Sebastian Prehn

Open Source Software Development Process

Term Paper

July 29, 2007

TU Kaiserslautern
AG Software Engineering
Seminar SS07

Contents

1

Introduction

Open Source Software Development Process - why care?

1.1 Open Source Software Development Process - an Alternative

Open source development originated from the practice of software developers sharing pieces of code for free among their colleagues. The rise of the internet overcame the local and commercial boundaries and enabled larger groups of peers to collaborate.
"The main reason for this success is the growth of the Internet, which made collaboration between programmers feasible on a scale much larger than was possible before"[1].
This gave birth to open-source movements creating non-profit projects e.g. the free UNIX derivative Linux.

While the business world is tempted to make use of free software for the obvious potential of cutting costs, most commercial vendors argue that open-source software lacked reliability and oftentimes a higher TCO were the result of its usage.
Additionally an open-source development team might be seen as a group of hobbyist software engineers simply lacking the proof of competence. Therefore open-source projects tend to have a reputation of having inferior quality compared to their commercial competitors.

Nevertheless a number of open-source success stories has awakened professionals and reduced acceptance problems. The open-source development model seems to become an alternative to classical software development.
Companies like Netscape disclosed their software to gain market share as well as new development power. As a result today many end-users download their free browser and make it the world's number two web browser with steadily

rising market share.[1]
Furthermore, industrial software development commonly utilizes open-source endeavors like the Eclipse IDE or JUnit. In production environments the Apache Webserver, Apache Tomcat Container, JBoss Application Server and several Linux distributions gain more and more importance. This indicates that open software can indeed be of high quality and very successful.

So for a software engineer trying to ensure product quality by conducting software projects in a carefully chosen process, e.g. the V-Modell XT[2], one question arises: How is it possible that these projects succeed in an environment that at first glance seems very chaotic? Can we learn to leverage power of open-source development in commercial projects? In order to find out this paper will contrast open-source and traditional development.

[1] according to `http://marketshare.hitslink.com` "Top Browser Market Share Trend for June, 2006 to May, 2007"

[2] V-Modell XT: Development Standard for IT Systems of the Federal Republic of Germany

2

Open Source Software Development Process

To take a closer look at the way open-source software is being developed one has to accept one first thesis, that is:

"Not every project can be developed in open-source."

Thus, in some cases it is infeasible to judge both fields, commercial and open-source, by the same scale. Yet for a wide variety of systems the model is suitable.[2]

So what are the key features a project has to have to attract developers? In the famous essay "The Cathedral and the Bazaar"[3] Eric Raymond explains that an open-source project must be setup in certain ways to attract developers and users alike. Thus, a new project needs at least an executable prototype or something to really raise interest. [1] A well known personality in the community also greatly helps attracting peer attention. On the other side, very customer specific projects, e.g. yet another custom made ERP system, will be very unlikely to attract developers enough, to make them invest their precious time for free. But why should anyone work for free?

Motivating developers is important as in any software project, yet the motivation of developer in open-source and commercial projects differs. While traditionally developers "grind away for pay at programs they neither need nor love" [3] the developers in open-source projects have a personal interest in their software and its success. Raymond's first thesis says:

"Every good work of software starts by
scratching a developer's personal itch." [3]

This also implies freedom of choice for the developers, since they themselves can decide on the requirements without beeing constrained by contracts.

[1] see chapter on Project Lifecycle 2.2

2.1 Definitions

While open-source software is cleanly defined by the Open Source Initiative [2] there is no clear definition to the development process. That is because the development process itself is not based on a strict process model. Raymond[3] compared classical software development and the open-source development style as the "Cathedral and the Bazaar". According to him creating closed source software is like building a cathedral incorporating strong planning, hierarchies, and centralized development and responsibilities.
The open-source development model could much more be compared to a bazaar, where anyone has a word and differing ideas, approaches and agendas mix. Out of all these submissions a "stable system could seemingly emerge only by a succession of miracles"[3].

According to an Apache case study[4] the usually mentioned main differences between commercial and open-source projects are:

- Open-source systems are built by potentially large numbers (i.e., hundreds or even thousands) of volunteers.
- Work is not assigned; people undertake the work they choose.
- There is no explicit system -level design, or even detailed design
- There is no project plan, schedule, or list of deliverables.

For commercial software engineers it might be surprising that open-source projects relying on far less design documents, contracts, project plans or development processes can have success. This success is presumably due to the different nature of the project setup explained in chapter 2.

