Towards a Product Model of Open Source Software in a Commercial Environment

Jianjun Deng, Tilman Seifert, Sascha Vogel
Software and Systems Engineering
Technical University of Munich
Boltzmannstr. 3, D-85748 Garching, Germany
{deng, seifert, vogels}@in.tum.de

Abstract

Commercial organizations show increasing interest in using and improving Open Source (OS) software – they want to participate in the OS community, but still have commercial interests. We believe there is not necessarily a conflict of interests, and the OS community can gain from the participation of commercial organizations. But this special situation requires a sound understanding of how open source development works. In this paper we describe the process of developing OSS in a commercial environment. We identify which work products have to be built up and discuss the differences between traditional open source development without financial interest and commercial OS development. The described process model also focuses on licenses, tool support and infrastructures. The concept we introduce is based on an evaluation of different projects and case studies.

1. Introduction

There are many reasons for commercial organizations to be interested in using OS software (OSS), e. g. usually the quality or the transparency of OSS. Many companies not only use OSS, but also take advantage of the available source code and work on the code base to improve it, fix bugs, or add features or interfaces in order to make it even more useful for their specific purposes.

1.1. OS Development in a Commercial Environment

The development of OSS in a commercial context rises interesting questions, including architecture, development life cycle, tool support, licenses, business models, and their respective interactions and dependencies.

In this paper we consider the development process. We analyze the distinct properties of OS projects and commercial projects; we identify requirements for a development process that is appropriate for developing OSS in a commercial context, and we derive a development life cycle fulfilling these requirements.

1.2. Mutual Gains

OS is based on a "give and take" approach. This works not only between the developers themselves, but also between different types of organizations with different backgrounds and incentives. The participation of companies in OS communities offers chances for both sides.

Commercial organizations can profit from using OSS, but on the other hand, there are concerns for a commercial organization when the OSS is not only used, but also improved. Some are of a technical nature, e. g.: How well are OS projects documented? How does maintenance work, when nobody can be held responsible? There are also legal concerns, e. g. about licenses, liability, and a business model that works without conflict with the OS licenses.

As soon as organizational and legal difficulties are solved, commercial organizations can offer a lot for OS projects and contribute to the OS community, such as: developers with defined budget are committed to their assigned tasks and make planning more reliable so that projects can evolve faster, reliable service, thorough documentation. The OSS is being incorporated into commercially sold products; if the company improves the OS part by adding features or by removing bugs, these improvements will be given back to the community.

1.3 Paper Structure

This paper is structured as follows: we first focus on the description of some general aspects which are necessary to

define, categorize and to manage OS projects. Particularly, management is a precondition for commercialized development of OSS (see section 3). These aspects define an OS project framework.

Based on this framework in the following we derive different aspects of OSS development. These aspects are called process views such as *software usage* for instance. Each view is assigned to a set of products including required activities, results and further conditions. As products strongly relate to each other, we additionally specify a product network as described in section 4. The relations are characterized by requiring and producing requirements.

OS projects strongly depend on different types as outlined in the project framework. That makes a tailoring mechanism necessary, to adapt the product network and thus the development process to a specific project type. One possible tailoring mechanism is introduced in section 5.

The paper ends with a summary briefly enumerating all results of this paper.

2. Related Work

When considering OS together with the aspect of traditional software engineering and project management in a commercial environment in particular, it is indispensable to examine on one hand the growing and not yet well documented field of OS development and on the other hand the area of traditional process models.

The interface between commercial and OS development is not yet discussed very deeply. In most cases people focus more on business concepts such as [3] or [10].

For a more technical view there are traditional software development processes or assessment methods such as the V-Model [4], the rational unified process (RUP) [5] or the capability maturity model [9].

There are some interesting ideas for processes of OS software development in [8, 11] or [14] but they do not consider OS in a commercial environment.

