make sense of their worlds (Trice and Beyer, 1993). While closely related to behavior, beliefs, values, and norms are unique concepts as defined below (Trice and Beyer, 1993):

- **Beliefs** – Express cause and effect relations (i.e. behaviors lead to outcomes).
- **Values** – Express preferences for certain behaviors or for certain outcomes.
- **Norms** – Express which behaviors are expected by others and are culturally acceptable

As members of the FSF, free software developers share an ideology based on the belief in free software and the belief in freedom of choice. These beliefs are espoused in the literature on free software (Williams, 2002). The values of cooperative work and community are inferred from this research. Figure 1 shows a conceptual diagram of the GNUe case study. The causal conditions consist of the beliefs (free software and freedom of choice) and the values (cooperative work and community). The phenomenon is the free software development process – its formal and informal work practices. The interaction/action occurs on the IRC and mailing lists. It consists of 1) the conflict over the use of a non-free tool to create a graphic diagram of the emerging GNUe system design, 2) the conflict over the use of a non-free tool to create GNUe documentation. The consequences are: 1) building community; 2) resolution of conflicts with a reinforcement of the beliefs; and 3) teamwork is strengthened. The beliefs, values, and norms are described below; the consequences are presented in the Discussion section.

5.1.1 Belief in Free Software

The belief in free software appears to be a core motivator of free software developers. GNUe developers extol the virtues of free software on its Web site and in daily activity on the IRC logs. The FSF Web site has many references to the ideological importance of developing and maintaining free software (See http://www.fsf.org). This belief is manifested in electronic artifacts such as the Web pages, source code, GPL license, software design diagrams, and accompanying articles on their Web site and elsewhere. The data analysis of the GNUe cases showed that this belief varies from moderate to strong in strength. For example, those who have a strong belief in free software refuse to use any form of non-free software (such as a commercial text editor) for development purposes. The variation in strength of this variable becomes the focal point of case two.

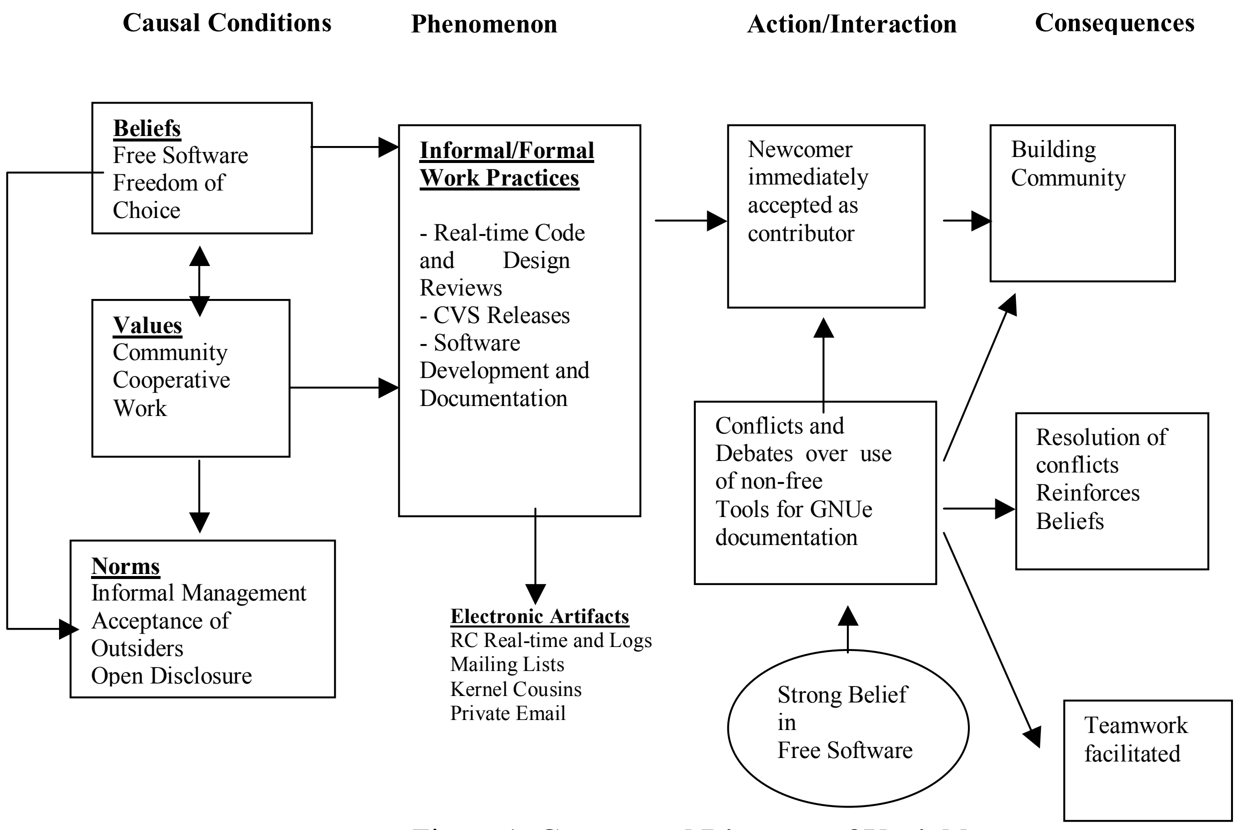


Figure 1. **Conceptual Diagram of Variables**

5.1.2 Belief in Freedom of Choice

Open source software developers are attracted to the occupation of OSSD for its freedom of choice in work assignments. Both paid and unpaid GNUe participants to some degree can select the work they prefer. This belief is manifested in the informal methods used to assign or select work in an open source project. During an interview with one of the core contributors of GNUe, Derek, at a LinuxWorld conference in August 2002, we asked how assignments were made and monitored. Derek answered with:

"The number one rule in free software is 'never do timelines or roadmaps'."

The belief in freedom of choice also refers to the ability to select the tool of choice to develop free software. Some OSS developers believe that a mix of free versus non-free software tools is acceptable when developing free software, while others adhere to the belief in free software only.

5.1.3 Value in Community

The beliefs in free software and freedom of choice foster a value in community building as part of routine work. This value is evident in the IRC archives when newcomers join GNUe offering suggestions, or pointing out bugs, and GNUe contributors quickly accept them as part of the community. For example, when frequent contributors (insiders) have a problem with procedures or code related to free versus non-free software, the maintainers rally around the insider trying to convince him that a temporary use of non-free software is OK.

5.1.4 Value in Cooperative Work

The GNUe community's beliefs in free software and freedom of choice combined with the value in community foster a value in cooperative work. As with previous researchers (Easterbrook, 1993; Kollock and Smith, 1996; Smith, 1999), our results indicate that conflict arises during the course of cooperative work. GNUe contributors work cooperatively to resolve conflicts through the use of IRC and mailing lists.

5.1.5 Open Disclosure

Open disclosure refers to the open content of the GNUe Web site including the software source code, documentation, and archived records of IRC, kernel cousins, and mailing list interchanges. The GNUe contributors join others online via IRC on a daily basis and record the conversations for future reference. All documentation and source code are easily downloaded from the GNUe Web site and user criticism is welcomed by frequent GNUe maintainers.

5.1.6 Informal Management
The entire GNUe virtual organization is informal. There is no lead organization or prime contractor that has brought together the alliance of individuals and sponsoring firms as a network virtual organization. It is more of an emergent organizational form where participants have in a sense discovered each other, and have brought together their individual competencies and contributions in a way whereby they can be integrated or made to interoperate (Crowston and Scozzi, 2002). The participants come from different small companies or act as individuals that collectively move the GNUe software and the GNUe community forward. Thus, the participants self-organize in a manner more like a meritocracy (Fielding, 1999). There is a flow to the work determined by participants' availability.