2.2 Project Lifecycle

In the classical process, no matter whether it is a classical or agile development method, requirements are elicited first. The customer usually tenders a job, with more or less detailed requirements description. Service providers reply with offers upon which the customer selects a provider. A contract with detailed monetary arrangements is setup.
From that point on the project manager has to make sure that the development costs run within the boundaries of the contract made. This in consideration, the whole project is scheduled into work units and iterations to consume the least resources possible while still fulfilling the requirements and making the deadline. If costs run up to high the project fails.
This means there is a lot of pressure on the development team during the project. At the same time they have to build it the way the customer wants it leaving the developers not much of a choice.

In contrast stands the open-source project lifecycle. Requirements are rarely gathered before the start of the project [5]. Instead someone comes

[2] `http://opensource.org`

up with an initial idea, usually because of some personal interest. If no similar project is available that could be extended he chooses to start a new project and pursues closed source development of an initial prototype. Most sources agree that initial bazaar style development is not possible [6, 3].
With all important design decisions made and a presentable prototype, capable of raising expectations, it is time to go public and acquire fellows. Obviously this works best if project initiator has a good reputation already.[3]
Now it is time to build the community, release early and often and live up to the bazaar style.
When project members, including the initiator, loose interest they hand over the project to someone else. Only if the project looses users and/or developers the project dies.

One can conclude that the open-source lifecycle is less structured and very agile. So agile that the developers themself choose the requirements, their work units and the iteration cycles! In open-source development developers are not stuck in the project against their will, no one forces them[3].

2.3 Roles

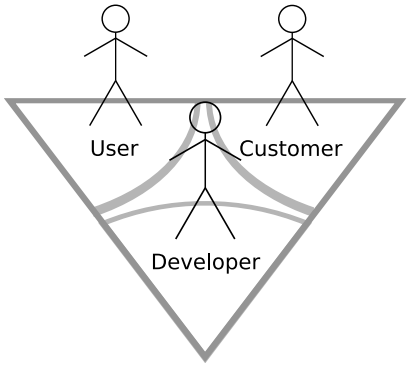

Fig. 2.1. Typical commercial roles and interests.

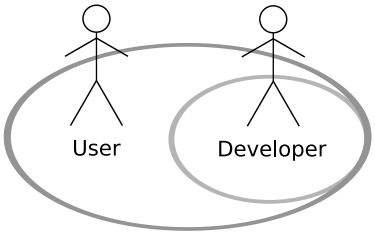

Fig. 2.2. Typical open-source roles and interests.

2.3.1 Roles in Commercial Projects

In a commercial project at least two groups of stakeholders can be identified. The *customer* financing the project and *users* having to work with the

[3] except for some peer pressure and fear of loosing a good reputation

product. As a third role the service provider or short the *developer* can be identified. Between each of these groups exist clashing interests (see figure 2.1). Here some examples:

- Users wish for comfortable functionality, but the customer wants to keep expenses as low as possible.
- The customer wants to change business processes but the users are well-used to old processes.
- Users or Customers have "crazy wishes" conflicting with the functional understanding of the developers.

2.3.2 Roles in Open-Source Projects

In open-source development *users* and *developers* are present as well, but the customer role is eliminated and it is rather split up and distributed to the user and developer role. Some of the customer requirements are shift to the user requirements while the top level projects decisions are made by the core developer team.

It is infeasible to see developers and users as separated groups. The developer **is a** user, this is where his personal interest for the project originated from in the first place. In bazaar style development users are treated as co-developers[3] supplying valuable feedback such as bug reports and patches or fixes.
This results in a large intersection between the two groups and thus in far less clashing interests. (see figure 2.2)

While it is natural to have hierarchies in commercial development it is worth taking a closer look on the developer role in open-source development. Here we usually find a flat hierarchy in order to coordinate the project. The developer role is usually split up into two levels of sub roles: *core developers* and *contributors*. (see figure 2.3)

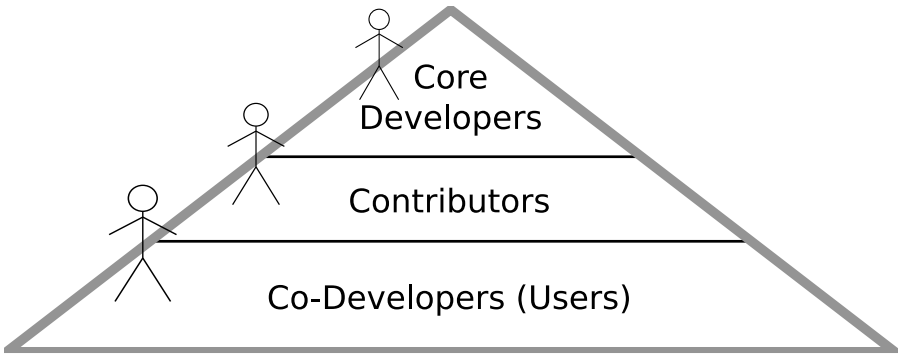

Fig. 2.3. Subroles for developers: *core developer*, *contributor*, *co-developer*

The core developer teams includes the project leader(s) and often times its members are designated as beeing the only ones able to commit code-changes to the project's code base.
Major decisions are made within this team. How these decisions are made however is not unified. While Linus Torvalds has the last word on all Linux kernel modifications other projects like the Apache project rely on a majority consensus in the core developer team.