This paper integrates the idea of building up OSS and offering possibilities to manage and control an OS project. This work is strongly related to existing projects such as Linux [6], Mozilla [7] or Apache [1] to always guarantee for adequacy and substantiality.

3. Framework for OS Development in a Commercial Environment

Based on observations in the OS community and existing and valid process models for traditional software development projects, in the following we describe some general and typical conditions for developing OS software in a commercial environment.

3.1. Project Categories

Architecture, development model, license models, business models, tool support, and infrastructure are strongly related – for software development in general and for OS software development in particular. Especially in a commercial environment, it is crucial to strictly follow the rules given by the different OS licenses. These different licenses impose constraints on the development, they even influence the architecture of a software that includes OS parts as well as closed source parts.

Architecture Types We introduce a categorization of projects based on different architecture types:

- First, we distinguish between “tight” and “loose” coupling of OS and non-OS components. Some OS products or libraries allow to change the source code and to use it even in commercial products, while other OS products inhibit such a use of their sources. The license model might therefore have great influence on the software architecture of the whole product. In figure 1, the left side with types 1 and 3 shows loose coupling, using only defined interfaces; the right side with types 2 and 4 shows tight coupling, e. g. when several components are integrated into one software product.

- Usually, closed source development is based on OS software, and driven by its features and architecture. But there are cases when it is the other way around. This is our second distinction; in figure 1 the upper half with types 1 and 2 shows that OSS is the basis that is used with or integrated into closed source development. In the lower half (types 3 and 4), OSS is based on closed source development. Examples for this situation might be an OS framework that requires a commercial database, or a public interface to a component that is available only as a “black box”, such as cryptographic libraries.

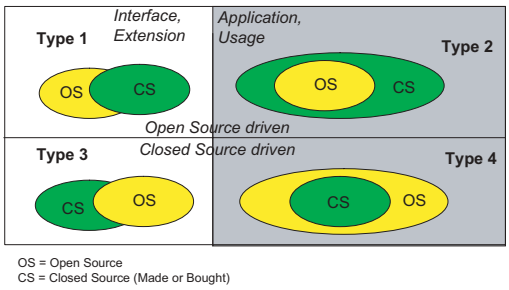

Figure 1. Project Classes

The reasons for choosing one of the architecture types might be motivated not only by technical aspects but also by license models which allow only a specific usage. We need to understand the different license models before we can start to build on or to integrate OS software into any project. Here we clearly see the connection to the business model.

Software Usage We can further distinguish project types according to the usage of OSS: On one hand, there is OSS that is directly integrated into a product that is sold to a customer, and on the other hand there are OSS tools that are used in the development process. Here are some examples for the first case:

- A company wants to use some OSS in order to extend it or to build on it and sell its own part to a customer. This situation is very common; it coincides with types 1 and 2 in figure 1.
- A software company has built a reusable piece of software and wants to publish this piece as OSS. There can be several motivations behind:
  - to demonstrate competence (in order to acquire new projects)
  - to give something to the OS community, and to profit from the improvements that are contributed by the community. Here the company hopes for the dynamics of an interested community.
  - to overcome legal difficulties: The reusable piece of software is not sold by itself but reused in the next project for another customer.

The OS development of tools is even more common. For example, a software company wants to use and enhance a development tool (e. g. Eclipse, the OS Java development environment that can be extended by using its plug-in mechanism) and contributes some extension (e. g. a useful plug-in). This is work that is done anyway, it is directly related to the company's projects but not part of the project results (in the sense that it is sold to a customer).

In all cases, commercial organizations participate in an OS community; they need to integrate the OSS development into their own development process.

3.2. Bridging the gap between management restrictions and OS developing freedom

OS projects work different than commercial projects. Release planning for OS projects is driven by different forces: often technical maturity has higher priority than time-to-market. The development team is highly distributed and collaborates according to rules defined by the smallest common denominator (usually source code, administrated in a repository).

