Improvement Opportunities for the Open Source Software Development Approach and How to utilize them

Paper for the OSSIE ’03 Workshop

Stefan Dietze

Fraunhofer ISST, Mollstr. 1
10178 Berlin, Germany

stefan.dietze@isst.fhg.de

Abstract. This paper describes some aspects of the software development approach which is used in open source software development projects and has evolved over time to a successful software development model. It identifies and the core processes and the deployed software infrastructure of this software development model. This enables the identification of possible improvement opportunities and possibilities to enhance the open source software development model and to adopt some elements of this approach in a commercial software development environment. Since open source software development has already been proven, that it is able to produce successful software products it seems to make sense to integrate some successful elements of this approach into proprietary and distributed software development projects.

1 Introduction

Open Source Software (OSS) has reached a remarkably high popularity in many different application domains in the last years. This suggests the hypothesis that the underlying development model which obviously has the ability to produce successful software products should be considered as a reliable and viable approach in the area of software engineering. In contrast to proprietary software engineering this new paradigm of software development has not yet been researched much despite the fact that open source based software products have reached a high level of software quality and popularity across many different usage areas.
This success of Open Source Software leads to the suggestion that the development processes in general and especially the requirements engineering processes are well

suited to the demands of users and developers of OSS. Therefore, these practices and the deployed software tools used to support the identified processes need to be analysed in order to find out, how to integrate successful elements of the OSS approach into proprietary software development projects.

This paper contains some results of an empirical comparative case study of OSS development processes aimed at the identification and formal specification of a descriptive process model for evolutionary and collaborative OSS development. The study is focussed on the processes, roles, artifacts and the deployed software infrastructure, which is used to support the development approach. The developed descriptive process model enables the further improvement of the identified processes and its software infrastructure and opens the opportunity to integrate such practices into traditional software development approaches.

2 Open Source

This chapter provides a brief overview on open source software and its development processes.

2.1 Open Source Software (OSS)

An appropriate explanation of the open source term is provided by the Open Source Initiative (OSI), which has developed the Open Source Definition (OSD) [1]. This definition contains a set of criteria that have to be considered in the software license models used for OSS in accordance with the OSD. The following aspects summarize the fundamental elements of the OSD:

- Free distribution and redistribution
- Publicly available source code
- Possibility of source code modification and redistribution
- No discrimination of certain users/groups or employment domains

All license models that follow the criteria defined in the OSD can be considered to be compatible to the understanding of OSS as defined by the OSI. In addition the OSI provides a list of all certified software licenses [2].

2.2 Open Source Software Development

While the popularity of OSS is growing continously the research in the underlying development practices was not developed simultaneously. Although many existing OSS projects have successfully developed individual practices and use specific processes, it is possible to define some common characteristics that can be identified in most of the OSS-Development projects [4], [5], [6], [7], [8]:

- Collaborative development
- Globally distributed actors
- Voluntariness of participation
- High diversity of capabilities and qualifications of all actors
- Interactions exclusively through web-based technologies
- Individual development activities performed in parallel
- Dynamic publication of new software releases
- No central management authority
- Truly independent, community-based peer review

This leads to the metaphor of a “bazaar” that represents the characteristics of the OSS development practices in contrast to a “cathedral” representing the centralized and strictly controlled traditional software development [9].

3 Identified Process Model

Based on the comparative case study, a generalizable OSSD process model can be defined, which describes the identified OSS development approach in every relevant aspect. All identified processes are defined in different process views, describing the process from different perspectives according to an preliminary defined meta model based on the UML meta model (Unified Modelling Language). The following chapter describes some of the key aspects of the identified process model.

3.1 Initial Prototype Development vs. Gradually Software Improvement

OSS development projects can be divided into the initial processes aimed at the development and release of the initial prototype of the OSS and the subsequently following process of gradually software improvement. The initial prototype development mostly is performed by the initiator of the project and takes place in a closed and sometimes commercial environment. This can be considered to not being part of the real and collaborative OSS development process. The original OSSD characteristics of collaborative distributed development only are associated with the process of gradual software improvements. These improvements contain both bug fixes and enhancements of system functionality.