5.1.7 Immediate Acceptance of Outsider Critiques
In the GNUe organization, outsiders who have not visited the GNUe IRC before, can easily join the discussion and give criticisms of the code or procedures. Sometimes this criticism revolves around the use of free versus non-free tools and other times it is related to attempts to fix bugs in the code. In either case, the GNUe maintainers who discuss these critiques respect and respond to outsiders reviews with serious consideration even without knowing the reviewer's credentials.

6 GNUe Case Study
The GNUe case study consists of the analysis of three cases of software development communication over the IRC. They involve 1) the debate over the use of a non-free tool for creation of a graphic; 2) the debate over the use of a non-free tool for GNUe documentation creation and maintenance; and 3) the initiation of a newcomer who fixes bugs in realtime. Each case will be described briefly in this section. For a more detailed description, see (Elliott and Scacchi, 2003).

6.1 Case One – Use of Non-Free Graphic Tool for Documentation

In this section we present the first case study that reveals a trajectory of a conflict and debate over the use of a non-free tool to create a graphic on the GNUe Web site (See http://www.gnuenterprise.org/irc-logs/gnue-public.log.25Nov2001). This exchange takes place on November 25, 2001 on the IRC channel and ends the next morning. This example illustrates the ease with which a newcomer comes onboard and criticizes the methods used to produce a graphical representation of a screenshot on the GNUe Web site. CyrilB, an outsider to GNUe, finds a graphic that was created using Adobe Photoshop, a non-free graphical tool. He begins the interchange with a challenge to anyone onboard stating that "it is quite shocking" to see the use of non-free software on a free software project. He exhibits a **strong belief in free software,** which causes a debate lasting a couple of days. Table 1 displays the total number of contributors and the number of days of the conflict. Eight of the nine regular GNUe contributors were software developers and one was working on documentation. The infrequent contributors drifted on and off throughout the day – sometimes lurking and other times involved in the discussion.

Total Contributors	**Regular Contributors**	**Infrequent Contributors**	**Number of Days**
17	9	8	1

Table 1 – Contributors and Duration of Conflict in Case One

The **strong belief in free software** of the outsider leads to conflict among those insiders who have a moderate view of the use of free software for GNUe software development. A daylong debate ensues among the Neilt, creator of the graphic, CyrilB, and other GNUe contributors regarding the use of a non-free software tool to create a graphic for a GNUe screenshot for Web site documentation.

CyrilB uses his **strong view of belief in free software** to promote the spirit of the free software movement by exclaiming that images on the gnuenterpise.org Web site seem to be made with non-free Adobe software. His reaction provokes strong reactions from GNUe contributors:

> "I hope I'm wrong: it is quite shocking…We should avoid using non-free software at all cost, am I wrong? **(Strong BIFS-1)**"

Reinhard responds with a **moderate view of belief in free software:**

> "Our main goal is to produce good free software. We accept contributions without regarding what tools were used to do the work especially we accept documentation in nearly any form we can get because we are desparate for documentation." **(Moderate View BIFS-1).**

Once CyrilB has pointed out the use of the non-free graphic, Neilt, who originally created the GNUe diagram using Adobe Photoshop, joins the IRC, reviews the previous discussion on the archived IRC, and returns to discuss the issue with Reinhard and CyrilB. A lively argument ensues between Neilt and others with onlookers contributing suggestions for the use of free tools to develop the Adobe graphic.

Meanwhile Maniac, who has been "listening" to this debate, jumps in and gives technical details about a PNG image. Then Reinhard and Neilt agree that CyrilB had a valid point since a PNG has no vector information stored and so it would be difficult to use free software to edit the graphic. These exchanges illustrate how participants use the IRC medium to support and enable the cooperative work needed to resolve this issue. It also conveys the community spirit and cooperative work ethic that is a value in the GNUe work culture. They both agree to wait until CyrilB comes back to give more suggestions for an alternative.

Outside critiques of software and procedures used during development are common to the GNUe project. One of the norms of the work culture is **immediate acceptance of outsider**

contributions. Eventually, Neilt, the creator of the non-free graphic questioned CyrilB's qualifications and was satisfied when he learned that CyrilB was a member of the European Free Software Foundation. However, he was willing to fix the graphic prior to the revelation of CyrilB's credentials.

Consequences of the debate are a reinforcement of the **belief in free software, value in community,** and **value in cooperative work;** and a recreation of a Web site graphic with free software to replace the original created with a non-free software tool.

6.2 Case Two – Use of Non-Free Software for GNUe Documentation

The second case study explores project insider review of the procedures and practices for developing GNUe documentation (See http://www.gnuenterprise.org/irc-logs/gnue-public.log.15Nov2001 for the full three day logs). Once again the debate revolves around polarized views of the use of non-free tools to develop GNUe documentation. In this case, Chillywilly, a frequent contributor, balks at the need to implement a non-free tool on his computer in order to edit the documentation associated with a current release. Even though his colleagues attempt to dissuade him from his concerns by suggesting that he can use any editor – free or non-free- to read the documentation in HTML or other formats, Chillywilly refuses to back down from his stance based on a **strong belief in free software**. This debate lasts three days. Table 2 displays the number of contributors and their classification for participation in case two. This case exemplifies the fierce adherence to the belief in free software held by some purists in the free software movement and how it directs the work of the day. While the three day debate reinforces beliefs and values of the culture, at the same time, it ties up valuable time which could have been spent writing code or documentation, yet it contributes to community building.

Total Contributors	Regular Contributors	Infrequent Contributors	Number of Days
24	9	15	3

Table 2 – Contributors and Duration of Conflict over Documentation

In order to understand this example, some background information is needed. The GNUe core maintainers selected a free tool to use for all documentation called *docbook* (http://www.docbook.org). DocBook is based on an SGML document type definition which provides a system for writing structured documents using SGML or XML. However, several GNUe developers as of November 15, 2001 were having trouble with its installation. Consequently, they resorted to using lyx tool to create documentation (http://www.lyx.org)...

The problem with lyx is that even though it was developed as a free software tool, its graphical user interface (GUI) requires the installation of a non-free graphics package (called *libxforms*). Chillywilly gets upset with the fact that he has to install non-free software in order to read and edit GNUe documentation. A lengthy discussion ensues with debates over which tool to use for GNUe documentation. This debate lasts for three days taking up much of the IRC time until Chillywilly finally gives up the argument. The strength in the **belief in free software** drives this discussion. The debate and its resolution also illustrate the tremendous effort by developers to collaborate and work cooperatively through the use of the IRC channel. Although the discussion is heated at moments, a sense of fun also pervades. Chillywilly begins on the November 14, 2001 IRC with an observation that a fellow collaborator, jamest, has made documents with lyx:

```
Action: chillywilly trout whips jamest for making lyx docs
Action: jcater troutslaps chillywilly for troutslapping jamest for making easy to do docs
<chillywilly> lyx requires non-free software
<Maniac> lyx rules
<chillywilly> should that be acceptable for a GNU project?
```

> <jcater> chillywilly: basically, given the time frame we are in, it's either LyX documentation with this release, or no documentation for a while (until we can get some other stinking system in place)
> <jcater> pick one :)
> <chillywilly> use docbook then

…

> <Maniac> lyx's graphics library is non-gpl **(i.e. non-free software)**
> <chillywilly> I'm not writing your docs for you
> <Maniac> this is an issue the developers are aware of but do not, at this time, have the time to rectify
> <chillywilly> Maniac: because they are **** KDE nazis
> <chillywilly> that's who the original lyz authors are matthias, et. al.
> <Maniac> well, my understanding is, they are working toward UI independance, to make it able to use differnt toolkits ie. kde, gnome, xyz as time/coding permit

Maniac questions chillywilly's incessant reminders about using non-free software as though this myopic view of free software development is unnecessary. Chillywilly continues his debate showing his **strong view of free software**.