Yet OS projects do not run without goals and plans. The term "OS Project Management" is no contradiction, but the project management activities can be reduced to a minimum. They mainly consist of "technical" activities such as deciding which code to check in and which code not.

In the context of project management it is important to bridge the gap described in the heading of this section. One possible way is to introduce a role concept and assign some responsibilities to each role which than have to be voluntarily accepted by the OS community. In order to keep freedom in the development process all roles related to organizational issues should be assigned to groups of people to realize a democratic decision and steering processes. This is indispensable to perform OS project as this helps bearing lots of individual considerations. However, we introduce a role model with a democratic steering committee and which shows how specific roles can participate directly or indirectly in the steering process.

Role	Resposibilities
User	• Report Errors • Propose new Requirements and Error-Correction
Developer	• Coding and giving feedback for the specification • Documentation • Test for Releases
Commiter	• Review Code of Developer • Participate on vote for Check-in reviewed Code • Check-in Code into repository • Vote for integrating of new commiters
Project board	• Project steering • Vote for integrating new board members • Collect new features and decide which have to be applied • Identify and define new (sub) projects
Release Manager	• Specify new features for future releases • block finalizing new releases • Pre release-testing • commit new releases

Figure 2. Related Project Roles

3.3. Process Requirements

Since we want to integrate OS development into a commercial environment, we consider the most important aspects for both sides.

OS projects as well as commercial projects want to produce high-quality software. But both types of projects face different constraints. The most obvious one is time: In commercial projects the time is limited by budget constraints and contracts. For OS developers, available time per week might also be restricted, but usually there are no hard deadlines to meet. Therefore, they can set different priorities e. g. in release planning.

First, we look at OS development. From section 3.2 we conclude that we need a lightweight, but still planned pro-

cess that takes into account a high degree of distributed collaboration:

1. A *role concept* that is powerful enough to distinguish different tasks, and simple enough to be easily described and quickly adopted by the community.

2. A *document concept* that is capable of installing review mechanisms for quality assurance while at the same time not setting up any barriers that might discourage developers. It needs to canalize the highly distributed and parallel development. It should allow for well-structured documents (such as linking source code to the corresponding documentation).

Second, we consider commercial organizations. They want to use a defined process that induces quality and efficiency. The postulations for a high-quality process of most process models as described in section 2 boil down to three important issues:

3. *Planned activities*: Every step and every decision is planned (a priori) and traceable (a posteriori).

4. *Transparency* of the process and the current state of the project for every stake holder in the project.

5. *Communication*: Many reviews, both "horizontally" (between developers) and "vertically" (between managers, developers and customers/users), are important for everyone to have a realistic view of the project and to keep up a high quality standard.

4. Product Model

Based on the introduced framework, we now define different process views facilitating all aforementioned requirements. The views include the specification of products which sum up activities, results and interrelations between them. Products are also associated to project roles, project categories and some general attributes as outlined in section 4.6. Furthermore, products depend on each other as they produce some output for other products or they require the output of other products as input. The technique to describe product networks is based on the concept of work products which can also be found in [13] and [2]. Based on product network the activity and result schedule can be derived. In the following we introduce and precisely characterize the terms work product and view:

- **Work products:** Traditional process models in general do not suit exactly for OS development processes. Most process models such as the V-Model or the RUP are organized by ordered activities requiring or producing results. In OS projects in most cases it is not possible to exactly specify well-defined activities or results respectively. Instead it is important to specify process object which can jointly describe these two elements and their relations. Here these objects are called work products, whereas the characteristic includes more than just activities and results as described in section 4.6. Work products are integrated into a complex network of products. This network results from relations between products describing specific dependency rules, e. g.: one product needs some other products as input. This way, parallel activities can be synchronized.

- **Views:** As mentioned above the network defines a very complex structure of development objects. For methodological reasons it is important to furthermore structure the network in order to ease understanding. Thus we introduced views, collating sets of products with similar concerns. The views are not disjoint which makes intersections of views possible. Thus products can occur in more than one view.