3. 2 Key Processes and Roles

The initial release of the OSS prototype can be perceived as the starting point of the OSSD process, which in turn is a gradual process of software improvement. The OSS approach can be divided in the following key process categories:

- Management processes

- Environment processes
- Development processes

Whereas the development processes describe the collaborative and highly distributed activities to improve the source code and to develop all projects artifacts, the management and environment processes support the development activities from a central position. The activities within these main processes are executed in parallel and mainly independent from each other.

The following figure (Fig.1) represents these core process categories and the roles involved in this model in a UML-based use case view:

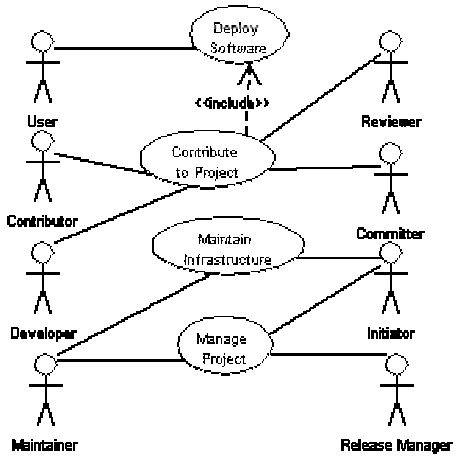


Fig. 1. Identified key processes and roles

All collaborative development activities are represented by the use case 'Contribute to Project' and are executed by distributed actors who could be aggregated to the following roles, which were identified in the case studies:

- User
- Contributor
- Developer
- Reviewer
- Committer

These roles usually are not defined explicitly but describe a certain set of actors, who fulfill a certain set of functions and tasks. It can also be defined a common set of characteristics, e.g. user privileges, which all actors fulfilling a certain role are associated with.

Whereas the user role simply represents passive users of the OSS, every actor actively involved in the process can be regarded as a contributor. Typical activities of a

contributor are e.g. the activities of contributing bug reports, enhancement requests or documentation artifacts. Actors identified as developers already have modified source code and submitted code modifications in the form of patches to the community. Developers, as they are defined in the process model do not have the ability to integrate the patch into the central development source code. The user privilege for committing source code modifications only is assigned to a few users, the so-called committers. The assignment of this privilege is primarily done by the maintainer, who represents the final management authority of the project. Reviewers are actively involved in the discussion and review of patches and in the most cases they also are developers or committers.

Usually an actor is associated with more than one role. For example the development of source code as a developer implies the usage of the OSS and the submission of patches makes a developer also being a contributor.

3.3 Central Maintainers and Core Developers

The maintenance processes for maintaining the infrastructure and to support and coordinate all development activities primarily are executed by a central maintainer or a maintenance committee. In fact the initiator of the project is in most cases the same person as the primary maintainer. The maintenance actors provide central services, which cannot be provided by globally distributed actors for example a common software infrastructure that provides central access to all shared resources.

In many OSS projects, the role of a release manager could be identified whose primary task is to create and publish new software releases [3],[4], [10], [11]. This role is part of a core of developers, whose actors are responsible for a large number of source code modifications and peripheral activities. This developer core could be identified in most of the OSS projects and very often is responsible for activities which require enhanced qualifications, skills, privileges and knowledge of the source code. This comprises for example the commit of patches, the test processes as part of a release process, or the patch-review activities when the development source code is closed and in “code freeze”.
As described above, these roles and actors are necessary to provide a certain level of quality assurance and to supplement processes which might not have been accomplished sufficiently by the distributed actors.

3.4 Process Parallelism and Process Autonomy

The overall process modell at its highest level of abstraction is represented in the following figure (Fig.2).