Another interesting observation is that the field of experienced developers is usually respected so that "new developers tend to focus on areas where the former maintainer is no longer interested in working, or in the development of new architectures and features that have no preexisting claims (frontier building)"[4].

2.4 Software Architecture

As open-source software needs to be extensible and allow for extremely parallel development "one characteristic that is shared by the largest and most successful open-source projects [...] is a software architecture designed to promote anarchic collaboration through extensions while at the same time preserving centralized control over the interfaces." [7]. The Apache web server is a system explicitly designed for extensibility[4]. Linus Torvalds says the open-source development model requires a system as modular as possible, because otherwise you can't easily have people working in parallel[8]. Torvalds himself very carefully keeps controll over the interfaces of the Linux kernel, while many developers can contribute kernel modules.
Another good example for modular design is the Eclipse IDE built on top of the service oriented OSGi framework, which allows developers to contribute pluggable modules.
In general the system architecture of open-source software has to be very modular. This results in a typical open-source two level architecture. The core level usually is as slim as possible and maintained by core developers. The higher level typically consists of modules and extensions from a large number of contributer.

Paulson and Eberlein find that the hypothesis "open-source projects are more modular than closed source projects"[6] is not supported by their arguably not representative study. This might be due to the fact that modularity is not a property that applies to open-source software only, but to any larger scale project.

2.5 Tool Support

In the commercial world a company can afford to buy a couple of seats of an expensive development tool. The open-source community preferably uses free tools to develope free software. This keeps the threshold for new developers low. It has to be emphasized that tools play a very important role in open-source development. They enable distributed work around the globe. Common tools in open-source software development are:

- A project usually maintains a website for software distribution and project presentation.
- Since community feedback is extremely important to developers bug reporting tools (e.g. *Bugzilla*) are used.
- A forum or wiki serves as documentation and knowledge exchange to users and developers.
- Code documentation is automatically generated by *Javadoc* or *Doxygen*.
- Version control and configuration management is typically handled by *CVS* or it's successor *SVN*.
- Communication within the team relies hardly on face-to-face meeting. This might seem less effective but has the advantage that the communication is documented[5] through an internet base communication channel like: mailing lists, forums, IRC, instant messaging or even plain old TODO or STATUS files[4].
- Design and development: Eclipse, Netbeans, Emacs, Vim, ArgoUML, Dia
- Compiling: *GCC* or Sun's *javac* usually in conjunction with common build tools like: *Apache Ant*, *GNU make* or *SCons*.
- Debugging: *GDB* or *Valgrind*
- Automated testing: *JUnit*, *PHPUnit*, *PyUnit*, *NUnit* or *Tinderbox*
- Deployment: Linux Package Management Systems APT, RPM, Gentoo Portage

Obviously the programmers' tasks are quite well supported by this toolchain. A software-engineer or manager might miss support for higher-level project coordination[1] like tools for requirements elicitation, group decision making, task scheduling, process tracking, cost estimation or test suite design. J.E. Robbins explains: "The lack of tools in some of these areas is understandable since open-source projects do not need to meet deadlines or balance budgets. However, better requirements management and testing tools would certainly seem just as useful in open-source work as they are in traditional development."[5]

[4] done so in the Apache project

2.6 Development Processes

In classical development methods processes are designed to produce quality. These processes are targeted to ensure that the costs run within their estimates, that all requirements and deadlines are met and traceability is assured. This is why much time and effort is spent in requirements analysis, design and specification. A common view on the software development process is the V-Model (figure A.1). It describes a series of specification steps from rough to fine grained design before actual coding starts. Once the software is written a series of tests is conducted. During the whole process the artifacts are verified and validated against their predecessors. Ideally the project and the code is well documented and the customer receives a finished product that has no defects and meets all requirements.

In open-source development there are usually no hard deadlines to meet and requirements are not statued beforehand. Therefore working units are not necessarily scheduled ahead of time. Instead developers tackle issues from their personal list of ideas, bug reports or change requests through community feedback as they appear. A lot of the processes that are traditionally targeted to get the requirements and design documents right are therefore obsolete. In the V-Model example this affects the descending specification branch, leaving out a large overhead. Instead the open-source development process concentrates on getting the code-changes done with high quality and low defect rates.