In the following we roughly describe all identified views for the development product network. To demonstrate our model we additionally refine one specific view before we continue to show in detail how the different technical and organizational aspects of OS and traditional software development can be combined within products.

4.1 View 1: Software Usage

An important element of OS development is a fast feedback loop from end users to the developers. Often this works pretty well because the developers are users themselves, but also users who are not developers need easy access to a bug reporting system or possibilities to suggest new features.

4.2 View 2: Project Initialization

At the project's beginning each project must set up technical infrastructures to facilitate storing and sharing of code and documentation, releasing, logging, bug reporting and tracing, as well as communicating with each other. Usually, a web portal will be realized and configured to give access to these services. To attract people for development and to make the project popular, the initiator should specify the main goals, the architecture as well as structural, behavioral and technological requirements. This information should be propagated to an adequate OS community, e. g. by news postings.

4.3 View 3: Code Development Cycle

In OS code is a shared resource and each role has the right to contribute his code for testing or adding new system features. To ensure that these contributions are highly qualified and fit into the whole system an exhaustive quality assurance cycle has to be established including extensive test and review scenarios before submitting the final code into the repository.

4.4 View 4: Release Management

Release Management is a key approach for successful OS projects. In most cases, the system requirements evolve over time. Thus, release plans have to be defined that allow for changes and the gradual introduction of requirement.

Another role of the release planning is to steer the OS project. This can be achieved by defining milestones for instance three times a year, when new releases have to be assembled. This forces the community to contribute their finalized code to participate in new releases.

4.5 Refining the Release Management View

In the last sections we briefly described the different views to develop OS software. For all views we defined product networks in detail as shown in figure 3. In order not to go beyond this scope we just describe exemplarily the detailed product network for one specific view that is the release management.

Usually the release cycle can be defined as follows. In the beginning of a release cycle, branches are created independently from the main trunk. Bug fixes as well as new features are implemented and tested in separate branches. The project board decides which changes will be added to the main branch of the project. At some point, the project board decides which features will be part of the next release and assumes only bug fixes for the next release of the main branch. The goal is to release only reasonably stable versions of the project.

This way, developers have free choice of what they want to implement or improve, while the project steering committee uses the main releases to integrate the different development branches and to synchronize all efforts.

4.6 Matching organizational and technical Aspects for OS

The work products can be refined in order to integrate the aforementioned project roles, the life cycle requirements and the dynamic of the development process. A work product is instantly illustrated in figure 4. In this figure the characteristic features are listed and described.

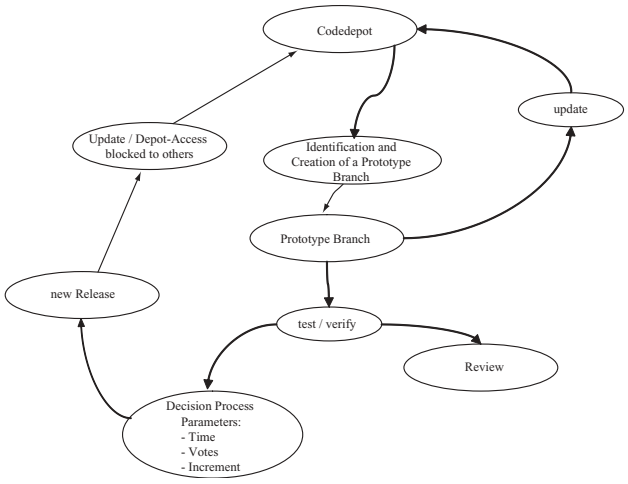

Figure 3. Product Structure *Release Management*

With work products in mind it is possible to define a product infrastructure which gives methodological hints as well as help in developing the needed results in order to efficiently and effectively run through the life cycle.