Reinhard agrees with chillywilly as do others, but in order to complete the documentation, they agree to use an interim solution. Chillywilly is so adamantly opposed to the use of non-free software that he references Richard Stallman as part of his reasoning – "**I will NOT install lyx and make vrms unhappy**". This passage shows how RMS is considered the "guru" of the free software movement. Eventually chillywilly sends an email to the mailing list:

> "OK, I saw on the commit list that you guys made some LyX documents. I think it is extremely ***that a GNU project would require me to install non-free software in order to read and modify the documentation. I mean if I cannot make vrms happy on my debian system them what good am I as a Free Software developer? Is docbook really this much of a pain? I can build html versions of stuff on my box if this is what we have to do. This just irks me beyond anything. I really shouldn't have to be harping on this issue for a GNU project, but some ppl like to take convenience over freedom and this should not be tolerated… Is it really that unreasonable to request that we not use something that requires ppl to install non-free software? Please let me know. (Chillywilly, mailing list)"

A lengthy discussion of technical issues unrelated to the documentation problem ensues. Meanwhile Jcater has sent a reply to Chillywilly's message to the mailing list.:

"I would like to personally apologize to the discussion list for the childish email you recently received. It stemmed from a conversation in IRC that quickly got out of hand. It was never our intention to alienate users by using a non-standard documentation format such as LyX. Writing documentation is a tedious chore few programmers enjoy. The developers of the GNUe client tools are no exception…The upcoming release was originally planned for this past weekend. James and I decided to postpone the release… LyX was chosen because it is usable and, more importantly, installable. After many failed attempts at installing the requirements for docbook, James and I made the decision that LyX-based documentation with the upcoming 0.1.0 releases was better than no documentation at all…
PPS, By the way, Daniel, using/writing Free software is NOT about making RMS happy or unhappy. He's a great guy and all, but not the center of the free universe, nor the motivating factor in many (most?) of our lives. For me, my motivation to be here is a free future for my son (Jcater, mailing list)."

The belief in freedom is a motivating factor for Jcater as stated above, even freedom for his son.

6.3 Case Three – Newcomer Asking for Help with GNUe Installation

In this example, mcb30 joins the IRC as a newcomer who wants to install and use GNUe business applications for his small business in England (http://www.gnuenterprise.org/irc-logs/gnue-public.log.16Nov2001). In addition, he offers his services as a contributor and immediately starts fixing bugs in realtime. This case is a good example of the community building spirit of GNUe since mcb30 is immediately accepted by frequent contributors especially because he posts significant bug fixes very rapidly.

```
<mcb30> Is anyone here awake and listening?
<reinhard> yes
<mcb30> Excellent.  I'm trying to get a CVS copy of GNUe up and running for the first(ish)
time - do you mind if I ask for a few hints?
<reinhard> shoot away :)
<reinhard> btw what exactly are you trying to run?
<reinhard> as "GNUe" as a whole doesen't exist (yet)
<reinhard> GNUe is a meta-project (a group of related projects)
<mcb30> OK - what I want to do is get *something* running so I can get a feel for what
there is, what state of development it's in etc. - I'd like to contribute but I need to know
what already exists first!
<reinhard> ok cool
<reinhard> let me give you a quick overview
<mcb30> I have finally (about 5 minutes ago) managed to get "setup.py devel" to work
```

```
properly - there are 2 bugs in it
<mcb30> ok
```

Mcb30 goes offline and continues to fix bugs. He then comes back and suggests that he has a patch file to help

```
<mcb30> I've got a patch file - who should I send it to?  jcater?
<reinhard> jcater or jamest
<mcb30> ok, will do, thanks
<reinhard> mcb30: btw sorry if i tell you things you already know :)
<mcb30> don't worry - I'd rather be told twice than not at all! :-)
<reinhard> people appearing here in IRC sometimes have _very_ different levels of
information :)

<reinhard> look at examples/python/addrbook.py
<mcb30> excellent, thanks!
<mcb30> will have a play around
<reinhard> mcb30: i will have to thank you
<reinhard> mcb30: we are happy if you are going to help us
<reinhard> gotta leave now
```

Later mcb30 comes back to the IRC and posts code that he wrote to fix a problem and several frequent contributors thank him and say that they wish they could hire him for pay. As with the first case, contributors immediately accept them into the "club" and, as the chat unfolds, they ask him for his credentials, motivation, and location (mcb30 is an educational consultant for IT in the English school system).

7 Discussion

The three examples from the GNUe case study will be discussed in this section in relation to the three main themes found in the data: realtime teamwork, building community, and conflict resoluton. Each example comes from a detailed conding and content analysis of the IRCs.

7.1 Building Community

Kollock (1996) suggests that there are design principles for building a successful online community such as identity persistence. He draws upon the work of Godwin (1994) showing that allowing users to resolve their own disputes without outside interference and providing institutional memory are two principles for making vritual communities work. Applying these principles to the GNUe project shows that disputes are resolved simultaneously via IRC, and recorded in IRC archives as a form of institutional memory. In the GNUe virtual community, the community is continuously changing (when newcomers join even if for a brief time) yet the core maintainers are dedicated for long periods of time. Here is a quote from Derek, a core maintainer, who believes that the IRC helps them sustain their community:

> “Many free software folks think IRC is a waste of time as there is 'goofing off', but honestly I can say its what builds a community. I think a community is necessary to survive. For example GNUe has been around for more than 3 years. I can not tell you how many projects have come and gone that were supposed be competition or such. I put our longevity solely to the fact that we have a community.” (Derek, email interview (2002))

7.2 Conflict Resolution

In the two conflict resolution GNUe cases presented here, both issues resulted in a solution by debate on the IRC and mailing lists. In the first case, the contributor who created the graphic with ADOBE photoshop agreed to change it in the future using a free tool. In the second case, chillywilly stopped badgering his co-workers about the use of a non-free graphics package to complete documentation. His colleagues essentially told him to get back to work and use a text editor if he is so worried about the use of *lyx* until they all can use the free software *docbook*. In both cases, the conflicts were resolved in a reasonable amount of time via the IRC exchanges. At the same time, the beliefs in free software are reinforced by people defending their positions and this, in turn, helps to perpetuate the community.

7.3 Facilitating Teamwork

In each case there was evidence that as the day proceeded on the IRC, people were going offline to experiment with free software that would help to resolve the conflict (i.e. a free graphics package and a free text editor). Many infrequent contributors or newcomers who were lurking and watching the problem unfold on the IRC, also gave technical advice for a tool to use to solve the problem. The realtime aspect of the work clearly facilitates the teamwork since people could simultaneously work together solving a technical problem. In the third case, a newcomer who was having trouble with the GNUe installation was directed by a maintainer to the original author of the code. Surprisingly, the original author joins the IRC that day and discusses the bugs with mcb30.

8 Practical Implications

We have shown that the persistent recording of daily work using instant messaging (IRC) and Kernel Cousins can serve as a community building avenue. Managers of open source might benefit from incorporating these CMC mediums into their computing infrastructures. It assists employees in conflict management and also binds the groups together by reinforcing the organizational culture. As illustrated in the non-conflict GNUe example, the IRC serves as an expertise Q&A repository. The author of the software quickly emerged and mcb30 was able to gain detailed knowledge of how the system works. In addition, the IRC enables realtime software design and debugging. As F/OOSD projects proliferate, managers should consider the benefits of using an IRC to facilitate software development and to help build a community..

