

Chapter 1

OPEN SOURCE SOFTWARE DEVELOPMENT PROCESS MODELING

Jacques LONCHAMP

LORIA, BP 254, 54500 Vandoeuvre-les-Nancy, France (jloncham@loria.fr)

Abstract: This chapter draws attention to software process modeling for open source software development. It proposes a three-layered open source software development process model. Its ‘definitional’ and ‘generic’ levels specify the common features of all fully-fledged open source projects. Its ‘specific’ level allows to describe fine-grained process model fragments characteristic of different open source projects. In this chapter, the specific level is exemplified with the release management process of NetBeans IDE and Apache HTTP Server projects. The underlying modeling approach is SPEM (Software Process Engineering Meta-model) from the OMG. The paper closes with a discussion of the interest of explicit software process models for (1) process understanding and communication, (2) process comparison, reuse, and improvement, (3) process enactment support.

Key words: open source software development process modeling, open source software development, open source software, SPEM, software process modeling.

1. INTRODUCTION

In the last ten years, open source software (OSS) has attracted the attention of not only the practitioner, but also the business and the research communities. In short, OSS is a software whose source code may be freely modified and redistributed with few restrictions, and which is produced by loosely organized, ad-hoc communities consisting of contributors from all over the world who seldom if ever meet face-to-face, and who share a strong sense of commitment [1]. The basic principle for the OSS development process (OSSDP) is that by sharing source code, developers cooperate under a model of systematic peer-review, and take advantage of parallel debugging that leads to innovation and rapid advancement [2]. Today, Linux and

Apache Server are used in respectively 30% and 60% of the Internet's public servers. This demonstrates that OSSDP can produce software of high quality and functionality. Other success stories include Perl, Tcl, Python and PHP programming languages, sendmail mail handler, Mozilla browser, MySQL database server, Eclipse and NetBeans Java integrated development environments. Recently, many organizations have started to look towards OSS and OSSDP as a way to minimize their development efforts by reusing open source code and to provide greater flexibility in their development practices [3].

Two factors may impede this growing interest in OSSDP. First, neither Apache, Mozilla, NetBeans, or any other OSS projects, provide documents on their Web portals that explicitly and precisely describe what development processes are employed. OSS projects do not typically provide explicit process models, prescriptions, or schemes other than what may be implicit in the use of certain development tools for version control and source code compilation. Secondly, most studies that report on OSS projects like Apache and Mozilla [4, 5, 6] provide only informal narrative descriptions of the overall software development process. Such narrative descriptions cannot be easily analyzed, compared, visualized, enacted, and transferred for reuse in other projects. Consequently, developers who want to join an OSS project must discover its underlying development process by using public information sources on the Web. These sources include process enactment information such as informal task prescriptions, community and information structure, work roles, project and product development histories, electronic messages and communications patterns among project participants. Such a discovery approach is very tedious and the variability in development process performance across iterations can blur its results. Similarly, software engineers wanting to start a new OSS project, cannot reuse explicit descriptions or models of the software processes and must discover them through ad hoc trial-and-error. Finally, government agencies, academic institutions and industrial firms which begin to consider OSSDP seek to find what are the best processes or development practices to follow [7]. Explicit modeling of these processes in forms that can be shared, reviewed, modified, and redistributed could be an important contribution to their dissemination and continuous improvement.

The lack of interest in software process modeling techniques observed within and outside of the open source community can be attributed to several reasons. First, it could be argued that OSS projects are ingrained in the hacker culture and represents the antithesis of software engineering [8], with a 'bazaar' [9] or 'development in the wild' style. Just as there is no single development model for proprietary software, neither is there only one detailed model in the OSS world. However, many observations show that at

least a set of common features (roles, activities, tools, etc.) are shared by many fully-fledged OSS projects, i.e., it exists a kind of high level generic software process model that we will try to specify. Secondly, it could be objected that OSS development practices are continuously evolving in these communities whose members operate with a high degree of autonomy. Therefore, modeling such processes at the risk of freezing them could be counterproductive. Our observations suggest that most of these evolutions stay at a very detailed level. Other observations relate evolutions to the infancy of OSS projects and to temporary crisis periods [10]. The most important process fragments of a mature project seem rather stable and often core participants make efforts in order to stabilize and standardize them. Lastly, process model formalisms are often criticized: they would be too complex, too low level and fine-grained and not easy to use and share. We will show that the Software Process Engineering Meta-model (SPEM) [11] from the Object Management Group (OMG) can constitute a good candidate for OSSDP modeling because of two main reasons: first, it provides a minimal set of modeling elements, allowing both structural and behavioral descriptions at different levels of formality and granularity, and second, it uses UML [12] as a notation.

This chapter is organized around four themes. First, some definitions about open source projects are given and the consequences of their diversity from the process modeling perspective are discussed. Second, the SPEM meta model that we use for modeling OSSDPs is presented. Third, our three-layered model proposal with its ‘definitional’, ‘generic’, and ‘specific’ levels is described and illustrated. Finally, the interest of software process modeling of OSS projects for (1) process understanding and communication, (2) process comparison, reuse and improvement, (3) process enactment support, is discussed.

2. OPEN SOURCE PROJECTS

It is frequent to make a distinction between the terms ‘free software’ and ‘open source software’. Free software refers not to price but to liberty to modify and redistribute source code. The Free Software Foundation [13], founded by Richard Stallman, advocates the use of its GNU General Public License (GPL) as a copyright license which creates and promotes freedom. He writes “to understand the concept, you should think of free speech, not free beer” [14]. The term ‘open source’ was coined by a group of people concerned that the term ‘free software’ was anathema to businesses. This resulted in the creation of the Open Source Initiative (OSI) [15]. We use the acronym OSS for both movements for the sake of simplicity and because

both movements share most of their practical goals and follow similar development processes. The OSI definition [16] includes the following criteria:

- *free redistribution*: the license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution; no royalty or fee is required for such sale,
- *source code*: the program must include source code,
- *derived works*: modifications and derived works are allowed, not necessarily subject to the same license as the original work,
- *integrity of author's source code*: derived works must carry different names or version numbers than the original work,
- *no discrimination against persons or groups*,
- *no discrimination against fields of endeavor*,
- *distribution of license*: no need of any additional license,
- *license must not be specific to a product*,
- *license must not restrict other software*,
- *license must be technology-neutral*.

GNU GPL, BSD, Apache, MPL (Mozilla) and Artistic (Perl) licenses are all examples of licenses that conform the OSI definition, unlike Sun Community Source License [8].

Another distinction can be drawn between OSS projects that result from the initiative of a given individual or group of individuals, and OSS projects that are supported by, or organized within, industrial software companies. Examples here include the NetBeans [17] and Eclipse [18] OSS projects that are both developing Java-based interactive development environments, based in part on the corporate support respectively from SUN and IBM. The consequences are noticeable in the way these projects are managed (e.g. composition of the steering committee, decision-making processes) and through the existence of peripheral processes under the exclusive responsibility of the company which backs the project (mainly quality insurance processes). But as will be shown in section 4.2.1 the release process of NetBeans is not deeply impacted by such a corporate support.

OSS projects can also be classified into communities of interest, centered about the production of software for different application domains, such as games, Internet infrastructure, software system design, astronomy, etc. This factor has a low impact on how the software is produced [19].

At the opposite, the project community size is important. Below some critical mass, in terms of active developers, OSSDPs do not match our generic description, as for instance the extreme case of the 'solo work, internal patches' scenario of [20]. The definitional level of our model specifies what a 'fully-fledged' OSS project is, roughly corresponding to the 'team work, external patches' scenario [20].

3. SPEM META MODEL DESCRIPTION

This section presents the Software Process Engineering Meta-model (SPEM) defined by the OMG. SPEM is a meta-model for defining software engineering process models and their components [11]. A tool based on SPEM would be a tool for process model authoring and customizing. The actual enactment of processes, i.e. planning and executing a project using a process model described with SPEM, is not in the scope of this process modeling approach.

The modeling approach is object-oriented and uses UML as a notation [12]. The SPEM specification is structured as a UML profile, i.e. a set of stereotypes, tags and constraints added to the UML standard semantics, and provides also a complete MOF-based meta-model [21]. SPEM is built from the SPEM_Foundation package, which is a subset of UML 1.4, and the SPEM_Extensions package, which adds the constructs and semantics required for software process engineering. Such an approach facilitates exchange with both UML tools and Meta Object Facility (MOF) based tools or repositories. Figure 1-1 shows the four-layered architecture of modeling as defined by the OMG.

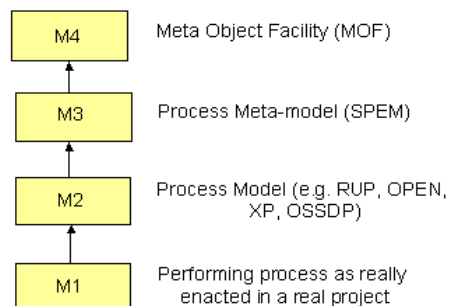


Figure 1-1. The OMG modeling architecture.

At the core of SPEM is the idea that a software development process is a collaboration between abstract active entities, called ‘*Process Roles*’, that perform operations, called ‘*Activities*’, on concrete, tangible entities, called ‘*Work Products*’. More precisely (see Figure 1-2), a process model definition is built out of *Model Elements*. Each *Model Element* describes one aspect of a software engineering process, and can be associated to an *External Description* in some natural language, suitable for a reader of the process model. A *Dependency* is a process-specific relationship between process *Model Elements*. For instance, a ‘precedes’ dependency acts from one *Activity* (or *Work Definition*) to another to indicate start-start, finish-start or finish-finish dependencies between the work described, depending on the

value of the ‘kind’ attribute. *Guidance* is a *Model Element* associated with the major *Model Elements*, which contains additional descriptions for practitioners such as techniques, guidelines and UML profiles, procedures, standards, templates of work products, examples of work products, definitions, and so on.

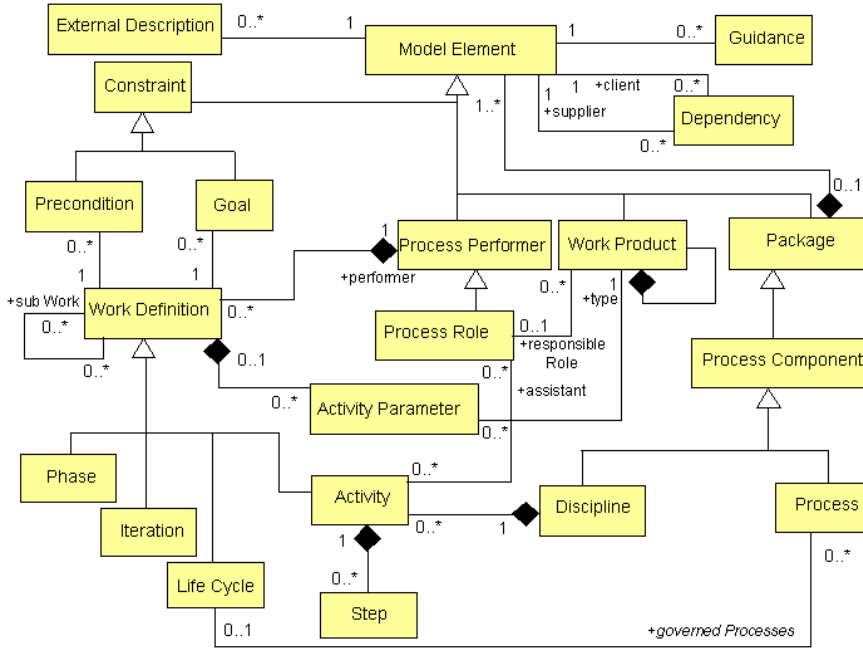


Figure 1-2. SPEM main classes.