A work product is described by the following features:

- **Id Work Product:** Name and identifier of the work product such as WP.X.Y and name: *Init Project*. The id results from the view number the work product is part of, followed by a key representing the general alphanumeric order the product has in relation to the rest of the products.
- **Visibility:** The visibility describes whether a product is visible in a special project type.
- **Intention:** The intention classifies whether the product is needed in order to develop an interface or an application, respectively.
- **Purpose:** Goal of creating this product.
- **Description:** This feature describes what to do and what to produce in the context of this work product. For instance address the OS-Community in the view of project initialization.
- **Pre-Relation:** These relations describe necessities which have to exist in order to perform the included activities or to generate results. These necessities result from other products and thus this relation defines a kind of input/output or consumer/producer relation. Usually, relations are identified by work product ids. In case of alternative products, the id is enclosed in brackets. This also holds for the post-relation.

WP.{2, 3, 4}.1		*Codedepot*
Visibility		*Open Source = true and Closed Source = true*
Intention		*Interface Definition = true and Application = false*
Purpose		A Codedepot offers the infrastructure to check-in code ressources such as classes, components as well as documents for comments.
Description		The Codedepot consists of different elements summarized by documents. A document can either be a code fragment of a specific language i.e. Java, C or C++ or a document needed for comments or specification during the design or analysis respectively.
	Activities	* Install tools to support programming * Install a repository to organize documents * Install document structure to contently connect documents
	Results	* Development environment * Define Document infrastructure * Initialize right policy to documents * [Update Document infrastructure]
Pre Relation		* An open source software project has to be initialized. This is done in work product WP.{2}.2 *Initialization*. Particularly, a repository has to exist. * Alternatively the Codedepot can be influenced by updates via the developers which is described in work produc t [WP {3,4}.3] *Update*. This does not have to be related.
Attributes	Associated Roles	* The resposibility is assigned to the *Project Board*. They can `insert, delte` or `modify` elements. * The *Developer* can `read` and `modify` elements * The *User* can `read` elements
	Configuration	Tools are checked-in with version 1.0.0, other documents are not existent.
	Operations	`read, insert, delete, modify, check-in, check-out` documents
	State	The product has the state *in Progress*. All results are not yet realized and all activitites except installing the Respository are not yet startet
Post Relation		* The availability of the Codedepot enables the definition of a new version or release respectively. This is done by work product WP.{4}.3 *Identification and Creation of a Prototype Branch*. * Additionally, the Codedepot offers the possibility to code which is described in WP{2,3}.5 *Modification and Coding* * Furthermore documents can be downloaded and reviewed by the download work product WP{3}.5 *download*

Figure 4. Example of a Work Product

- **Properties:** This feature contains a list of sub-features. These are attributes which can be principally assigned to work products.

  1. **Associated Role:** The role describes a set of functions which have to be fulfilled either by one person or a group of persons. This might be a software developer or the cooperation of a software developer an a system user.
  2. **Configuration:** This attribute describes the state of a configuration of this product, i.e. inserted into repository or logged, etc..
  3. **Operations:** This feature describes which changes are possible in order to keep consistency. For instance substitute variable x by y as well as check-in result z.
  4. **State:** This attribute specifies the state of a product such as: in process, finished, tested, verified, initiated, etc. These states can arbitrarily be defined by the project executives.

- **Post-Relation:** This relation documents the impact of the product according to its current state for associated work products, that is products which state depends on the state of this product.

An instance of a work product is given in figure 4. The example instantiates the given template of work products.