9 Future Research

We plan to continue with the analysis of GNUe data and compare the results with other free software communities. Likewise, we expect to find similar beliefs and values in some open source projects and plan to explore this phenomena. In this way, we can assertain whether GNUe is in fact a unique culture (Martin, 2002) or whether other free software projects have similar software development processes. A review of other GNU projects shows evidence of proselitization of beliefs in free software (http://www.gnu.org/projects/projects.html). In addition, in the LINUX community, there is an ongoing dispute about using Bitkeeper (non-free) versus CVS (free) as a case management system. Other future research of interest is to determine if having strong beliefs and values regarding free software contribute to a successful, productive F/OOSD community.

10 Conclusions

Previous CSCW research has not addressed how the collection of IRC messaging, IRC transcript logs, and email lists, and periodic digests (Kernel Cousins) can be collectively mobilized and routinely used to create a virtual organization that embodies, transmits, and reaffirms the cultural beliefs, values, and norms such as those found in free software projects like GNUe. Strong organizational cultural beliefs in an F/OOSD virtual community combined with persistent recordation of chat logs tie a group together and helps to build a community and perpetuate the project. The beliefs in freedom, free software, and freedom of choice create a special bond for the people working on free software projects. These beliefs foster the values of cooperative work and community-building. Schein's (1990) theory of organizational culture includes revelation of underlying assumptions of cultural members that are on a mostly unconscious level. In the GNUe world, the underlying assumptions of cooperative work and community-building become ingrained in the everyday work practices in their pursuit of an electronic business and ERP system implemented as free software. These beliefs and values enhance and motivate acceptance of outsiders' criticisms and resolution of conflict despite the distance separation and amorphous state of the contributor population.

11 References

Avison, D. E., & Myers, M. D. (1995). Information Systems and Anthropology: An Anthropological Perspective on IT and Organizational Culture. *Information Technology and People, 8*, (43-56).

Berquist, M. and J. Ljungberg, The power of gifts: organizing social relationships in open source communities, *Info. Systems J.*, 11(4), 305-320, October 2001.

Crowston, K., & Scozzi, B. (2002). *Exploring Strengths and Limits on Open Source Software Engineering Processes: A Research Agenda.* Paper presented at the *2nd Workshop on Open Source Software Engineering*, Orlando, Florida.

DiBona, C., Ockman, S., & Stone, M. (1999). *Open Sources: Voices from the Open Source Revolution*. Sebastol, CA: O'Reilly & Associates Inc.

Dubé, L., & Robey, D. (1999). Software Stories: Three Cultural Perspectives on the Organizational Practices of Software Development. *Accounting, Management and Information Technologies, 9*(4), 223-259.

Easterbrook, S. (Ed.). (1993). *CSCW: Cooperation or Conflict*. New York: Springer-Verlag.

Elliott, M. (2000). *Organizational Culture and Computer-Supported Cooperative Work in a Common Information Space: Case Processing in the Criminal Courts*. (Vol. Unpublished Dissertation). Irvine: University of California, Irvine.

Elliott, M. (2003). *The Virtual Organizational Culture of a Free Software Development Community.* Paper presented at the *3rd Workshop on Open Source Software*, Portland, Oregon.

Elliott, M., & Scacchi, W. (2002). Communicating and Mitigating Conflict in Open Source Software Development Projects, Working Paper: Institute for Software Research, University of California, Irvine, http://www.ics.uci.edu/~melliott/commossd.htm

Elliott, M., & Scacchi, W. (2003). Free Software: A Case Study of Software Development in a Virtual Organizational Culture. Working Paper: Institute for Software Research, University of Calfornia Irvine, http://www.ics.uci.edu/~wscacchi/Papers/New/Elliott-Scacchi-GNUe-study-DRAFT.pdf

Feller, J., & Fitzgerald, B. (2002). *Understanding Open Source Software Development*. N.Y.: Addison-Wesley.

Fielding, R. T. (1999). Shared Leadership in the Apache Project. *Communications of the ACM, 42*(4), 42-43.

K. Fogel (1999). *Supporting Open Source Development with CVS.* Scottsdale, AZ: Coriolis Press.

Easterbrook, S. M., Beck, E. E., Goodlet, J. S., Plowman, M., Sharples, M., & Wood, C. C. (1993). A Survey of Empirical Studies of Conflict. In S. M. Easterbrook (Ed.), *CSCW: Cooperation or Conflict?,* 1-68. London: Springer-Verlag.

Godwin, M. (1984). Nine Principles for Making Virtual Communities Work. *Wired,* 2.06, 72-73.

Gregory, K. (1983). Native-view Paradigms: Multiple Cultures and Culture Conflicts in Organizations. *Administrative Science Quarterly, 28*, 359-376.

James D. Herbsleb, Rebecca E. Grinter (1999). Splitting the Organization and Integrating the Code: Conway's Law Revisited. ICSE 1999: 85-95.

Hine, C. (2000). *Virtual Ethnography*. London: Sage.

Koch, S., & Schneider, G. (2000). *Results from Software Engineering Research into Open Source Development Projects Using Public Data*, Wirtschaftsuniversitat Wien.

Kollock, P. (1996). Design Principles for Online Communities, *The Internet and Society: Harvard Conference Proceedings,* http://sscnet.ucla.edu/soc/faculty/kollock/papers/design.htm.

Kollock, P., & Smith, M. (1996). Managing the Virtual Commons: Cooperation and Conflict in Computer Communities. In S. Herring (Ed.), *Computer-Mediated Communication: Linguistic, Social, and Cross-Cultural Perspectives*, 109-128, Amsterdam: John Benjamins.

Kollock, P., & Smith, M. A. (1999). Communities in Cyberspace. In M. A. Smith & P. Kollock (Eds.), *Communities in Cyberspace* (pp. 3-25). New York, NY: Routledge.

Mackenzie, A., Rouchy, P., & Rouncefield, M. (2002). *Rebel Code? The Open Source 'Code' of Work.* Paper presented at the Open Source Software Development Workshop, February 25-26, 2002, Newcastle-upon-Tyne, UK.

Martin, J. (2002). *Organizational Culture: Mapping the Terrain*. Thousand Oaks: Sage Publications.

Mockus, A., Fielding, R., & Herbsleb, J. (2002). Two Case Studies on Open Source Software Development: Apache and Mozilla. *ACM Trans. Software Engineering and Methodology,* 11(3), 309-346.

Mockus, A., Fielding, R. T., & Herbsleb, J. (2000). A Case Study of Open Source Software Development: The Apache Server. *Proc. 22nd Intern. Conf. on Software Engineering*, 263-272, Limerick, IR.

Nardi, B., Whittaker, S., Bradner, E. Interaction and Outeraction: Instant Messaging in Action. (2000). Proceedings CSCW 2000.

Noll, J., & Scacchi, W. (1999). Supporting Software Development in Virtual Enterprises. *Journal of Digital Information,* 1(4), http://jodi.ecs.soton.ac.uk/.

Olsson, S. (2000). *Ethnography and Internet: Differences in Doing Ethnography in Real and Virtual Environments.* Paper presented at the IRIS 23, Laboratorium for Interaction Technology, University of Trollhattan Uddevalla.

Ott, J. (1989). *The Organizational Culture Perspective*. Pacific Grove, CA: Brooks/Cole.

Pace, R. C. (1990). Personalized and Depersonalized Conflict in Small Group Discussions: An Examination of Differentiation. *Small Group Research, 21*(1), 79-96.

Pavlicek, R. G. (2000). *Embracing Insanity: Open Source Software Development*. Indianapolis, IN: SAMS Publishing.

Raymond, E. S. (2001). *The Cathedral & The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA: O'Reilly & Associates.

Robey, D., & Azevedo, A. (1994). Cultural Analysis of the Organizational Consequences of Information Technology. *Accounting, Management, and Information Technology,* 4(1), 23-37.

Sawyer, S. (2001). Effects of Intra-Group Conflict on Packaged Software Development Team Performance. *Information Systems Journal, 11*, (155-178).