A *Work Product* is a description of a piece of information or physical entity produced or used by the *Activities* of the software engineering process. Examples of *Work Products* include models, plans, code, executables, documents, databases, and so on.

A *Work Definition* is a *Model Element* describing the execution, the operations performed, and the transformations enacted on the *Work Products* by the *Process Roles*. *Activity*, *Iteration*, *Phase*, and *Lifecycle* are kinds of *Work Definition*. Any *Work Definition* can be associated with *Preconditions* and with *Goals*. They are both *Constraints*, expressed in terms of the states of the *Work Products* that are *Activity Parameters* to this *Work Definition*. The *Precondition* defines what *Work Products* are needed and in which state they must be to allow the work definition to start. *Activities* are the main element of work. A *Step* is an atomic and fine-grained *Model Element* used to decompose *Activities*. *Activities* are partially ordered sets of *Steps*. The

Life Cycle associated to a *Process* is a *Work Definition* containing all the work to be done in a software engineering process. This *Lifecycle* can be decomposed into *Phases* and/or *Iterations*. A *Phase* is a high-level work definition, bounded by a milestone that can be expressed in terms of *Goals*: which *Work Products* and in which state they must be completed. An *Iteration* is a large-grained *Work Definition* that represents a set of *Work Definitions* focusing on a portion of the *Process* that results in a release (internal or external).

A *Process Performer* is a *Model Element* describing the owner of *Work Definitions*. *Process Performer* is used for work definitions that cannot be associated with individual *Process Roles*, such as a *Lifecycle* or a *Phase*. A *Process Role* describes the responsibilities and competencies of an individual carrying out *Activities* within a *Process*, and responsible for certain *Work Products*.

Process packages allow any arbitrary (and overlapping) groupings of process Definition Elements. A Process Component is a package that has some internal consistency, and that is used for structuring a large Process. A Process is a complete description of a software engineering process, in term of Process Performers, Process Roles, Work Definitions, Work Products, and associated *Guidance*. A *Discipline* is a *Process Package* organized from the perspective of one of the software engineering disciplines: configuration management, analysis and design, test, and so forth.

Being a UML Profile, the SPEM benefits of UML diagrams to present different perspectives of a software process model: in particular, class diagram, package diagram, activity diagram and use case diagram. The SPEM notation suggests alternate representations for most frequently used concrete classes of the meta-model: these icons are depicted in Figure 1-3.

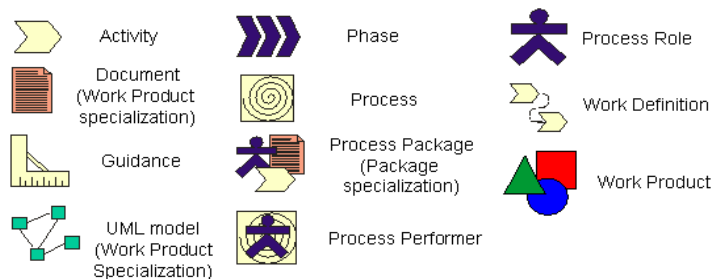


Figure 1-3. SPEM icons.

SPEM standard aims at accommodating a large range of existing and described software development processes, and not excluding them by having too many features or constraints [11].

4. OSSDP MODELING

4.1 The definitional level

Our first highly abstract model is shown in Figure 1-4. The whole OSSDP is described as a single SPEM *Process Package* with two *Disciplines*: ‘Software development processes’ and ‘Community processes’. This prescriptive model indicates that a fully-fledged OSSDP *requires the implication of a wide and organized community of distributed volunteer contributors*.

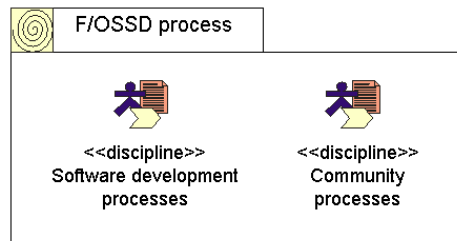


Figure 1-4. The definitional model.

Most OSS projects are actually designed and developed by individuals, not communities: 57% have one or two developers [22] (34% according to [23]), and only 15% of them have more than 10 developers [22] (19% according to [23]). In the first category, these very small OSS projects are directed by a single ‘lead developer’ – usually the software’s original author – who assumes all the responsibilities and interacts with a small community of end users.

Our model focuses on the latter category, roughly corresponding to the ‘team work, external patches’ scenario of [20], and which includes the most successful OSS projects.

For some authors, small OSS projects are projects still in infancy, and large projects are mature ones [20]. For instance, Stephano Mazzocchi’s ‘Stellar Model’ [24] compares these stages and lifecycles to the ones of stars and gravitational systems in general: expansion, fragmentation, contraction. In our model, we are describing mature OSS projects.

4.2 The generic level

Our second level defines a generic model of OSSDP, resulting from a synthesis of many studies that report on OSS projects, and a survey of a

number of OSS Web portals. It is divided in two parts: the global view and the use case view.

4.2.1 The global view

Each *Discipline* of the definitional level is first described as a set of *Model Elements: Process Roles* with the *Work Definitions* they perform (*Activities* or complex *Work Definitions*), *Work Products (Documents)* and *Guidance* entities mainly describing *tool usage* (see Figures 1-5 and 1-6).

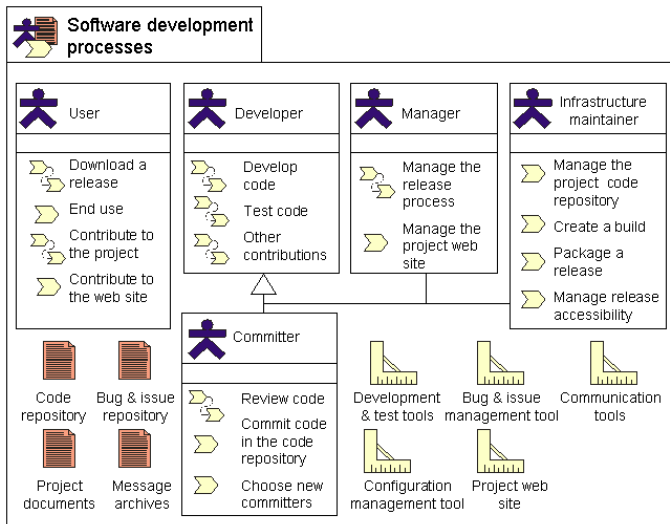


Figure 1-5. The Software Development Discipline.

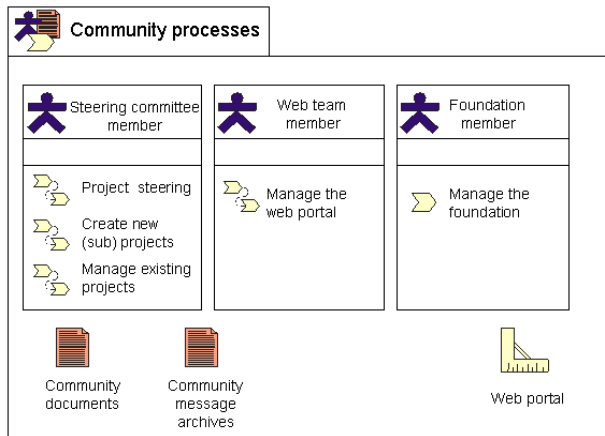


Figure 1-6. The Community Development Discipline.

Process Roles reflect the different levels of participation which exist in all OSS projects. They “simply reflect a natural gradient of interest, competence and commitment” (E. Raymond, cited in [8]).

Our model can abstract different concrete development organizations. For instance, in projects with a closed ‘inner circle’, all Developers are also Committers who are granted write access rights to the project code repository. In some projects, the Steering Committee is an elected governing board (e.g. Apache Group), while other projects have a single project owner (or ‘benevolent-dictator’ [25]). When the number of participants grows, secondary leaders emerge. Rather than project owners, they normally act as managers (e.g., release managers) or maintainers (e.g., module or infrastructure maintainers). More generally, leaders are also Developers which is a radical difference from traditional development models. Most projects operate as meritocracies [8]: the more someone participates, the more merit or trust they earn from their peers, and the more they are allowed to do. When the size of the software becomes too large, new functionalities are added by means of ancillary (sub) projects. By this way, development teams are kept small so that coordination can be handled by simple and often implicit mechanisms.

Our model also emphasizes that tool mediation is the norm for OSS projects: all policies such as authentication or regulation of commit privileges are enforced by the tools on the project server. Most tools are open source software and similar across projects, lowering the entry barrier for participation. CVS and Subversion (version management), Bugzilla and GNATS (bug and issue tracking), Hypermail (mailbox to HTML transformer) are some examples of popular tools in open source communities. Communication is predominantly asynchronous through private mail and public or semi public mailing lists.

4.2.2 The use case view

The complex *Work Definitions* of both *Disciplines* are refined with SPEM use case diagrams which in turn recursively define *Activities* or simpler *Work Definitions*. Use cases allow to specify that several *Process Roles* are collaborating within a complex *Work Definition*. These graphical descriptions are easy to understand and can be further clarified with SPEM *External Descriptions*.

Figure 1-7 shows the User-oriented use case diagrams refining ‘Download a release’ and ‘Contribute to the project’ *Work Definitions*. These diagrams emphasize the fact that the most important participants in OSSDP are the people who use the software. Users contribute to the project by providing feedback to developers in the form of bug reports and feature

suggestions, and by participating in various issue discussions. Peer review is implicit in OSS projects: source code is available to every user, and technical communication is conducted in public. Most developers start out as users and therefore guide their development efforts from the user's perspective.

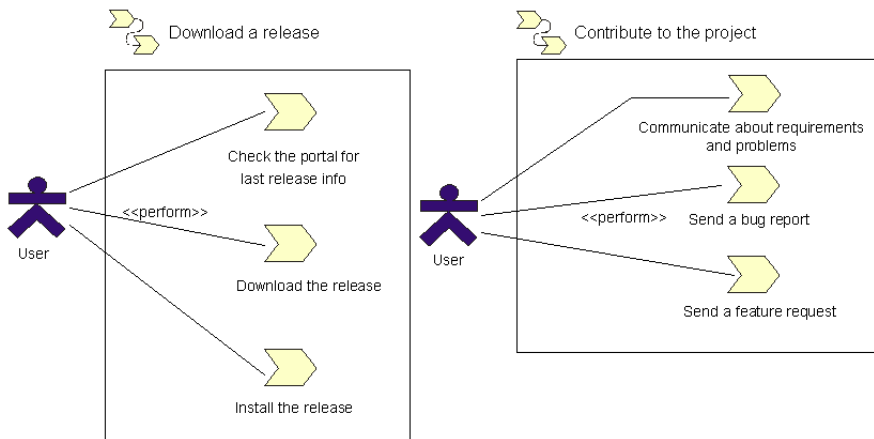


Figure 1-7. The User-oriented use cases.