5. Tailoring the Work Products

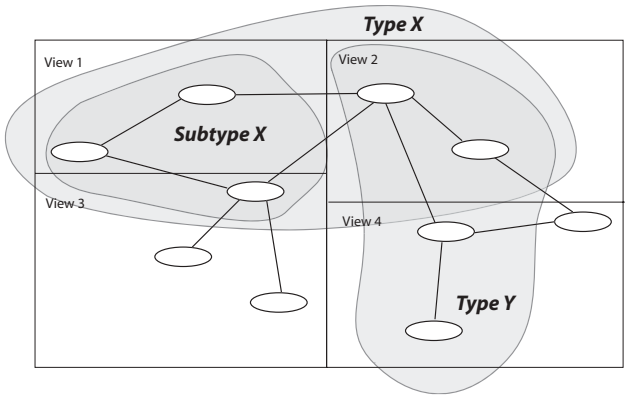

Figure 5. Tailoring the Product Model

As mentioned above each product has to define the context the product is needed for. In other words it has to be stated, whether it is optional or obligatory to fulfill the product. This offers possibility to optimize the product structure according to the identified project type. This way of optimizing the product model is called tailoring.

There is also the possibility to unify different types of tailored product structures. This can be done by simply unifying the assigned products. Analogously other project constellations can be matched.

The tailoring includes one more possibility to tailor the product structure. This can be achieved by considering the pre- and post-relations. For each product which is obligatorily required in one of the pre- or post-relation of one specific tailoring there has to exist one other complementary product which fulfills this requirement. The optionally required products offer room for optimization. This additional tailoring mechanisms can lead to a set of subtypes.

The idea of tailoring is illustrated in a simplified way in figure 5. In this figure the relations between work products

and views as well as the relations between project types and work products can be seen. The tailoring types have intersections but they focus on and emphasis specific parts of the product model. This is independent of the different process views as mentioned above.

6. Summary

In this paper we discussed in detail aspects of OSS development for commercial use. In this context we addressed the research topics of OS development processes, OS process models and tailoring mechanisms for OS process models. In the following the main results in this paper are summed up:

- We identified different categories of OS projects based on existing projects such as Mozilla or Linux, etc.
- Roles have been introduced to enable the management of a distributed and parallel development processes.
- Furthermore, based on the project categories we identified typical requirements which should be realized by instances of OSS development processes.
- Additionally, related to existing projects and the relevant literature, we summed up typical and needed process activities and built up an OS process model based on the concept of work products and product networks.
- Moreover, based on product networks we offered a tailoring mechanism to assign the product model to one specific project type.

Acknowledgments

We want to thank Dr. Thomas Wieland for his constructive and valuable input. This work was funded by the research projects “NOW” and “ViSEK”, both granted by the German Department of Education and Research, BMBF.

References

[1] Apache XML Project. http://xml.apache.org/guidelines.html, Feb 2003.

[2] B. Deifel, W. Schwerin, and S. Vogel. Work Products for Integrated Software Development. Technical report, Technische Universität München, 1999.

[3] M. Fink. *The Business and Economics of Linux and Open Source*. Prentice Hall, 2002.

[4] IABG. Willkommen zum v–modell, November 1999.

[5] P. Kruchten. *The Rational Unified Process – An Introduction*. Addison Wesley, 1998.

[6] Lonux project. http://www.linux.org/, Feb 2003.

[7] Mozilla project. http://www.mozilla.org, Feb 2003.

[8] Open source as a process. http://www.computeruser.com/articles/2008,5,36,1,0801,01.html, Feb 2003.

[9] M. C. Paulk, C. V. Weber, B. Curtis, and M. B. Chrissis. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.

[10] E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O’Reilly and Associates, Sebastopol, California, 1999.

[11] Research on open source software development. http://www.isr.uci.edu/research-open-source.html, Feb 2003.

[12] W. Scacchi. Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering? In *2nd Workshop on Open Source Software Engineering*. ICSE 02, May 2002.

[13] W. Schwerin. Models of Systems, Work Products, and Notations. In *roceedings of Intl. Workshop on Model Engineering ECOOP, Cannes France*, 2000.

[14] Software development practices in open software development communities: A comparative case study. http://opensource.ucc.ie/icse2001/scacchi.pdf, Feb 2003.