Scacchi, W. (2002a). *Open EC/B: A Case Study in Electronic Commerce and Open Source Software Development* (Technical Report). Irvine, CA: University of California, Irvine.

Scacchi, W. (2002b). Understanding Requirements for Developing Open Source Software Systems. *IEE Proceedings - Software, 149*(2), 24-39.

Schein, E. (1990). Organizational Culture. *American Psychologist, 45*, 109-119.

Schein, E. H. (1991). The Role of the Founder in the Creation of Organizational Culture. In P. J. Frost, L. F. Moore, M. R. Louis, C. C. Lundberg, & J. Martin (Eds.), *Reframing Organizational Culture* (pp. 14-25). Newbury Park, CA: SAGE Publications, Inc.

Schein, E. H. (1992). *Organizational Culture and Leadership*. San Francisco: Jossy-Bass.

Smith, A. D. (1999). Problems of Conflict Management in Virtual Communities. In M. A. Smith & P. Kollock (Eds.), *Communities in Cyberspace* (pp. 134-163). New York, NY: Routledge.

Stallman, R. (1999a). *Free Software Foundation Brochure*. Cambridge, MA: Free Software Foundation.

Stallman, R. (1999b). The GNU Operating System and the Free Software Movement. In C. DiBona, S. Ockman, & M. Stone (Eds.), *Open Sources: Voices from the Open Source Revolution* (pp. 53-70). Sebastopol, CA: O'Reilly & Associates, Inc.

Strauss, A. L., & Corbin, J. (1990). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques*. Newbury Park, CA: Sage Publications.

Trice, H. M., & Beyer, J. M. (1993). *The Cultures of Work Organizations*. Englewood Cliffs, NJ: Prentice Hall.

Williams, S. (2002). *Free as in Freedom: Richard Stallman's Crusade for Free Software*. Sebastopol, CA: O'Reilly & Associates.

Yin, R. K. (1994). *Case Study Research, Design and Methods.* 2nd ed. Newbury Park: Sage Publications.

Collaboration, Leadership, Control, and Conflict Negotiation in the *Netbeans.org* Open Source Software Development Community

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

Abstract

Large open source software development communities are quickly learning that, to be successful, they must integrate efforts not only among the organizations investing developers within the community and unaffiliated volunteer contributors, but also negotiate relationships with external groups hoping to sway the social and technical direction of the community and its products. Leadership and control sharing across organizations and individuals in and between communities are common sources of conflict. Such conflict often leads to breakdowns in collaboration. This paper seeks to explore the negotiation of these conflicts, collaborative efforts, and leadership and control structures in the Netbeans.org community.

Keywords

Collaboration, Conflict Negotiation, Leadership, Process, Open Source Software Development, Netbeans.org

1. Introduction

Is open source software development (OSSD) best characterized as being strictly cooperative, or as cooperative and in conflict at the same time [Easterbrook 1993]? Conflict clearly arises during sustained software development efforts [e.g., Sawyer 2001]. But previous studies of conflict associated with Internet-based communities has focused attention to that found in specific OSSD projects operating as virtual organizations [Elliott and Scacchi 2003], as non-profit foundations [O'Mahony 2004], or in online discussion communities [Smith 1999]. None of these studies specifically help us understand the kinds of conflict, cooperation, and collaboration that arises or is needed to coordinate large-scale OSSD processes and effort in large project communities where corporate sponsorship may be a central facet of OSSD.

NetBeans.org is one of the largest OSSD communities around these days [cf. Jensen and Scacchi 2003]. Netbeans.org is a Java-focused OSSD community backed by Sun Microsystems devoted to creating both an integrated development environment (IDE) for developing large Java-based applications, as well as a platform for development of other software products. Originally started as a student project in 1996, the Netbeans.org project was acquired and subsequently released as an open source community project by Sun, whose Netbeans.org team includes many of the community's core developers. While the issues presented here stem from observations in the Netbeans.org community, they are by no means limited to this community, nor have their challenges been insurmountable.

Our study focuses on three items. First, we identify the objects of interaction among participants in the NetBeans.org community that are media through which collaboration, leadership, control and conflict negotiation are expressed and enacted. Second, we explore relationships arising in NetBeans.org on an intra-community level. Then, we look at relationships between communities like Netbeans.org and other communities and organizations.

2. Objects of Interaction

Much of the development work that occurs in an open source software project centers around the creation, update, and other actions (e.g., copy, move, delete) applied to a variety of software development artifacts. These artifacts serve as coordination mechanisms [Schmidt and Simone 1996, Simone and Mark 1999], in that they help participants communicate, document, and otherwise make sense of what the emerging software system is suppose to do, how it should be or was accomplished, who did what, what went wrong before, how to fix it, and so forth. Furthermore, within a project community these artifacts help coordinate local, project-specific development activities, whereas between multiple project communities, these artifacts emerge as boundary objects [Star 1990] through which inter-community activities and relations are negotiated and revised. The artifacts may take the form of text messages posted to a project discussion list, Web pages, source code directories and files, site maps, and more, and they are employed as the primary media through which software requirements and design are expressed. These "software informalisms" [Scacchi 2002] are especially important as coordination mechanisms in OSSD projects since participants generally are not co-located, they do not meet face-to-face, and authority and expertise relationships among participants is up for grabs.

The NetBeans IDE is intended to support the development of Web-compatible Java applications. In the context of the NetBeans.org project and its role within a larger Web-compatible information infrastructure, additional artifacts come into play within and across projects. These include the content transfer protocols like the HyperText Transfer Protocol (http) which are systematically specified in Internet standards like RFC documents, as well as more narrowly focused communication state controllers associated with remote procedure calls (or remote method invocations). They also include shared data description formats like the HyperText Markup Language (html) and the eXtensible Markup Language (XML), as well as client-side or server-side data processing scripts (e.g., CGI routines). Such descriptions may be further interpreted to enable externally developed modules to serve as application/module plug-ins, which enable secondary or embedded applications to be associated with an OSS system. Other artifacts are brought in from other OSSD projects to serve as project support tools, such as those used to record and store system defect (bug) reports (Issuzilla), email list managers, and even large comprehensive collaborative software development environments and project portals, like SourceCast [Augustin, Bressler, and Smith 2002]. Finally, OSSD projects may share both static and operational artifacts in the course of collaborating or cooperating through mutually intelligible and interoperable development processes, which might take an explicit form like the Java Community Process (JCP), or an implicit and embedded form such as that which emerges from use of project repositories whose contents are shared and synchronized through tools that control and track alternative versions (CVS), bug reports, or Web site content updates.

Accordingly, in order to explore where issues of collaboration, leadership, control and conflict may arise within or across related OSSD projects, then one place to look to see such issues is in how project participants create, update, exchange, debate, and make sense of the software informalisms that are employed to coordinate their development activities. This is the approach taken here in exploring the issues both within the NetBeans.org project community, as well as across the (fr)agile ecosystem [Highsmith 2002] of inter-related OSSD projects that situate NetBeans.org within a Web information infrastructure.

3. Intra-Community Issues

As noted in the first section, NetBeans.org is a large and complex OSSD project. To help convey a sense of the complexity, semi-structured modeling techniques such as rich pictures [Monk and Howard 1998] can be used to provide a visual overview of the context that situates the creation and manipulation of the software informalisms and OSSD processes that can be observed in the NetBeans.org project [Oza, *et al*, 2002]. Figure 1 displays such a rich picture, highlighting the variety of roles that participants in the NetBeans.org project perform, the types of concerns they have in each role, and the development tasks they regularly enact, as part of the configuration of OSSD activities they articulate and coordinate through software informalisms [cf . Simone and Mark 1999].