In general, OSS projects have a small, elite team of capable developers, all of whom are granted write access to the source code repository ('Committers'). This core group creates the vast majority of new functionality. A much larger group mainly provides bug fixes ('Developers'). Small increments (bug fix or enhancement) and rapid iteration typify all OSS projects. Figure 1-8 shows the Developer and Committer-oriented use case diagrams refining 'Develop code', 'Test code', 'Review code', and 'Other contributions' *Work Definitions*.

As we can see in Figures 1-7 and 1-8, there is no formal requirements process: requirements are determined implicitly, as whatever the developers actually build. Since developers are also end users and domain experts, they should understand the requirements in a deep way.

Design activities are also missing from Figure 1-8. Design takes place at the very beginning of the project when an early version of the product is produced by an individual or a small closed group (see the 'Provide Initial Code' *Activity* in Figure 1-10). This early version is sometimes built from scratch, or more often, it reuses and extends an existing product. For instance, Apache was based on the NSCA HTTPD server and Mozilla was derived from the Netscape Communicator code base.

In OSS projects, there are no distinct phases: participants work concurrently on whatever task (code, test, discuss, etc.) they find interesting.

A good modular design of the product under construction makes such division of labor and parallelism easier.

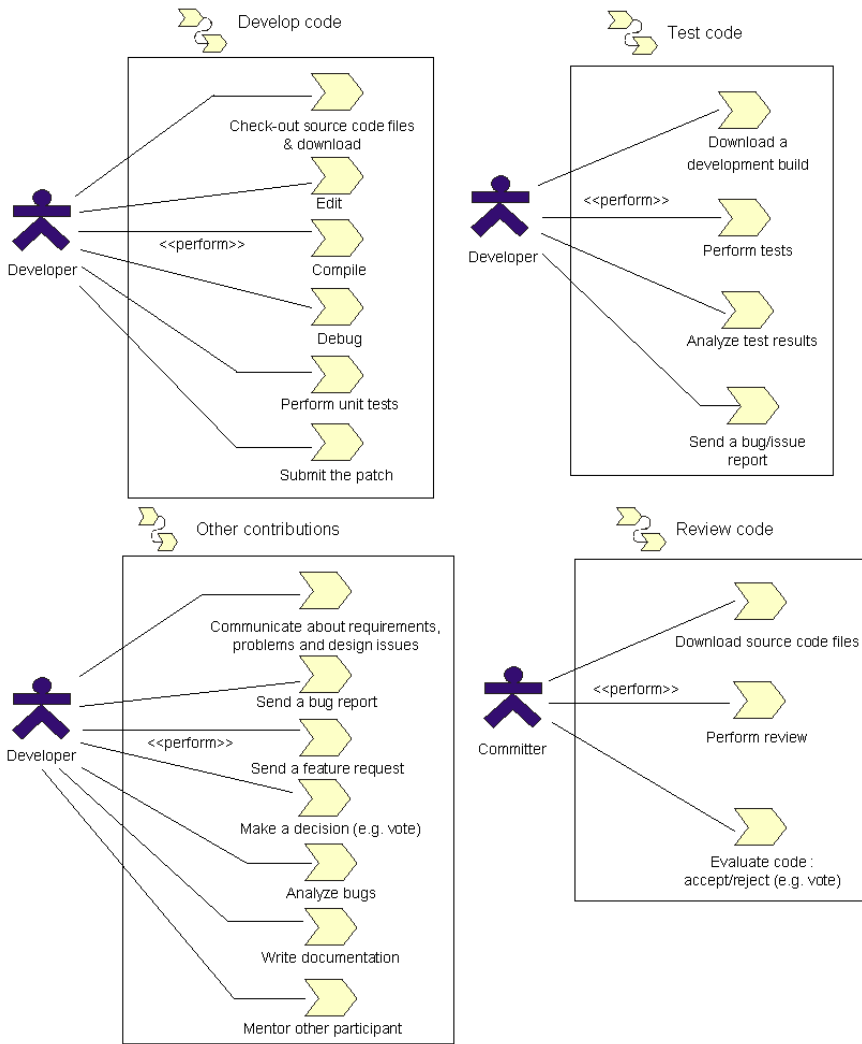


Figure 1-8. The Developer and Committer-oriented use cases.

To synchronize change, at some point, all important changes are merged into a new release. Developers must contribute their finalized code for the new release. Release cycles overlap, with release $i+1$ development starting in parallel with release i reviewing and debugging. At the highest level of parallelism, some OSS projects also maintain parallel code branches: one for ongoing development and the other for stability and widespread use. Linux

is a well known example of this approach, where the middle number of the version identifier characterizes the release: odd numbers are for development kernels and even numbers for stable ones. Figure 1-9 depicts the Manager-oriented use cases related to the release management process.

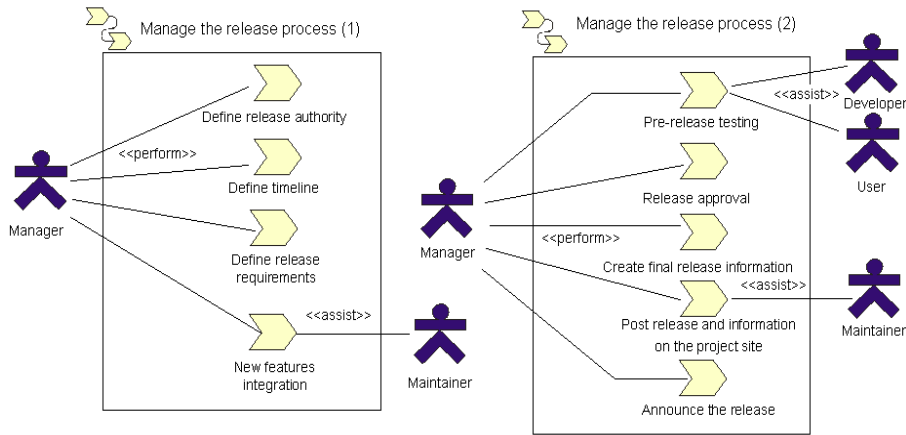


Figure 1-9. The Manager-oriented use cases.

Most activities at the community level (see Figure 1-10) are oriented for making easier participation and communication, such as Web portal management.

Open source development is much more informal than usual software engineering projects: there are typically no plans or schedules. Some projects have a brief vision summary and a development roadmap, produced by the Steering Committee Members (see Figure 1-10) and describing for instance the milestone schedule for the next year (in the Mozilla project). But, as participants are volunteers there is no real commitment to deliver something within a fixed timeframe.

4.2.3 Discussion

Our prescriptive model is consistent with Scacchi’s generic OSSDP model [26] (Figure 1-11) or Gilliam’s model [27] (Figure 1-12). It has a larger scope and encompasses all the basic aspects of software development, in terms of Roles, Tools, Documents, and Activities. In a different way, Scacchi and Gilliam models emphasize the cyclic nature of the overall process and the central role of experience sharing. Our model can also be compared with textual descriptive process models or frameworks, such as [8] and [28]. Unlike all these informal descriptions, our model can be refined until it provides a precise and description of specific existing OSSDPs,

amenable to systematic analysis, comparison, and re-use. This point will be discussed and exemplified in the next sections.

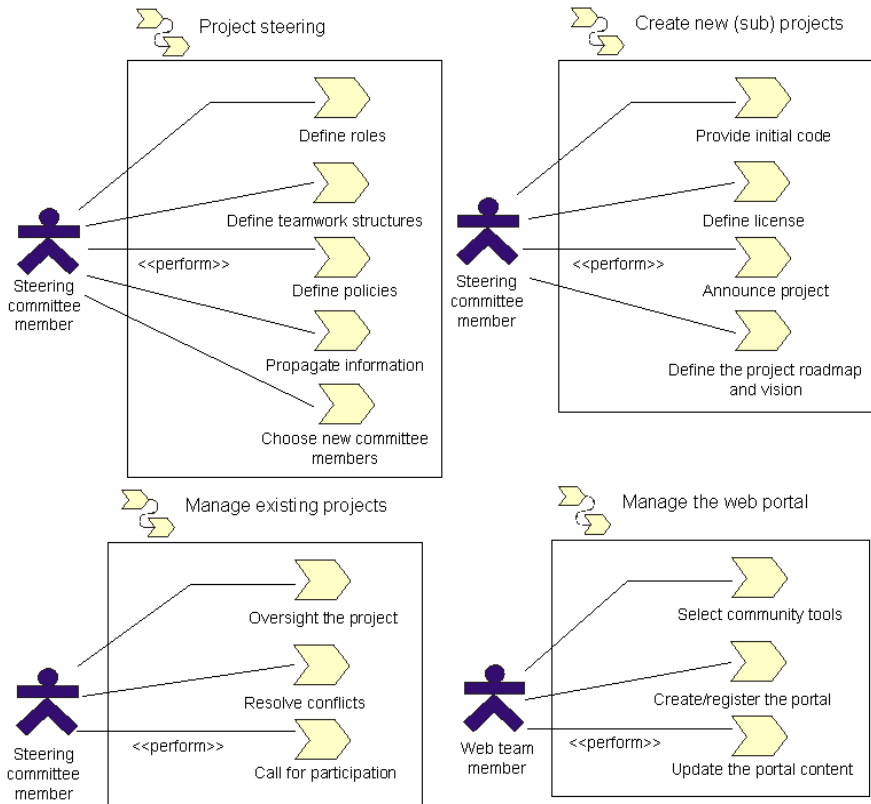


Figure 1-10. Community-oriented use cases.

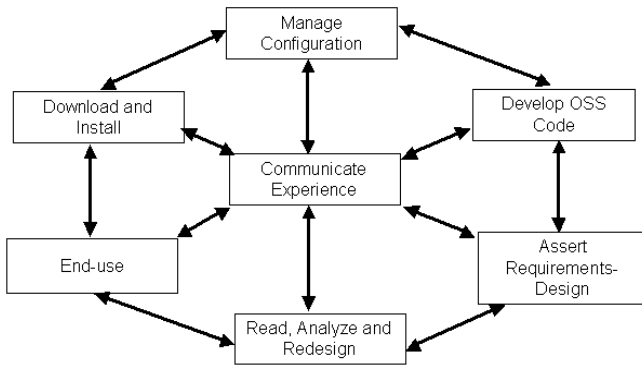


Figure 1-11. Scacchi's generic OSSDP model.