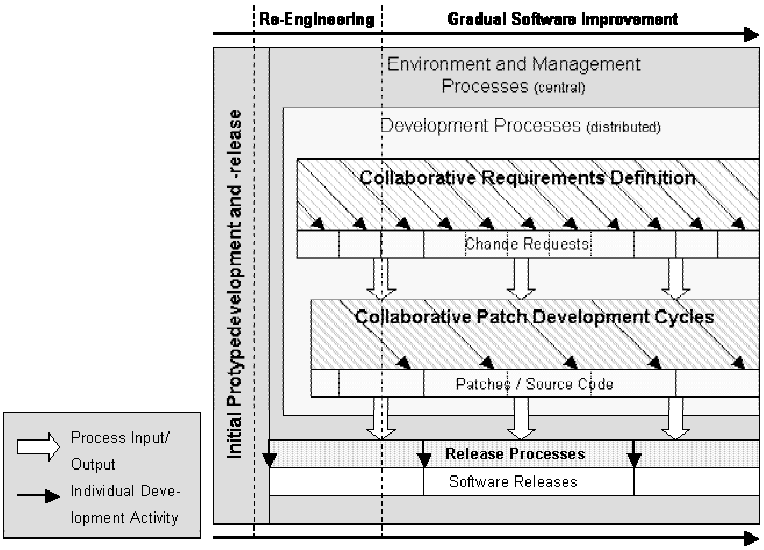


Fig. 2. Overall process model of the OSS approach

This model contains the key processes of the OSS process model and is defined relatively informal in order to provide a brief overview of the central process elements.

After the initial development and release of the prototype, first environmental and management activities have to be accomplished in order to establish a project community and to setup an initial software infrastructure, e.g. a website and a central source code repository. These environmental and management processes are primarily executed by the maintainers of an OSS project and are necessary to enable the development processes by a distributed project community.

For means of simplification only the requirements definition and the patch development processes are visualized as explicit development processes because both represent the primary activities of OSS development. A very important aspect of the distributed activities within the development processes is the fact, that the most processes are executed continously, in parallel and mainly idependent from each other. This separates the OSS approach from all traditional software engineering approaches and current development practices within a commercial environment.

For example a developer who wants to develop and submit a patch acts absolutely independent from other developers and also from contributors who are contributing code change requests. The only relation between these two processes is the fact, that the developer selects a certain requirements artifact, i.e. a certain bug report or enhancement request and implements this requirement. Therefore, the process of contributing change requests is performed independently from the development of source code patches, but the database of change requests is the starting point for every patch development cycle.

A similar relation does exist between the management process of publishing a new software release and the collaborative patch development cycles. Whereas these release processes are executed continously and mainly independent from each other the source code database, respectively all committed patches, can be regarded as the basis for every software release process.

This high level of process autonomy and the fact, that in general every actor selects activities by himself, the overall development progress can not be planned and scheduled and no services can be guaranteed. These weaknesses will be adressed with chapter '4 Improvement Opportunities for the OSSD approach'.

3.5 Identified Infrastructure

The collaborative development activities partly are supported by software tools which can be divided into communication and collaboration tools [12]. These tools represent the central infrastructure which supports the distributed development in OSS projects up to now and consists primarily of open source software. The software infrastructure described in this chapter provides a minimal set of functionalities necessary for OSSD projects but can be enhanced comprehensively as it is suggested in chapter '5.2 Enhancement of the Software Infrastructure'.

The communication infrastructure enables the communication between all involved highly distributed actors. Primarily, communication is based on mailing lists and newsgroups partly extended by IRC-based (Internet Releay Chat) communication. The project website represents the basis for all user interactions and is used as central communication platform and provides user interfaces to all centrally provided software tools. Thus, it enables the interaction between all actors and the usage of the central collaboration tools which provide services like bug tracking or source code control.

Software tools used for configuration management and version control of the centrally organized development source code can be seen as the most important collaboration tools. In this area, CVS (Concurrent Versions System) has established as a quasi-standard. CVS provides a central source code repository and enables the distributed development of a commonly used base of development source code by keeping track on every committed source code modification.

Another very important collaboration software deployed in most OSS projects is the bug tracking system that represents a central repository of all bug reports and enhancement requests. Because every individual patch development cycle is based on such a specific change request, this tool is a central part of the infrastructure. It enables the structured definition of software requirements based on metadata and thus facilitates the technological implementation of lifecycles for these requirements artifacts.

4 Improvement Opportunities for the OSSD Approach

The detailed analysis and formal modelling of all aspects of the OSS approach led to the definition of some implications of the defined process model that can represent the base for further improvements and enhancements. Approaching these goals could lead to a higher level of quality assurance for both, the development processes and the developed OSS and could make the OSS development approach more applicable for the development of proprietary software in a commercial environment.