We have observed at least three kinds of issues arise *within* an OSSD community like NetBeans.org. These are collaboration, leadership and control, and conflict.

3.1. Collaboration

According to the Netbeans.org community Web site, interested individuals may participate in the community by joining in discussions on mailing lists, filing bug and enhancement reports, contributing Web content, source code, newsletter articles, and language translations. These activities can be done in isolation, without coordinating with other community members, and then offered up for consideration and inclusion. As we'll see, reducing the need for collaboration is a common practice in the community that gives rise to positive and negative effects. We discuss collaboration in terms of policies that support process structures that prevent conflict, looking at task completion guidelines and community architecture.

3.1.1. Policies and Guidelines

The NetBeans.org community has detailed procedural guidelines[1] for most common development tasks, from submitting bug fixes to user interface design and creating a new release. These guidelines come in two flavors: development task and design style guidelines. In general, these policies are practiced and followed without question. Ironically, the procedures for policy revision have not been specified.

Precedent states that revisions are brought up on the community or module discussion mailing lists, where they are debated and either ratified or rejected by consensus. Developers are expected to take notice of the decision and act accordingly, while the requisite guideline documents are updated to reflect the changes. In addition, as some communities resort to "public flogging" for failure to follow stated procedures, requests for revision are rare and usually well known among concerned parties, so no such flogging is done within Netbeans.org.

Overall, these policies allow individual developers to work independently within a process structure that enables collaboration by encouraging or reinforcing developers to work in ways that are expected by their fellow community members, as well as congruent with the community process.

3.1.2. Separation of Concerns: an Architectural Strategy for Collaborative Success

Software products are increasingly developing a modular, plug-in application program interface (API) architectural style in order to facilitate development of add-on components that extend system functionality. This strategy has been essential in an open source arena that carries freedom of extensibility as a basic privilege or, in some cases, the right of free speech or freedom of expression through contributed source code. But this separation of concerns strategy for code management also provides a degree of separation of concerns in developer management, and therefore, collaboration.

In concept, a module team can take the plug-in API specification and develop a modular extension for the system using any development process in complete isolation from the rest of the community. This ability is very attractive to third-party contributors in the Netbeans.org community who may be uninterested in becoming involved with the technical and socio-political issues of the community, or who are unwilling or unable to contribute their source code back to the community. Thus, this separation of concerns in the Netbeans.org design architecture engenders separation of concerns in the process architecture. Of course, this is limited by the extent that each module in the Netbeans.org community is dependent on other modules.

Last, volunteer community members have periodically observed difficulties collaborating with volunteer community members. For example, at one point a lack of responsiveness of the (primarily Sun employed) user interface team[2], whose influence spans the entire community, could be observed. This coordination breakdown led to the monumental failure of usability efforts for a period when usability was arguably the most-cited reason users chose competing tools over Netbeans.org. Thus, a collaboration failure gave rise to product failure. Only by overcoming collaboration issues was Netbeans.org able to deliver a satisfactory usability experience[3].

3.2. Leadership and Control

Ignoring internal Sun (and third party) enterprise structure, there are five observable layers of the Netbeans.org community hierarchy. Members may take on multiple roles some of which span several of these layers. At the bottom layer are users, followed by source contributors, module-level managers, project level release managers (i.e. IDE *or* platform), and finally, community level managers (i.e. IDE *and* platform) at the top-most layer. Interestingly, the "management" positions are simply limited to coordinating roles; they carry no other technical or managerial authority. The release manager, for example, has no authority to determine what will be included in and excluded from the release[4]. Nor does s/he have the authority to assign people to complete the tasks required to release the product. The same is true of module and community

1http://www.netbeans.org.org/community/guidelines/

2http://www.netbeans.org/servlets/ReadMsg?msgId=531512&listName=nbdiscuss

3http://www.javalobby.org/thread.jspa?forumID=61&threadID=9550#top

4http://www.netbeans.org.org/community/guidelines/process.html managers. Instead, their role is to announce the tasks that need to be done and wait for volunteers to accept responsibility.

Accountability and expectations of responsibility are based solely on precedent and volunteerism rather than explicit assignment, leading to confusion of the role of parties contributing to development. Leadership is not asserted until a community member champions a cause and while volunteerism is expected, this expectation is not always obvious. The lack of a clear authority structure is both a cause of freedom and chaos in open source development. Though often seen as one of its strengths in comparison to closed source efforts, it can lead to process failure if no one steps forward to perform critical activities or if misidentified expectations cause dissent.

The difficulties in collaboration across organizations within the community occasionally brought up in the community mailing lists stem from the lack of a shared understanding leadership in the community. This manifests itself in two ways: a lack of transparency in the decision making process and decision making without community consent. While not new phenomenon, they are especially poignant in a movement whose basic tenets include freedom and knowledge sharing.

3.2.1. Transparency in the Decision Making Process

In communities with a corporately backed core development effort, there are often decisions made that create a community-wide impact that are made company meetings. However, these decisions may not be explicitly communicated to the rest of the community. Likewise private communication between parties that is not made available on the community Web space or to the forwarded to other members is also hidden. This lack of transparency in decision-making process makes it difficult for other community members to understand and comply with the changes taking place if they are not questioned or rejected. This effect surfaced in the Netbeans.org community recently following a discussion of modifying the release process [cf. Erenkrantz 2003][5].

Given the magnitude of contributions from the primary benefactor, other developers were unsure of the responsibility and authority Sun assumed within the development process. The lack of a clearly stated policy outlining these bounds led to a flurry of excitement when Sun members announced major changes to the licensing scheme used by the community without any warning. It has also caused occasional collaboration breakdown throughout the community due to expectations of who would carry out which development tasks. The otherwise implicit nature of Sun's contributions in relation to other organizations and individuals has been revealed primarily through precedent rather than assertion.

3.2.2. Consent in the Decision Making Process

Without an authority structure, all decisions in development are done through consensus, except among those lacking transparency. In the case of the licensing scheme change, some developers felt that Sun was within its rights as the major contributor and the most exposed to legal threat [6] while others saw it as an attack on the "democratic protection mechanisms" of the community that ensure fairness between participating parties[7]. A lack of consideration and transparency in the decision making process tend to alienate those who are not consulted and erode the sense of community.

3.3. Conflict Resolution

Conflicts in the Netbeans.org community are resolved via community discussion mailing lists. The process usually begins when one member announces dissatisfaction with an issue in development. Those who also feel concern with the particular issue then write responses to the charges raised. At some point, the conversation dissipates- usually when emotions are set aside and clarifications have been made that provide an understanding of the issue at hand. If the problem persists, the community governance board is tasked with the responsibility of resolving the matter.

The governance board is composed of three individuals and has the role of ensuring the fairness throughout the community by solving persistent disputes. Two of the members are elected by the community, and one is appointed by Sun Microsystems. The board is, historically, a largely superficial entity whose authority and scope are questionable and untested. While it has been suggested that the board intercede on a few rare

5http://www.netbeans.org/servlets/BrowseList?listName=nbdiscuss&by=thread&from=19116&to=19116&first=1&count=41

6http://www.netbeans.org/servlets/ReadMsg?msgId=534707&listName=nbdiscuss

7http://www.netbeans.org/servlets/ReadMsg?msgId=534520&listName=nbdiscuss

occasions, the disputes have dissolved before the board has acted. Nevertheless, board elections are dutifully held every six months[8].

Board members are typically prominent members in the community. Their status carries somewhat more weight in community policy discussions, however, even when one member has suggested a decision, as no three board members have ever voted in resolution on any issue, and thus, it is unclear what effect would result. Their role, then, is more of a mediator: to drive community members to resolve the issue amongst themselves. To this end, they have been effective.