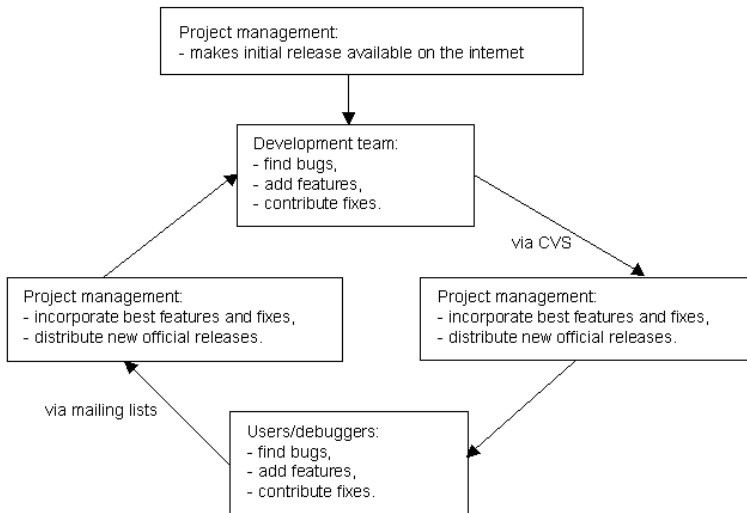


Figure 1-12. Gilliam's model of OSSDP.

Eric Raymond principles ([29]) stay at a more abstract level, but many of them have a direct counterpart in terms of the process model:

- every good work of software starts by scratching a developer's personal itch,
- good programmers know what to write; great ones know what to rewrite (and reuse),
- if you have the right attitude (i.e. code sharing), interesting problems will find you,
- treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging,
- release early; release often; and listen to yours customers,
- given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone,
- if you treat your beta-testers as if they are your most valuable resource, they will respond by becoming your most valuable resource,
- the next best thing to having good ideas is recognizing good ideas from your users; sometimes the latter is better,
- perfection (in design) is achieved not when there is nothing to add, but rather when there is nothing more to take away,
- to solve an interesting problem, start by finding a problem that is interesting to you,
- provided the development coordinator has a medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.

4.3 The specific level

At this level, our model describes the specific features discovered by analyzing the project shared information spaces on the Web. Each OSS project defines its own *Process Roles*, *Documents*, *Guidance*, and more or less detailed procedural behaviors that we translate into SPEM activity diagrams. In the next two subsections we exemplify the approach with the release management process of the NetBeans IDE project and the Apache HTTP Server project. We have chosen these projects because of several reasons: their rich information space, the availability of many studies that report on them ([30, 4, 10, 31, 32]), their release management process, since it reflects much of the underlying philosophy of OSS projects [33].

4.3.1 The NetBeans IDE release management process

In a first step, we specialize the concepts of the generic process model. Figure 1-13 exemplifies some specializations of the generic *Process Roles* and *Documents* entities.

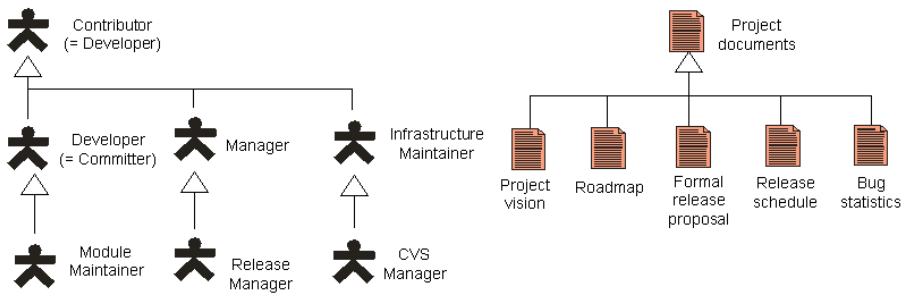


Figure 1-13. Generic concept specializations.

In NetBeans, Contributors do not have write-access to the source tree managed by CVS (Concurrent Versions System). Committers are called Developers, and have CVS write-access for some individual modules. Each module has one Module Maintainer who has check-in permissions (for that module or global), and who manages a group of Developers. All Managers and Maintainers are also Contributors. A number of project *Documents* play a central role for coordinating the participants during the release process.

The informal description of the release process found on the Web portal is a mail posted by the current Release Manager to the developers mailing list (nbdev). The description of the process may slightly evolve from one release to another. We give below an excerpt of two successive versions of that informal description (Figures 1-14 et 1-15).

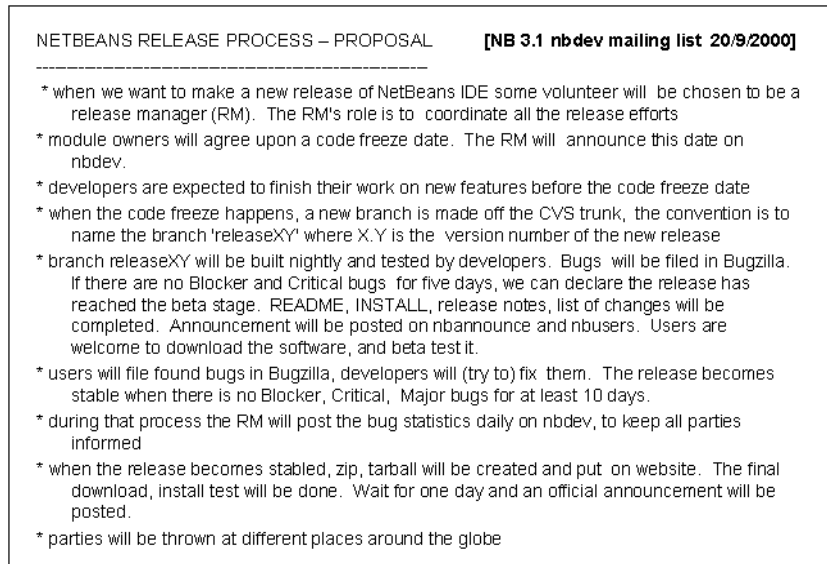


Figure 1-14. NetBeans release process description for version 3.1.

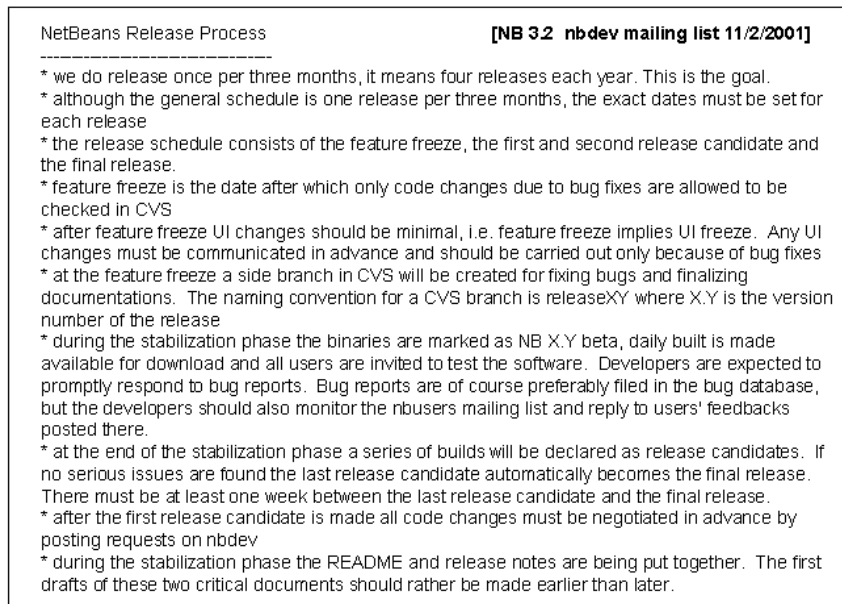


Figure 1-15. NetBeans release process description for version 3.2.

On the basis of these textual descriptions, we have devised a multi layered SPEM activity diagram. At the first level, Figure 1-16 specifies the sequence of all *Activities* of the generic use case model that was depicted by

Figure 1-9. Some *Activities* are then decomposed into smaller *Steps*. ‘Define release requirements’, ‘Define release authority’, ‘Define timeline’, and ‘Pre release testing’ *Activities* are taken below as examples of such refinements (see Figures 1-18, 1-20, 1-22, 1-23). For each refined *Step*, we give excerpts of documents, mainly emails from the developers mailing list, as their rationale or illustration. At this level, we are starting to devise process model fragments from process enactment instances discovered in the project information space, instead of formalizing process descriptions written by core participants as previously. The reliability of these model fragments is more questionable but can be strengthened by the frequency of the observed pattern in the project history. Some researchers propose to automatically extract these model fragments from the artifacts (source code files, messages in public discussion forums, Web pages), the artifact update events (version release announcements, Web page updates, message postings), and the work contexts (roadmap for software version releases, Web site architecture, communications systems in use) [34, 35].

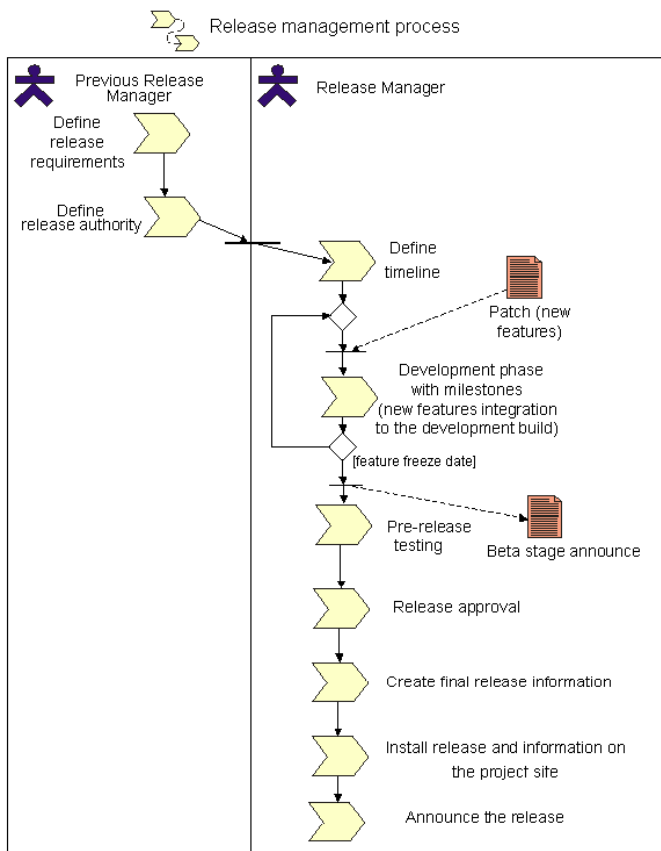


Figure 1-16. NetBeans release management process.