The following goals and implications concerning further process improvements or infrastructural support enhancements could be identified.

4.1 Quality Assurance: Compensation of insufficiently accomplished Processes

The detailed analysis of open source projects led to the realization that most contributors tend to work on different tasks with different intensity. The focus of most developers is on implementation of source code changes [13],[14],[5]. Therefore, the following activities should be supported by the definition of dedicated roles, processes, artifacts or supporting software tools:

- Softwaredesign
- Softwaretest (pre-release test)
- Documentation development
- Support (user- and developer support)

4.2 Minimization of redundant Activities

As described in [13] the parallel and autonomous character of the most processes causes a high level of redundancy within many activities which in turn is responsible for an unefficient deployment of developer resources. This leads to the requirement of an enormous high transparency to enable the actors the best possible access to all resources. This can be enforced by the central organisation of all informations and artifacts in a consistent and non-redundant way.

4.3 Increasing the Community by Minimization of Entry Barriers

One important success factor is the size of the project's community because a higher number of users and developers of the OSS best supports the distributed development model. Brooks Law ("Adding manpower to a late software project makes it later.") is not applicable to the OSS development approach because of the high grade of parallelism of all distributed individual development activities and their large autonomy and independence. This leads to the requirement to minimize the entry barriers for new actors in the process. This could be reached by a central and transparent software

infrastructure, high level of software quality e.g. modularization or readability and the exclusive usage of open and nonproprietary standards as part of the infrastructure and in the source code.

4.4 Requirements of Commercial Software Development

To improve OSSD methodologies in general, it seems to make sense, to have a look at the requirements of commercial software development projects. As described in [17], software development processes in a commercial environment underly special circumstances which differ from the OSSD approach in many aspects. Thus, the following requirements are obvious when developing software in commercial software development projects based on contracts with customers:

- Ability to schedule the development processes (milestones, available resources)
- Ability to meet deadlines
- Guaranteed supply of services (support, updates, documentation)
- Exact implementation of previously defined customer requirements
- Minimization of defect density of releases (exhaustive pre-release testing)

The whole development process in a commercial environment is organized to face the demands of the customer. The processes have to meet a schedule which is defined in a contract and therefore all development activities have to be planned according to the requirements of the customer. Thus, deadlines for certain milestones have to be met. Furthermore some services like user support or the supply of updates and documentation have to be provided on a guaranteed service level. As it is typically done in OSSD, the requirements for all implementation activities will be defined in a continous and collaborative process by the whole community, and never closed finally. Instead, in a commercial environment the requirements are defined at the beginning of the development approach by a customer and continously updated by this actor. Nevertheless, the customer expects a very low defect density directly after the release of a new software version, while he is paying a certain amount of money for this release. This expectation contradicts to the 'Release early, release often'-practice that is typical for the OSSD approach. Therefore, a well defined set of pre-release testing activities have to be accomplished before a new software release occurs.

The above described requirements have to be approached by an improved OSSD method, when its elements are considered to be used in a commercial environment. Furthermore, an improved process model which supports these requirements, could be considered as an prescriptive process model for OSSD in general.

5 Approaching Appropriate Processes and Software Support

As described above, a lot of opportunities exist in order to improve the existing OSS development model. Primarily, two possible proceedings could be identified to utilize these opportunities and to approach to some of the described goals:

- Definition of additional roles and processes
- Enhancing the software infrastructure

5.1 Definition of additional Roles and Processes

In order to support all insufficiently accomplished activities (review, test, support) new roles have to be defined or the area of responsibility of already identified roles has to be extended. According to [17], a well defined role concept could compensate most of the lacking services as they can be identified in many OSS projects.
An obvious solution would be to assign the new responsibilities to the team of core developers. The additional processes associated with these roles should not replace but supplement the identified development processes as they are defined in the descriptive process model. These processes aim at guaranteeing a higher level of quality assurance in OSS projects and could contain e.g. a dedicated source code review process, a requirements review process or a release process for the periodically supply of test builds.