4. Inter-Community Issues

As noted earlier, the NetBeans.org project is not an isolated OSSD project. Instead, the NetBeans IDE which is the focus of development activities in the NetBeans.org project community is envisioned to support the interactive development of Web-compatible software applications or services that can be accessed, executed, or served through other OSS systems like the Mozilla Web browser and Apache Web server. Thus, it is reasonable to explore how the NetBeans.org project community is situated within an ecosystem of inter-related OSSD projects that facilitate or constrain the intended usage of the NetBeans IDE. Figure 2 provides a rendering of some of the more visible OSSD projects that surround and embed the NetBeans.org within a Web information infrastructure. This rendering also suggests that issues of like collaboration and conflict can arise at the boundaries between projects, and thus these issues constitute relations that can emerge between project communities in OSSD ecosystem.

With such a framing in mind, we have observed at least three kinds of issues arise *across* OSSD communities that surround the NetBeans.org community. These are communication and collaboration, leadership and control, and conflict resolution.

4.1. Communication and Collaboration

In addition to their IDE, Netbeans.org also releases a general application development platform on which the IDE is based. Other organizations, such as BioBeans and RefactorIT communities build tools on top of or extending the NetBeans platform or IDE. How do these organizations interact with Netbeans.org, and how does Netbeans.org interact with other IDE and platform producing organizations? For some organizations, this collaboration may occur in terms of bug reports and feature requests submitted to the Netbeans.org issue-tracking repository. Additionally, they may also submit patches or participate in discussions on community mailing list or participate in the Netbeans.org "Move the Needle" branding initiative. Beyond this, Netbeans.org participates in the Sun sponsored *Java.net* meta-community, which hosts hundreds of Java-based OSSD projects developed by tens of thousands of individuals and organizations.

A fellow member of the Java.net community, the Java Tools Community, considered by some to be a working group[9] for the Java Community Process, is an attempt to bring tool developers together to form standards for tool interoperability. Thus Netbeans.org, through its relationship with Sun, is a collaborating community in the development of, and through compliance with, these standards, and looks to increasing collaboration with other tool developing organizations.

4.2. Leadership and Control

OSSD generally embrace the notion of choice between software products to build or use. At the same time, developers in any community seek success for their community, which translates to market share.

In some cases, communities developing alternative tools do so in peaceful coexistence, even collaboratively. In other cases, there is a greater sense of competition between rivals. NetBeans and its chief competitor Eclipse (backed largely by IBM) fall into the latter category. Eclipse has enjoyed some favor from users due to performance and usability issues of NetBeans, as well as IBM's significant marketing and development resource contributions. Yet, they have a willingness to consider collaborative efforts to satisfy demands for a single, unified IDE for the Java language that would serve as a platform for building Java development tools and a formidable competitor to Microsoft's .NET. Ultimately, the union was defeated, largely due to technical and organizational differences between Sun and IBM[10], including the inability or

8http://www.netbeans.org.org/about/os/who-board.html

9http://www.internetnews.com/dev-news/article.php/3295991

10http://www.adtmag.com/article.asp?id=8634, and unwillingness to determine how to integrate the architectures and code bases for their respective user interface development frameworks (Swing for NetBeans and SWT for Eclipse).

4.3. Conflict Resolution

Conflicts between collaborating communities are resolved in similar fashion to their means of communication- through discussion between Sun and Eclipse representatives, comments on the Netbeans.org mailing lists, or other prominent technical forums (e.g. Slashdot and developer blogs). Unfortunately, many of these discussions occur after the collaborating developer has moved away from using Netbeans.org (often, in favor of Eclipse). Nevertheless, the feedback they provide gives both parties an opportunity to increase understanding and assists the Netbeans.org community by guiding their technical direction.

5. Discussion and Conclusion

Generally, volunteer Netbeans.org developers expect Sun to provide leadership but not control. People outside the community (e.g. users, former users, and potential users) often voice their concerns in off-community forums (e.g., Slashdot, blogs, etc) rather than NetBeans.org community message boards, due to accountability or visibility barriers (creating an account, logging in accounts), small as they may seem to be. In addition, such message forums may not be a part of such an individual's daily work habits- they're more likely to visit a site like Slashdot.org than the Netbeans.org forum because they are not interested enough in staying abreast of NetBeans developments or participating in the community. Nonetheless, people working in, or interested in joining or studying OSSD projects, must address how best to communicate and collaborate their development processes and effort, how to facilitate or ignore project leadership and control, and how to work you way through conflicts that may or may not be resolvable by community participants.

Overall, we have observed three kinds of coordination and collaborating issues arise within OSSD project communities like NetBeans.org, and three similar kinds of issues arise across OSSD communities that surround NetBeans.org within an ecosystem of projects that constitute a Web information infrastructure. Previous studies of conflict in either OSSD projects have examined either smaller projects, or in virtual communities that do not per se develop software as their focus. As corporate interest and sponsorship of OSSD stimulates the formation of large projects, or else the consolidation of many smaller OSSD projects into some sort of for-profit or not-for-profit corporate enterprise for large-scale OSSD, then we will need to better understand issues of collaboration, conflict, and control in OSSD.

http://www.eweek.com/article2/0,1759,1460110,00.asp

6. Acknowledgments

The research described in this report is supported by grants from the National Science Foundation #ITR-0083075 and #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; and Margaret Elliott at the UCI Institute for Software Research.

7. References

Augustin, L., Bressler, D., and Smith, G., Accelerating Software Development through Collaboration, *Proc. 24th Intern. Conf. Software Engineering*, Orlando, FL, 559-563, May 2002,

Crowston, K. and Scozzi, B., Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings—Software*, 149(1), 3017, 2002.

Easterbrook, S. (ed.), *CSCW: Cooperation or Conflict*, Springer-Verlag, New York, 1993.

Elliott, M. The Virtual Organizational Culture of a Free Software Development Community, *Proc. 3rd Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering*, Portland, OR, May 2003.

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, *Proc. 3rd Workshop on Open Source Software Engineering, 25th Intern. Conf. Software Engineering*, Portland, OR, May 2003.

Highsmith, J., *Agile Software Development Ecosystems,* Addison-Wesley Pub. Co., 2002.

Jensen, C. and Scacchi, W., Automating the Discovery and Modeling of Open Source Software Processes, *Proc. 3rd Workshop on Open Source Software Engineering*, 25th. Intern. Conf. Software Engineering, Portland, OR, May 2003.

Jensen, C. and Scacchi, W., Process Modeling of the Web Internet Infrastructure, submitted for publication, June 2004.

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

O'Mahony, S., Non-profit Foundations and their Role in Community-Firm Software Collaboration, to appear in *Making Sense of the Bazaar: Perspectives on Open Source and Free Software*, J. Feller, B. Fitzgerald, S. Hissam, & K. Lakhani (Eds.), O'Reilly & Associates, Sebastopol, CA, 2004.

Sawyer, S., Effects of intra-group conflict on packaged software development team performance, *Information Systems J.,* 11, 155-178, 2001.

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

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

Schmidt, K., and Simone, C., Coordination Mechnanisms: Towards a Conceptual Foundation of CSCW System Design, *Computer Supported Cooperative Work,* 5(2-3), 155-200, 1996.

Simone, C. and Mark, G., Interoperability as a Means of Articulation Work, *Proc. Intern. Joint Conf. Work Activities Coordination and Collaboration,* San Francisco, CA, 39-48, ACM Press, 1999.

Smith, A.D., Problems of Conflict Management in Virtual Communities. In M.A. Smith and P. Kollock (eds.), *Communities in Cyberspace,* Routledge, New York, 134-163, 1999.

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, 37-54. Pitman, London, 1990.