Figure 1-17 shows a release proposal (a mail from nbdev mailing list) and Figure 1-18 explains how it has been constructed.

```

Subject: [nbdev] NB 3.4                                [NB 3.4 nbdev mailing list 13/1/2002]
NetBeans 3.4 Draft Release Proposal
Planned FCS: August 2002
High level theme:
  Usability, productivity, and runtime performance
Major proposed content:
  Improve User Interface and Usability.
  The primary focus being improved workflow and better integration of existing functionality.
  Full support for J2SE 1.4
  This includes both running on 1.4 as the primary JDK, as well as support for new features in
  1.4 (e.g. asserts, new swing components)
  Dependency Manager as part of the new Projects infrastructure.
  Provide base level support of (Projects) and MDR as needed to support the other goals (e.g.
  the Dependency Manager)
If you would like to make a contribution to NB 3.4 or you have suggestions particularly ones that fit
the themes mentioned above, please let everyone know.
Evan
    
```

Figure 1-17. A release proposal part of ‘Define release requirements’ activity.

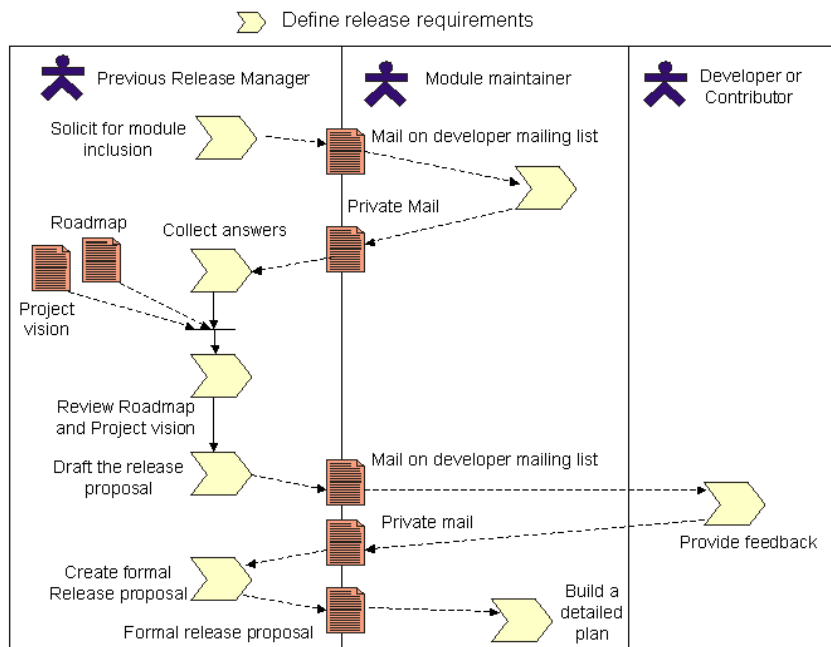


Figure 1-18. ‘Define release requirements’ refined activity diagram.

The ‘Define release authority’ Activity includes a public call for the designation of a new Release Manager (see the mails in Figure 1-19).

Subject: WANTED: Release Manager for NetBeans IDE 3.2 [NB3.2. nbdev mailing list 20/2/2001]

Hi everybody,

March 9, 2001, the feature freeze day for NB 3.2 is coming. We need someone to act as the release manager for this release. Anyone interested in contributing his/her time, energy and nerves ?

The responsibilities of the release manager are summed up in my recent post on nbdev.

----- [NB3.2. nbdev mailing list 22/2/2001]

Hi,

It seems that nobody is going to raise the hand. So, I'm going to do so. However if someone would like to change his mind and be the release manager I wouldn't mind.

----- [NB3.2. nbdev mailing list 23/2/2001]

Petr, Jesse, it's great that you volunteer. I am completely happy with Petr being the NB 3.2 release manager and Jesse dealing with CVS things. Community, I think we can close this soon. If anybody wants to object the election of Petr Hrebejk to be the NB 3.2 release manager, please post your opinions now. Otherwise on Monday I would consider this done and announce this fact to the world.

----- [NB3.2. nbdev mailing list 26/2/2001]

Subject: ANN: Petr Hrebejk is the release manager for NetBeans IDE 3.2

NetBeans developers have elected Petr Hrebejk to be the release manager for the upcoming release of the IDE. Jesse Glick is volunteering to help Petr with CVS technicalities.

Figure 1-19. Mails related to the 'Define release authority' activity.

The process for defining the release authority can slightly change from one iteration to the next. Figure 1-20 defines a kind of 'standard practice' with candidacy announcements and consensus establishment.

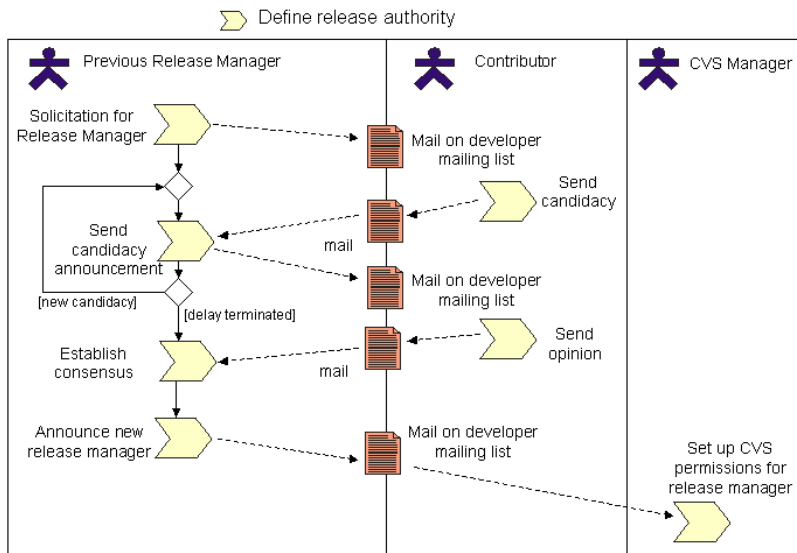


Figure 1-20. 'Define release authority' refined activity diagram.

Projects regularly enforce feature freeze (and/or code freeze). During NetBeans feature freeze, no new functionality can be added to the code base, however bug fixes are permitted. The 'Define timeline' Activity (Figure 1-22) produces a release schedule specifying all milestones (Figure 1-21).

Subject: [nbdev] Proposed 3.4 Schedule [NB3.4. nbdev mailing list 5/3/2002]

Here's a proposal for the 3.4 Schedule.
 During the development phase we will continue with weekly Q-builds process.
 We propose to have milestones every three weeks so we can check progress and catch potential problems earlier. The QA group has volunteered to do extra testing of the milestone builds. The last milestone is the feature freeze -- all features must be integrated by this date.
 We propose to enter High Resistance mode two weeks before the first release candidate.
 We hope that this process will ensure that RC1 will be a true Release Candidate. Additional release candidates will be produced as needed.

Development Phase
 Apr 03 Milestone 1
 Apr 24 Milestone 2
 May 15 Milestone 3, Feature Freeze

Beta
 Jun 05 Beta 1
 Jun 26 Beta 2
 Jul 07 Enter High Resistance

Release Candidate(s)
 Jul 24 RC 1
 <Additional RCs as needed>

Aug 21 FCS

A milestone has been met when all of its tasks have been implemented and tested, accessibility and I18N issues have been completed, and unit tests have been written.

Comments?
 Evan

Figure 1-21. A release schedule proposal part of 'Define timeline' activity.

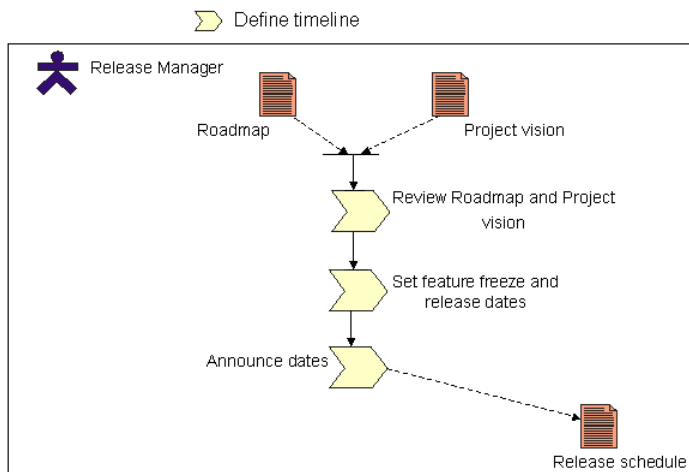


Figure 1-22. 'Define timeline' refined activity diagram.

The 'Pre-release testing' Step of Figure 1-16, which constitutes the core of the release process, is refined twice. The first refinement (Figure 1-23) shows the 'stabilization phase' followed by a sequence of release candidates (RC). The 'high resistance mode' ensures that lead programmers review code changes. Every RC build is created when there is no known critical bug. If any critical bug is discovered within one week after RC is built, the bug has to be fixed and a new RC is created. If no critical bug is discovered within one week in RC, this build will become the final (stable) release.

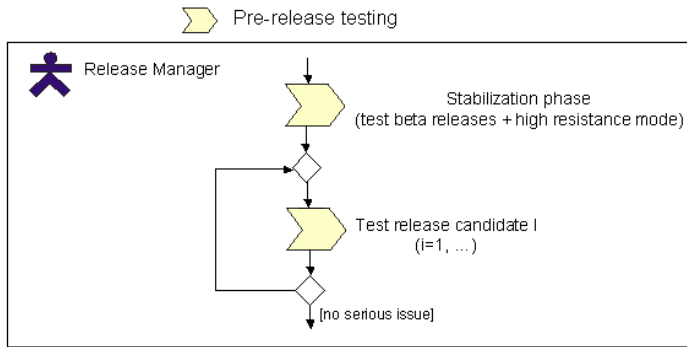


Figure 1-23. 'Pre-release testing' first refinement.

The second refinement (Figure 1-24) gives more details about the 'Stabilization phase' during which Developers and Contributors propose bug fixes to the beta release which is daily built.

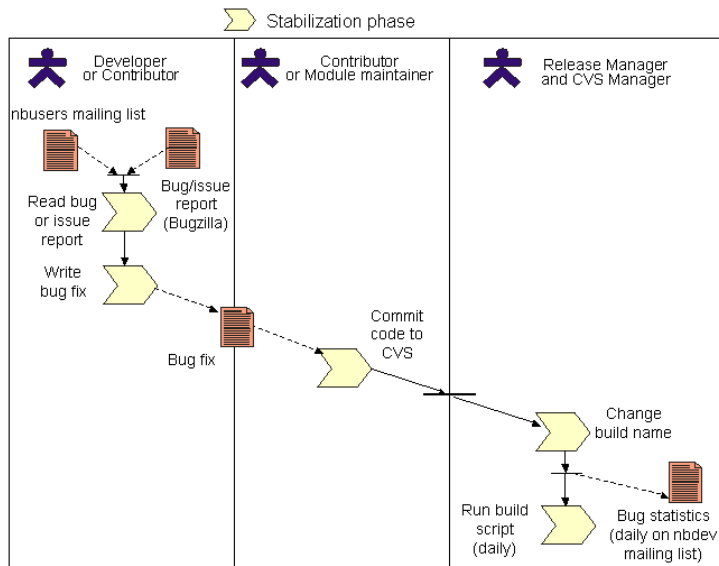


Figure 1-24. 'Pre-release testing' second refinement.