This could lead to a higher level of quality assurance for both the developed artifacts and the deployed development processes. Furthermore, assigning tasks and deadlines to certain actors, i.e. to core developers and especially to paid developers of a proprietary software project, can make the overall development processes much more schedulable and ensures the achieving of milestones within a given timeline. This in turn can make the OSS approach more applicable to contract based software development in a commercial environment.

5.2 Enhancement of the Software Infrastructure

The primary software development functionalities identified in the generalizable process modell are provided by the source code control system and by the bug tracking system [12]. These functionalities can be extremly enhanced by defining an appropriate software infrastructure which integrates all necessary functionalities to support the entire process model and to partly automate some of the core development processes.

5.2.1 Infrastructural Requirements

In general, an appropriate software infrastructure should provide transparently central access to all information relevant to the distributed actors. This information should be organized consistently and without redundancy to enable the collaborative usage of

common information resources and artifacts. Concerning the deployed software technology, the following requirements could be identified:

- Webbased software
- Support of standardized and non-proprietary artifact formats
- High level of usability
- No requirement of proprietary clientside software

Moreover, there also could be observed the tendency of OSS communities to exclusively rely on OS based software tools, what is called “Boot Strapping” [15]. This should also be an requirement to an integrated software framework both on the clientside and on the serverside.

5.2.2 Software Functionalities

The main functionalities that should be supported by an appropriate software infrastructure are concerned with supply of adequate and consistent communication channels and the appropriate management of all information resources and artifacts. Therefore, the implicitly identified roles and activities, as described above, should be explicitly supported by an adequate software infrastructure and the different processes and workflows should be automated and supported. Thus, it should provide the following additional functionalities and features which could enhance the basic software functionalities like source code configuration management and bug tracking:

- Integrated Issue Tracking
- Content Management
- Workflow Management

It seems meaningful to enhance the bug tracking functionality in order to build an integrated collaboration management system or issue tracking system, which provides a central repository for artifacts, which enable the structured description of all tasks and change requests, e.g. support requests or code change requests.. Such an integrated system would enable the community to describe every issue based on structured metadata, to control of the progress of all collaborative tasks independent from their subject and to manage code change requests, documentation development requests or environmental change requests dedicated to the maintainer of the infrastructure in one commonly used system.

Nevertheless such a task tracking system could be used to capture and manage all support requests of the community to control their progress. The integrated management of all support requests in such an infrastructure could compensate an enormous weakness of OSS projects, that is the lack of support. With the development of a central support database, all support informations can be defined consistently and without redundancy in a central repository, which enables queries in all available support information and to control all unanswered requests.

Furthermore, all content objects, e.g. documentation artifacts and web content, should be managed by an appropriate software infrastructure in order to enable collaborative editing of these artifacts by all involved actors [16].

By implementing well defined user rights based on the defined roles within such an infrastructure, appropriate roles can be assigned to all actors. In addition, routing of artifacts to the appropriate actors based on the current state of the artifact lifecycle or automated e-mail-information of certain roles triggered by defined events can be used to implement workflow functionalities and to automate all development processes. Workflows can also be supported by an approriate information management. Therefore, it seems to be useful, to provide actors with all artifacts, which contain information related to the task he is going to do. For example, a developer, who wants to fix a bug in the source code, could be interested in the change request artifact, describing the source code modifications causing the current bug. Furthermore, some mailing list entries could contain useful information about the latest development activities according to this piece of code. The structured description of artifacts based on appropriate metadata enables the integrated managenet and could be the basis for the linkage of all artifacts, which contain semantic relationships.

6 Conclusion

In general, there exist a lot of opportunities to improve the OSSD approach, but nevertheless it can be considered as a successful model that has been proven by the success of several important OSS projects like Apache HTTPD or the Linux Kernel. This paper has described some aspects of the OSSD approach and has outlined some current weak points and improvement opportunities. Furthermore, some possible proceedings for approaching to a better process model where presented which could lead to a higher level of efficiency and quality assurance for both, the processes and the developed software. This could be achieved by the assignment of tasks, e.g. for source code review or documentation development to certain actors who are part of the developer core of the project. Another important approach is the improvement of the software infrastructure, which is used to support the whole development model. Thus, content– and workflow management functionalities could be integrated and an integrated management of issues and artifacts seem to be necessary.