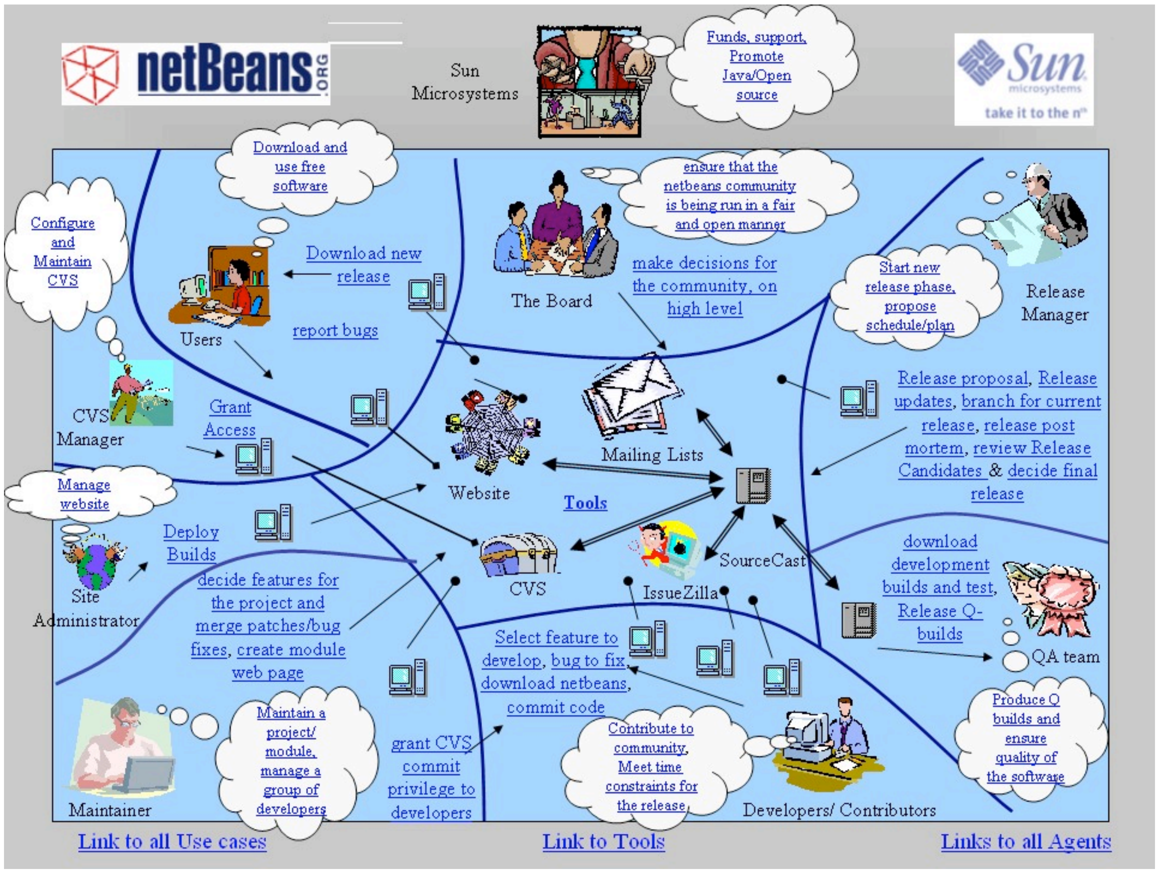

netBeans.org
Sun Microsystems
Funds, support, Promote Java/Open source
Sun microsystems take it to the n
Download and use free software
Configure and Maintain CVS
Download new release
Users
report bugs
ensure that the netbeans community is being run in a fair and open manner
The Board
make decisions for the community, on high level
Start new release phase, propose schedule/plan
Release Manager
CVS Manager
Grant Access
Website
Mailing Lists
Tools
Release proposal, Release updates, branch for current release, release post mortem, review Release Candidates & decide final release
Manage website
Deploy Builds
Site Administrator
CVS
IssueZilla
SourceCast
download development builds and test, Release Q-builds
QA team
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
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 1. A rich picture view of the roles (labeled icons), concerns (clouds), and activities (hyperlinked text) found in the intra-community context of NetBeans.org [cf. Oza, *et al*, 2002].

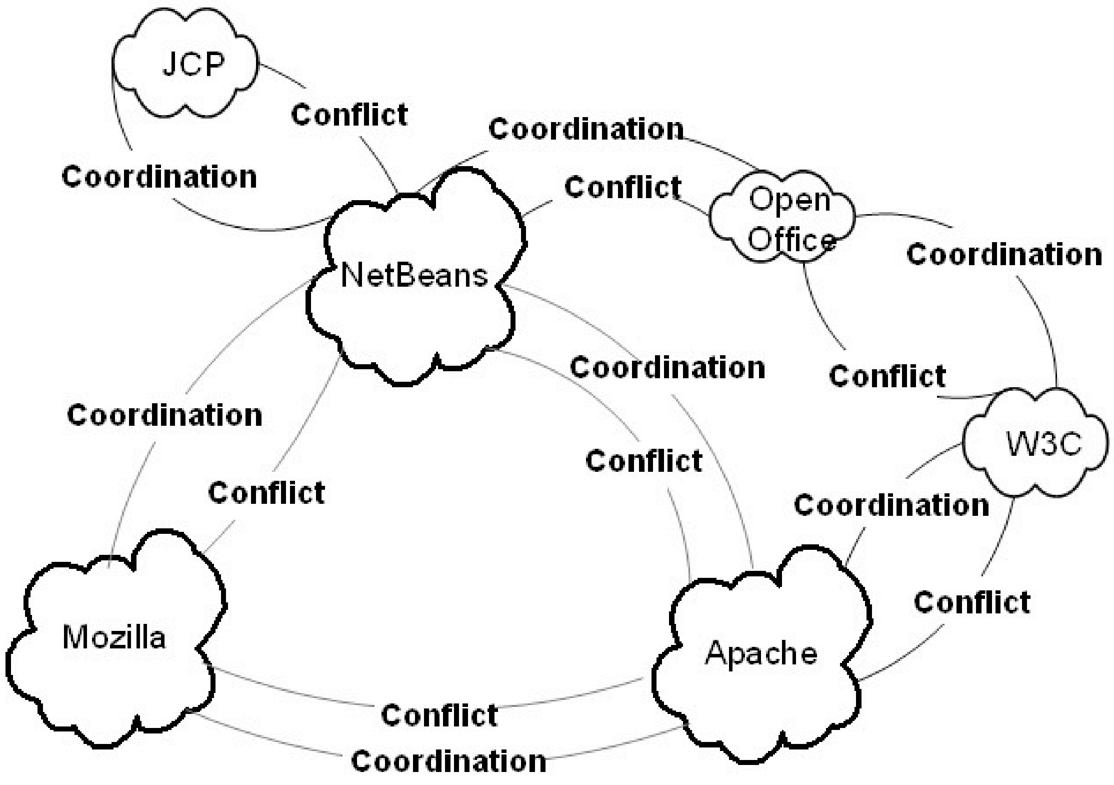


Figure 2. An overview of the inter-community ecosystem for NetBeans.org

Other project deliverables

The research efforts described in the preceding chapters involved the development, user, or enhancement of software tools (and input notations) that support process discovery, modeling, analysis, and enactment. No tools were developed to support process breakdown or recovery. As these “research-grade only” tools (or perhaps better said “prototypes”) are subject to ongoing development and refinement (including abandonment of earlier program versions no longer in use), we recommend using the contact information below to get the most current information, and where appropriate, access to the source code and related documentation (if any) for these tools.

To receive information regarding access to the software developed, used, or enhanced in the course of this research please contact:

For process modeling, analysis, and enactment tools described in this report, contact Dr. John Noll, jnoll@scu.edu.

For process discovery and modeling tools described in this report, contact Chris Jensen, cjensen@ics.uci.edu.