In addition, several *Activities* implemented by Sun's Quality Assurance team, responsible for the commercial product SunONE Studio which extends NetBeans, come with the open source release process such as weekly Q-builds, for ensuring an assured level of quality, and extra testing of the milestone builds during the stabilization phase. This close relationship with a 'commercial process' has no deep impact on NetBeans OSSDP.

4.4 The Apache Server release management process

On the Apache Server Web portal a document informally describes the release process (see Figure 1-25).

This document describes the general release policies used by the Apache HTTP Server Project. As described herein, this policy is not set in stone and may be adjusted by the Release Manager.

Who can make a release? Technically, any one can make a release of the source code due to the Apache Software License. However, only members of the Apache HTTP Server Project (committers) to project can make a release designated with Apache. Other people must call their release something other than "Apache" unless they obtain written permission from the Apache Software Foundation.

Who is in charge of a release? The release is coordinated by the Release Manager (hereafter, abbreviated as RM). Since this job requires coordination of the development community (and access to CVS), only committers to the project can be RM. However, there is no set RM. Any committer may perform a release at any time. In order to facilitate communication, it is deemed nice to alert the community with your planned release schedule before executing the release.

Who may make a good candidate for RM? Someone with lots of time to kill. Being an RM is a very important job in our community because it takes a fair amount of time to produce a stable release.

When do I know if it is a good time to release? It is our convention to indicate showstoppers in the STATUS file in the repository. A showstopper entry does not automatically imply that a release can not be made. As the RM has final authority on what makes it into a release, they can choose to ignore the entries. An item being denoted as a showstopper indicates that the group has come to a consensus that no further releases can be made until the entry is resolved. These items may be bugs, outstanding vetos that have not yet been resolved, or enhancements that must make it into the release.

What power does the RM yield? Regarding what makes it into a release, the RM is the unquestioned authority. No one can contest what makes it into the release.

How can an RM be confident in a release? The RM may perform sanity checks on release candidates. One highly recommended suggestion is to run the httpd-test suite against the candidate. The release candidate should pass all of the relevant tests before making it official. Another good idea is to coordinate running a candidate on apache.org for a period of time. This will require coordination with the current maintainers of apache.org's httpd instance. In the past, the group has liked to see approximately 48-72 hours of usage in production to certify that the release is functional in the real world.

What can I call this release? At this point, the release is an alpha. The Apache HTTP Server Project has three classifications for its releases: Alpha Beta General Availability (GA). Alpha indicates that the release is not meant for mainstream usage or may have serious problems that prohibit its use. When a release is initially created, it automatically becomes alpha quality. Beta indicates that at least three committers have voted positively for beta status and there were more positive than negative votes for beta designation. This indicates that it is expected to compile and perform basic tasks. However, there may be problems with this release that prohibit its widespread adoption. General Availability (GA) indicates that at least three committers have voted positively for GA status and that there were more positive than negative votes for GA designation. This release is recommended for production usage.

Who can vote? Non-committers may cast a vote for a release's quality. In fact, this is extremely encouraged as it provides much-needed feedback to the community about the release's quality. However, only binding votes casted by committers count towards the designation. Note that no one may veto a release.

How do we make it public? Once the release has reached the highest-available designation (as deemed by the RM), the release can be moved to the httpd distribution directory on apache.org. Approximately 24 to 48 hours after the files have been moved, a public announcement can be made. We wait this period so that the mirrors can receive the new release before the announcement. An email can then be sent to the announcements lists (announce@apache.org, announce@httpd.apache.org). Drafts of the announcement are usually posted on the development list before sending the announcement to let the community clarify any issues that we feel should be addressed in the announcement.

Should the announcement wait for binaries? In short, no. The only files that are required for a public release are the source tarballs (.tar.Z, .tar.gz). Volunteers can provide the Win32 source distribution and binaries, and other esoteric binaries.

Figure 1-25. Informal release process description.

Some examples of specialization of the generic *Process Roles* are shown in Figure 1-26. All Apache projects (HTTP Server, Jakarta, XML, etc.) are managed using a collaborative, consensus-based process. Apache is a meritocracy where the rights and responsibilities follow from the skills and contributions of participants. The Project Management Committee (PMC) is a group of Committers who take responsibility for the long-term direction of the projects in their area. Members of the PMC are self-selected Committers. There is a single PMC for each parent project which is commissioned directly by the Apache Software Foundation Board of Directors. The Board of Directors ultimately has the final decision making power on any project. They delegate this responsibility to the PMC of each project. Although the Release Manager has the ultimate say in what goes into the final release, the PMC can make suggestions. The PMC is in turn responsible for many sub-projects, each of which with its own group of Committers.

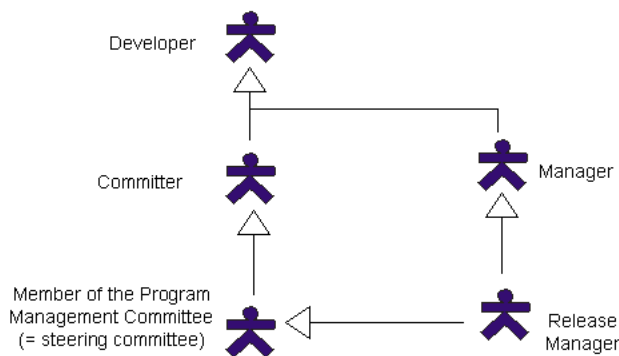


Figure 1-26. Some Apache Server Process Roles.

The high level activity diagram that can be devised from the informal process description is similar to the diagram drawn for the NetBeans process (see Figure 1-16). Differences stay at a more detailed level. For instance, the ‘Define Release authority’ *Activity* just includes the self designation from a Member of the PMC who accepts to act as the Release Manager, instead of the public call for candidates in the NetBeans process. Other specific *Activities* are described with more details below.

It should be noted that all important information about the release (its definition, timeline, status, changes, expected new features, and so on) is recorded within the repository STATUS file (see Figure 1-27). The STATUS file defines in particular ‘showstoppers’, which are issues that require a fix before the next release. They are defined by ‘lazy consensus’: a showstopper is valid if no Committer disputes the issue by sending a negative vote or a veto vote.

Each of the Apache Project's active source code repositories contain a file called "STATUS" which is used to keep track of the agenda and plans for work within that repository. The active STATUS files are automatically posted to the mailing list each week.

Many issues will be encountered by the project, each resulting in zero or more proposed action items. Action items must be raised on the mailing list and added to the relevant STATUS file. All action items may be voted on, but not all of them will require a formal vote. Types of Action Items :

- long Term Plans : are simply announcements that group members are working on particular issues related to the Apache software. These are not voted on.
- short Term Plans : are announcements that a developer is working on a particular set of documentation or code files, with the implication that other developers should avoid them or try to coordinate their changes. This is a good way to proactively avoid conflict and possible duplication of work.
- release Plan : is used to keep all the developers aware of when a release is desired, who will be the release manager, when the repository will be frozen in order to create the release, and assorted other trivia to keep us from tripping over ourselves during the final moments. Lazy majority decides each issue in the release plan.
- release Testing : after a new release is built, colloquially termed a tarball, it must be tested before being released to the public. Majority approval is required before the tarball can be publically released.
- showstoppers : are issues that require a fix be in place before the next public release. They are listed in the STATUS file in order to focus special attention on the problem. An issue becomes a showstopper when it is listed as such in STATUS and remains so by lazy consensus.
- product Changes : changes to the Apache products, including code and documentation, will appear as action items under several categories corresponding to the change status:
 - conceptplan : an idea or plan for a change. These are usually only listed in STATUS when the change is substantial, significantly impacts the API, or is likely to be controversial. Votes are being requested early so as to uncover conflicts before too much work is done.
 - proposed patch : a specific set of changes to the current product in the form of input to the patch command (a diff output).
 - committed change : a one-line summary of a change that has been committed to the repository since the last public release.

Figure 1-27. Description of the STATUS file.

This organization makes the 'Define Release requirements' and 'Define timeline' process fragments quite simple with a simple update of the STATUS file (see Figure 1-28).

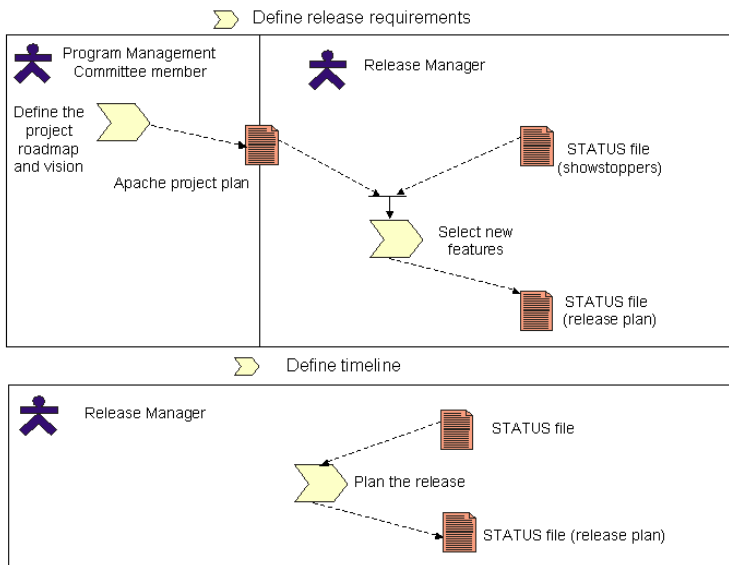


Figure 1-28. 'Define release requirements' and 'Define timeline' refinements.

‘Pre release testing’, which constitutes the core of the release process, is different from its NetBeans counterpart due to the democratic and distributed style of management of the Apache project, and to different quality insurance procedures.

First, transitions from the alpha stage (i.e., a release which may have serious problems that prohibits its use) to the beta stage (i.e., a release expected to compile and to perform basic tasks) and from the beta stage to the final stage (i.e., a release recommended for production usage) are collective decisions taken after a vote (see Figure 1-29) with a majority consensus rule: at least three Committers should have voted positively for the new status and the number of positive votes for that designation should exceed the number of negative votes.

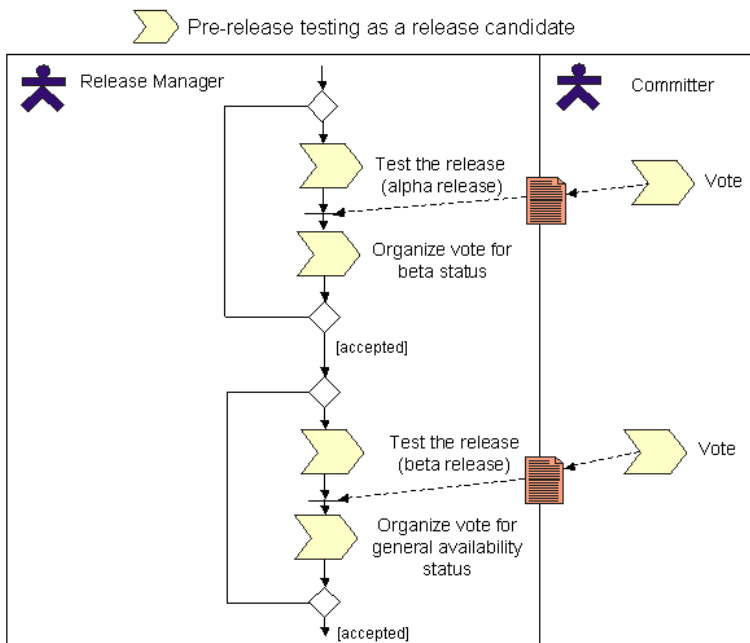


Figure 1-29. Pre-release testing.