The OSS based approach for collaborative software improvement can be seen as a viable process for software development that has been developed by users and developers of OSS in an evolutionary process throughout the previous evolution of OSSD practices. It is well suited to detect the requirements of the globally distributed users of a software and already proved its ability to produce successful software products.
Today, most software development projects, also proprietary ones, base on the collaboration of a more or less distributed developer and user community. Therefore the OSSD processes should be considered as an appropriate approach for distributed software development. It seems possible to integrate successful elements of the OSSD model into commercial software development projects. This could include for exam-

ple the adaptation of certain processes, like the bug-driven patch development process, or the adaptation of elements of the software infrastructure, which is used in OSSD. Most of all, the requirements analysis and definition processes are able to supplement the traditional requirements capturing processes used in proprietary software development projects.

References

1. Open Source Definition by Open Source Initiative, Available: http://opensource.org/docs/definition.php
2. OSI certified software licenses by The Open Source Initiative, Available: http://opensource.org/licenses/index.php
3. Davor Cubranic: „Open Source Software Development“, (2002, March 26), Available: http://sern.ucalgary.ca/~maurer/ICSE99WS/Submissions/Cubranic/Cubranic.html
4. Karl Fogel, Moshe Bar: “Open Source-Projekte mit CVS”, MITP 2002
5. Christina Gacek, Tony Lawrie, Budi Arief: „The many meanings of Open Source“, (2002, February 27), Available: http://citeseer.nj.nec.com/485228.html
6. Paul Vixie: „Software Engineering“, in: „Open Sources – Voices from the Open Source Revolution“, Herausgeber: Chris DiBona, Sam Ockman, Mark Stone, O’Reilly & Associates 1999
7. Walt Scacchi: “Software Development Practices in Open Software Development Communities: A Comparative Case Study“, (2002, May02), Position Paper, Available: http://opensource.ucc.ie/icse2001/scacchi.pdf
8. Stefan Koch: “Entwicklung von Open Source und kommerzieller Software: Unterschiede und Gemeinsamkeiten”, (2002, June 02), Available: http://wwwai.wu-wien.ac.at/~koch/forschung/sw-eng/vg01-folien.pdf
9. Eric Steven Raymond: „The Cathedral and the Bazar“, O’Reilly UK, 2001
10. Audris Mockus, Roy Fielding, James Herbsleb: „A Case Study of Open Source Software Development: The Apache Server“, (2002, May 22), Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), Available: http://citeseer.nj.nec.com/mockus00case.html
11. Christian Robottom Reis, Renata Pontin de Mattos Fortes: „An Overview of the Software EngineeringProcess and Tools in the Mozilla Project“, (2002, May 17), Available: http://www.async.com.br/~kiko/papers/mozse.pdf
12. Walt Scacchi: “Software Development Practices in Open Software Development Communities: A Comparative Case Study“, (2002, May02), Position Paper, Available: http://opensource.ucc.ie/icse2001/scacchi.pdf
13. Steven Weber: „The Political Economy of Open Source Software“, (2002, April 17), URL: http://e-conomy.berkeley.edu/publications/wp/wp140.pdf
14. Michael W. Godfrey and Qiang Tu: „Evolution in Open Source software: A case study“, in: Proceedings of the International Conference on Software Maintenance (ICSM 2000), San Jose, California, 2000, Available: http://plg.uwaterloo.ca/~migod/papers/icsm00.pdf
15. T.J. Halloran, William J. Scherlis: “High Quality and Open Source Software Practices”, Position paper, Available: http://opensource.ucc.ie/icse2002/HalloranScherlis.pdf
16. Justin R. Erenkrantz, T.J. Halloran, William L. Scherlis: „Beyond Code: Content Management and the Open Source Development Portal“, in Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE’03, May 2003
17. Jianjun Deng, Tilman Seifert, Sascha Vogel: “Towards a Product Model of Open Source Software in a Commercial Environment”, in Proceedings of the 3rd Workshop on Open Source Software Engineering at the ICSE’03, May 2003