Secondly, to ensure a high level of quality, different prescriptions should be satisfied:

- the regression test suite should be run against the release candidate; Figure 1-30 shows that new test cases are expected from Committers each time they fix a bug,
- each release candidate should be used in production (i.e., for running the main apache.org Web server) for a given period of two or three days (see Figure 1-31).

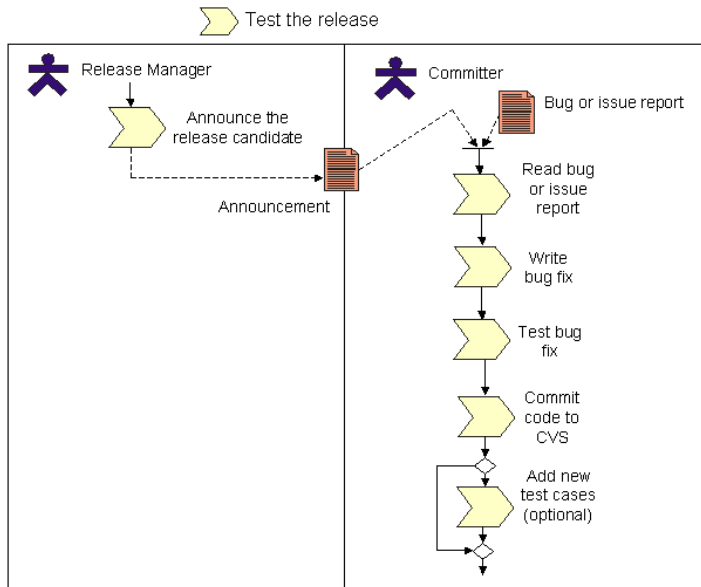


Figure 1-30. First refinement of pre-release testing.

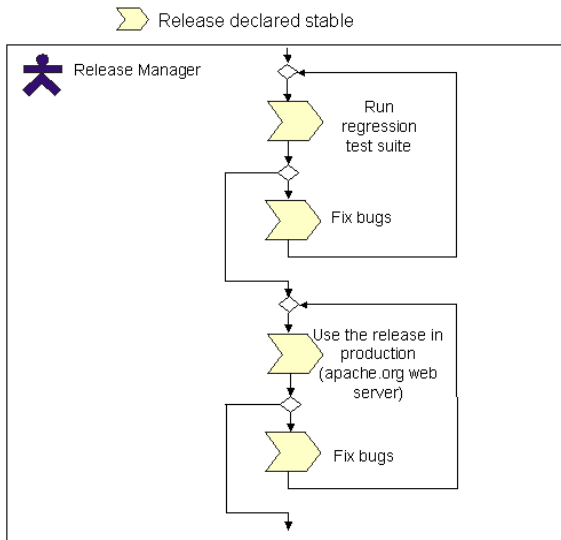


Figure 1-31. Second refinement of pre-release testing.

According to the informal process description (see Figure 1-25), these policies “are not set in stone and may be adjusted by the Release Manager” under the circumstances. It is worth noting that no process-oriented proposals or discussions can be found within recent Apache Server mailing

list archives while many of them can be found within NetBeans archives: debates about the Board election process, the Q-build process, the criteria and process a module has to pass to be marked as stable, and so on. Surprisingly, NetBeans process is more a collective construction than Apache process, while NetBeans is a project supported by an industrial software company. The reason could be the level of maturity of the process, higher in Apache than in NetBeans. Another study of the Apache project reports many process discussions during its early stages (in 1995) about the vote and patch system, show stopper bugs and code freeze [10].

5. DISCUSSION

Three main goals and benefits can be attached to the modeling of software processes [36], [37]: (1) process understanding and communication, (2) process comparison, reuse, and improvement, (3) process enactment support. We discuss these three aspects in the case of OSS development modeling in general, and in the case of using SPEM in particular.

5.1 Process understanding and communication

OSSDPs can be described as a network of (largely social) processes arranged in a highly dynamic topology [38]. Besides the release process that we have studied in depth in the previous section, other process fragments include testing, work coordination, critiquing, suggesting, tool-building, bug triage, negotiation, evaluation, etc.

Generally, models of specific process fragments with ad hoc implicit notations, such as the ‘life cycle of changes’ in FreeBSD project [39] (Figure 1-32), stay at a very abstract level. A detailed process fragment description should at least specify the main relationships between people (roles), products and activities.

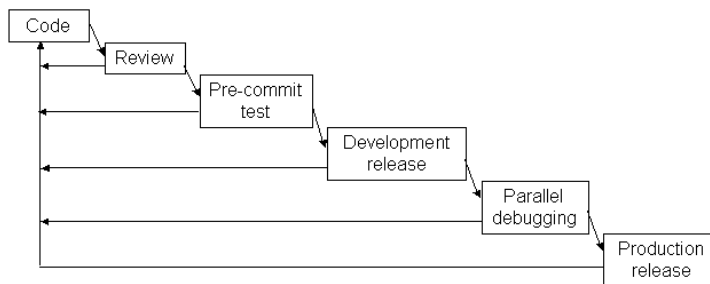


Figure 1-32. The life cycle of changes in FreeBSD project.

It has been partly done for different process fragments of the FreeBSD project in a following paper [40] with a more precise notation including roles and decision points (Figure 1-33).

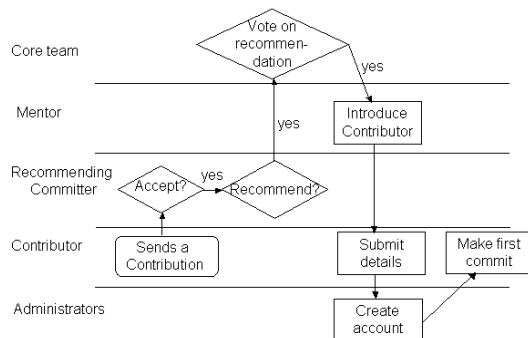


Figure 1-33. ‘Adding a new Committer’ process model fragment in FreeBSD project.

Many software process modeling formalisms are designed for describing the relatively ‘linear with feedback loops’ structure of classical software development processes. It is the case, for instance, of Petri net based formalisms [41, 42]. At the opposite, SPEM allows to describe different perspectives of a software process model through all basic UML diagrams. In this chapter we used nested package diagrams for defining the main *Model Elements*, use case diagrams for showing the relationship between *Process Roles* and the main *Work Definitions*, and activity diagrams for presenting the sequencing of *Activities* with their input and output *Work Products*. We could also use Class diagrams for representing the structure, decomposition, and dependencies of *Work Products*, and Statechart diagrams for specifying the behavior of SPEM *Model Elements*, and therefore all the remaining concepts of the SPEM meta model. For OSS communities, which are basically communities of developers, UML diagrams should be easier to understand and accept than any other process modeling formalism and a valuable alternative to informal textual descriptions. This could be the first step for promoting the use of software process models in the OSS community for process understanding and communication.

The main weakness of SPEM is its approach for modeling tools. Unlike many other software process meta models (e.g. [43, 44]), tools are not first class concepts. We have represented them within *Disciplines* by using the more general *Guidance* concept, because one possible kind of guidance is tool usage specification (called ‘*Tool Mentor*’). This solution is not fully satisfactory: no specific notation exists for specifying the relationships between tools and *Work Definitions*, *Work Products* and *Process Roles*. It

would be interesting to have more precise and systematic notations for specifying how tools mediate the development process because it constitutes a fundamental characteristics of all OSSDPs (see section 2.1).

5.2 Process comparison, reuse and improvement

OSSDPs mainly differ in their decision-making style, their coordination style, and their quality insurance procedures. All these aspects can be precisely documented through process modeling techniques, and the SPEM approach in particular.

For instance, Apache adopts an approach to coordination well suited to small projects. The server itself is kept small (77 kSLOC). Any functionality beyond the basic server is added by means of ancillary projects that interact with Apache only through Apache's well-defined interface. The coordination is successfully handled by a small core team (10-15 persons) using primarily implicit mechanisms: a knowledge of who has expertise in what area, general communication about what is going on, and who is doing what and when. There is no waiting for approvals, permission, and so forth. This highly implicit coordination style is exemplified by Figure 1.30 concerning bug fixing during the release process. The larger NetBeans project (758 kSLOC) includes more formal means of coordinating the work, such as module owners (Module Maintainer *Process Role*) who approve and perform changes to the modules. This more disciplined coordination style is exemplified by Figure 1.24 for the same bug fixing activity.

By codifying OSSDPs as formal models, the OSS community could share their 'best practices' as open source software process models [7]. Empirically, a process is good because people freely accept to follow it: participants 'are voting with their feet' [10]. New projects could start with such 'approved procedures' instead of reinventing everything through trial and errors. The high modularity of the SPEM approach could favor reuse of model fragments corresponding to loosely articulated sub process and therefore incremental process reuse and improvement.

Our multi level approach allows to analyze if a given project complies with our generic level and at which level of details it differs from other projects. The use of a well defined meta model with a sufficient expressive power is important. For instance, it is not easy to compare the release engineering process model fragment of FreeBSD as depicted by Figure 1-34 [40] with the corresponding model fragments from the Apache and NetBeans projects because many questions have no answer: who build the release schedule? when? how? which activities can take place after the feature freeze and after the code freeze? how the release is stabilized? who is in charge of the deployment? which tools are used?

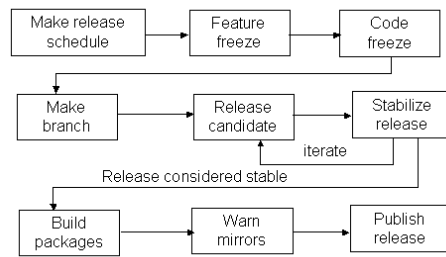


Figure 1-34. FreeBSD release engineering process model fragment

5.3 Process enactment support

Software process performers (developers, managers) can receive indirect support through guidance information, which helps them to perform their work, such as determining the current status of the process, the appropriate next steps to be executed, the decision points and their meanings, etc. Guidance is provided through manual or mechanical interpretation of software process models simultaneously and synchronously with the actual process performance [36]. Software process performers can also receive direct support through enactable software process models which are mechanically interpreted by process engines within process-aware tools, in order to orchestrate the performance of the actual development and to automate it as far as possible [36, 43].

Very few experiments aim at applying mechanical support to OSSDPs. Jensen and Scacchi describe a prototype for enacting formal models written in the PML language in order to simulate them [30]. Use of the GENESIS platform, a process-aware toolkit for supporting distributed software development, is discussed in the context of OSS projects [45]. The broader idea of ‘workflow support’ is sometimes mentioned, for specific process fragments [46], such as routing proposed changes by non Committers to Committers, notifying all developers that have recently checked-in changes to a group of code that its documentation has been updated, tracking and communicating workflow progress to project leaders. It is worth noting that the actual enactment of processes is not in the scope of SPEM [11].

Some OSS developers do not like the idea of specifying process models “I am opposed to a long rule-book as that satisfies lawyer-tendencies, and is counter to the technocentricity that the project so badly needs” (email reported in [40]). Process enactment support for development tasks would certainly be rejected by most of the OSS developers. However, tasks that are not development related, should be automated so that the Committers can do what they do the best and enjoy the most: develop software [40].

6. CONCLUSION

Today, software engineering offers a spectrum of approaches with process intensive methodologies such as the Capability Maturity Model [47] at one extreme and lightweight methodologies such as open source or agile methodologies [48] at the other. Lightweight methodologies emphasize the fact that software development is fundamentally a human activity. Some approaches which combine control with some flexibility, like Rational's Unified Process [49] are positioned in the middle.

Open source development is not a silver bullet [50] but just an alternative approach showing how the Internet can change the way software is constructed, deployed, and evolved. Open source development 'offers useful information about common problems as well as some possible solutions for globally distributed product development' [8]. Process modeling gives a great opportunity to analyze, compare, visualize, and transfer for reuse these possible solutions.

This chapter defines a multi level modeling approach for describing in a common framework both the generic characteristics of OSSDPs and the special features of specific projects. The chapter contrasts NetBeans and Apache release management processes, and discusses the strengths and weaknesses of the approach and the SPEM notation. This work is a first step towards the systematic description and analysis of OSSDPs. We aim at characterizing different families of OSSDPs by showing that each family share a larger set of common properties than those of the generic level.

It is tempting to suggest that closed source development and open source development could be hybridized [51]. The general opinion recognizes the interest of reusing some principles and solutions of OSSDP in other contexts for developing specific software products, such as tools and platforms [51], and more generally all applications faced by developers [3]. Our approach could help such hybridization by providing a common framework for analysis and discussion of all the processes by which a group of people can produce high quality software with a cooperative style of development departing from the traditional hierarchical and management driven style.

REFERENCES

- [1] *An Introduction to Open Source Communities*, E.E. Kim, Blue Oxen Associates, Technical report, BOA-00007, 2003.
- [2] *Results from Software Engineering Research into Open Source Development Projects Using Public Data*, S. Koch, and G. Schneider, Diskussionspapiere zum Tätigkeitsfeld Informationsverarbeitung und Informationswirtschaft, Hansen, H.,R., and Janko, W., 22, Wirtschaftsuniversität Wien, Austria, 2000.

- [3] *Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors*, A. W. Brown, and G. Booch, ICSR-7, LNCS 2319, Springer-Verlag, 2002, pp. 123-136.
- [4] *A Case Study of Open Source Software Development: The Apache Server*, A., Mockus, R.T. Fielding, J. Herbsleb, 21st International Conference on Software Engineering (ICSE), Los Angeles, CA, 1999, pp 263-272.
- [5] *Two case studies of open source software development: Apache and Mozilla*, A. Mockus, R.T. Fielding, and J. Herbsleb, 2002. ACM Transactions on Software Engineering and Methodology, 11(3), ACM, 2002, pp 309-346.
- [6] *An Overview of the Software Engineering Process and Tools in Mozilla Project*, C.R. Reis, and R. Pontin de Mattos Fortes, Workshop on OSS Development, Newcastle upon Tyne, UK, 2002, 162-182.
- [7] *Issues and Experiences in Modeling Open Source Software Development Processes*, W. Scacchi, 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, 121-126.
- [8] *A Descriptive Process Model for Open-Source Software Development*, Johnson, K., Master Thesis, Univ. Calgary, Alberta, 2001.
- [9] *The Cathedral & the Bazaar - Musings on Linux and Open Source by an Accidental Revolutionary*, Raymond, E. S., O'Reilly & Associates Inc., Sebastopol, 1999.
- [10] *The-user developer convergence: Innovation and software systems development in the Apache project*, Osterlie, T., Master Thesis, Norwegian Univ. of Science and Technology, 2003.
- [11] *Software Process Engineering Metamodel Specification, version 1.0*, OMG Document formal/02-11-14, 2002.
- [12] *Unified Modeling Language Specification, version 1.5*, OMG Document formal/2003-03-01, 2003.
- [13] *Free Software Foundation Web site*, <http://www.gnu.org>.
- [14] *The Free Software Definition* (on line), Stallman, R., <http://www.fsf.org/philosophy/free-sw.html>, 1999.
- [15] *Open Source Initiative Web site*, <http://www.opensource.org>.
- [16] *The open source Definition Version 1.9* (on line), OSI, <http://www.opensource.org>, 2003.
- [17] *Netbeans project Web site*, <http://www.netbeans.org>.
- [18] *Eclipse project Web site*, <http://www.eclipse.org>.
- [19] *Software Development Practices in Open Software Development Communities: A Comparative Case Study*, W. Scacchi, First Workshop on Open Source Software Engineering, ICSE'01, Toronto, Ontario, Canada, 2001.
- [20] *Characterizing the OSS process*, A. Capiluppi, P. Lago, M. Morisio, 1st Workshop on Open Source Software Engineering, ICSE'01, Toronto, Canada, 2001.
- [21] *Meta Object Facility Specification, version 1.4*, OMG Document formal/02-04-03, 2003.
- [22] *Evidences in the evolution of OS projects through Changelog Analyses*, A. Capiluppi, P. Lago, M. Morisio, 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, pp. 19-24.
- [23] *Cave or Community? An Empirical Examination of 100 Mature Open Source Projects*, S. Krishnamurthy, First Monday, 7 (6), 2002.
- [24] *The stellar model of open source, Version 0.1* (on line), Mazzocchi, S., Java Apache Project, <http://bioinformatics.weizmann.ac.il/software/apache/java/framework/stellar.html>, 1999.
- [25] *Homesteading the noosphere, Version 3.0* (on line), Raymond, E.S., <http://www.catb.org/~esr/writings/homesteading/>, 2000.

- [26] *Open Source Software Development Processes, version 2.5* (on line), W. Scacchi, <http://www.ics.uci.edu/~wscacchi/Software-Process/Open-Software-Process-Models/Open-Source-Software-Development-Processes.ppt>, 2002.
- [27] *Improving the Open Source Software Model with UML Case Tools*, Gilliam, J.O., Linux Gazette, 67, June 2001.
- [28] *A Framework for Open Source Projects*, Rothfuss, G.J., Master Thesis, Department of Information technology, University of Zurich, 2002.
- [29] *The Cathedral and the Bazaar, version 3.0* (on line), Raymond, E.S., <http://www.catb.org/~esr/cathedral-bazaar/cathedral-bazaar/>, 2000.
- [30] *Simulating an Automated Approach to Discovery and Modeling of Open Source Software Development Processes*, C. Jensen, and W. Scacchi, ProSim'03 Workshop on Software Process Simulation and Modeling, Portland, Oregon, 2003.
- [31] *Open Source Software Development Processes in the Apache Software Foundation* (on line), Ata, C., Gasca, V., Georgas, J., Lam, K., Rousseau, M., <http://www.ics.uci.edu/~michele/SP/final.doc>, 2002
- [32] *A First Look at the NetBeans Requirements and Release Process* (on line), Oza, M., Nistor, E., Hu, S., Jensen, C., Scacchi, W., <http://www.ics.uci.edu/~cjensen/papers/FirstLookNetBeans/>, 2002
- [33] *Release Management Within Open Source Projects*, J. R. Erenkrantz, 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, pp. 51-56.
- [34] *Automating the Discovery and Modeling of Open Source Software Development Processes*, Jensen, C., Scacchi, W., 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, pp. 75-78.
- [35] *Distributed Collective Practices and Free/Open-Source Software Problem Management: Perspectives and Methods*, Gasser, L., Ripoche, G., CITE'03, 2003, pp 349-365.
- [36] *A Structured Conceptual and Terminological Framework for Software Process Engineering*, Lonchamp, J., Second Int. Conf. On the Software Process (ICSP2), Berlin, RFA, IEEE Computer Society Press, 1993, pp 41-53.
- [37] *A field study of the software design process for large systems*, B. Curtis, H. Krasner, N. Iscoe, Communications of the ACM 31(11), 1988, pp 1268-1287.
- [38] *Understanding Continuous Design in F/OSS Projects*, L. Gasser, G. Ripoche, W. Scacchi, B. Penne1, ICSSEA, Paris, France, 2003.
- [39] Putting it all in the trunk: Incremental software development in the FreeBSD open source project, Jorgensen, N., *Information Systems Journal*, 11, 2001, pp 321-326.
- [40] *A project model for the FreeBSD Project*, Saers, N., Master thesis, University of Oslo, 2003.
- [41] *SPADE: An environment for software process analysis, design, and enactment*, In Software Process Modelling and Technology, chapter 9, Research Studies Press, 1994, pp 223-247.
- [42] *Software process analysis based on FUNSOFT nets*, W. Deiters, V. Gruhn, Systems Analysis Modelling Simulation 8 (4-5), 1991, pp 315-325.
- [43] *Software Process: Principles, Methodology, Technology*, J.C. Derniame, B.A. Kaba, D.G. Wastell (Eds.), Lecture Notes in Computer Science 1500, Springer Verlag, 1999.
- [44] *Towards a Reference Framework for Process Concepts*, R. Conradi, C. Fernstrom, A. Fuggetta, R. Snowdon, Software Process Technology - Proceedings of the 2nd European Software Process Modeling Workshop, Trondheim, Norway, Springer Verlag LNCS 635, 1992, pp 3-17.
- [45] *Open-Source Development Processes and Tools*, C. Boldyreff, J. Lavery, D. Nutter, and S. Rank, 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, pp 15-18.

- [46] *Beyond Code – Content Management and The Open Source Development portal* (position paper), Halloran, T.J., Scherlis, W.L., Erenkrantz, J.R., 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, pp. 69-74.
- [47] Capability Maturity Model Version 1.1., Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V., *IEEE Software*, 10, 4, 1993, pp 18-27.
- [48] *Is Open Source Software Development Essentially an Agile Method?*, Warsta, J., Abrahamsson, P., 3rd Workshop on Open Source Software Engineering, ICSE'03, Portland, Oregon 2003, pp 143-147.
- [49] *The Rational Unified Process – an Introduction*, Kruchten, P., Addison-Wesley, 1998.
- [50] *When is Free/Open Source Software Development Faster, Better, and Cheaper than Software Engineering?*, Scacchi, W., Working Paper, Institute for Software Research, UC Irvine, 2003.
- [51] *Why Not Improve Coordination in Distributed Software Development by Stealing Good Ideas from Open Source?*, Mockus, A., Herbsleb, J.D., 2nd Workshop on Open Source Software Engineering, ICSE'02, Orlando, Florida, 2002.