Developers usually state their commitment of working on an issue or a bug by assigning their name to it in some form. If it is a bigger issue a small task force is formed. Within this team communication and planning is rather casual. While planning the courses of actions a self determined deadline for the next release may be imposed at that point. Pretty straightforward coding starts. Documenting the code-changes ideally goes along with coding. Finally a user documentation has to be written; often times in several languages. Depending on the project the new code has to go through a series of code reviews and tests much like in commercial projects before reaching the unstable and somewhat later the stable release. This process[5] is illustrated in figure A.2.

While the waterfall does not even work for most of the commercial projects it wouldn't work for an open-source project at all. The system architecture is designed to enable distributed development. Development takes places iteratively and independently in small increments.

Looking at the artifacts produced in open-source development it becomes obvious that design documents are not part of any contract. Thus specifications usually concentrate on the pure technical necessities like protocols or standards. This is why fewer kinds of artifacts are needed. The two main types of artifacts found in open-source development are code and documentation. While the development process is really code-centric, the extend of documentation covers the whole spectrum from code comments, simple text

[5] Process description's source:[9], [4]

files over auto-generated javadoc or doxygen bundles to multilingual wikis used as knowledge management systems.

Milestones in open-source are usually code-releases. These are marked by an increased version count and published on the project's homepage. It is typical to maintain several releases, at least two: stable and unstable release.

For major releases a dedicated release manager within the core-team signals a heads up to all other developers. He then tags a version in the code repository so other developers can continue working in parallel. Packaging traditionally beeing a tedious task comes at much less expense in open-source projects. There are no CDs to burn, no paid advertising campaigns or what so ever. New releases are simply published on the project's homepage. This process of "release early, release often" coined by Linus Torvalds has a major advantage. Traditionally a bug in a release if a **bad** thing. Early and frequent unstable releases change the way of thinking about this. No longer is a bug in this kind of release a bad thing. Defects are found and fixes quickly[6] because there are "many eyeballs" looking for the problems (Raymond calleds this the "Linus's Law" [3]).

This, of course, is a luxury companies don't have. Neither do they have the human resources to do such extensive testing nor may it be desired to risk their reputation by revealing their product's weaknesses through premature releases. In conclusion open-source development adopts some practices found in commercial projects, e.g. code inspections by core-developers[4], but also has the luxury of an extensive beta tester community, enabling massive parallel testing. As a result Paulson and Eberlein find support for their hypothesis: "open-source projects generally have less defects than closed source projects, as defects are found and fixed rapidly".[6]

3

Examples

There is a vast variety of open-source projects. On *freshmeat.net*[1] for example over 43.000 projects are hosted. On sourceforge.net[2] it is even over 100.000 projects. Of course only a fraction of these projects are very active and widely known. It again shows how important it is to attract users and developers. Here are three examples of very successful and well known projects.

3.1 Netscape Browser

In January 1998 Netscape Communications, Inc announced plans to disclose the source for Netscape Communicator, hoping to gain development power and to deny Microsoft a monopoly on the browser market[3]. At the mean time this was the first large-scale, real-world test for a commercial company to adopt the bazaar style development model. Under the name Mozilla the project was very successful in gaining market share. Yet, the rush of new developers failed to appear and the browser kept being developed mostly by the Netscape employees. A reason for this was that getting started working on the code and compiling from source was quite complicated. A complicated build system, the large and complex code base, few documentation and the need for a proprietary license of the motif library made it cumbersome. This of course broke the rule of providing an easy to run prototype. Furthermore the Netscape license model interfered with the GPL making in impossible to interchange code.
In 1999 Jamie Zawinski, one of the project's principals, even resigned because of his discontent with the management. "Open source," he correctly observed, "is not magic pixie dust." The Netscape browser was one of the first commercial projects to go open-source and a test to the open source development model. Had open-source failed?

[1] `http://freshmeat.net`
[2] `http://sourceforge.net`

Not quite, the crisis was overcome by November 2000 and the nightly releases reached production usability.
In July 2004 the Mozilla Foundation was formally registered as a not-for-profit organisation. Today, the foundation maintains the popular Firefox browser and Thunderbird mail client and several other sub-projects. Mozilla Firefox holds approximately 15 percent of the world wide browser market share beeing runner-up behind Microsoft's Internet Explorer.[3]

3.2 Eclipse Platform

Eclipse is an open-source community focused on developing a universal platform of frameworks and exemplary tools that make it easy and cost-effective to build and deploy software [10].

The Eclipse platform was initially developed by a subsidiary of IBM purchased in 1996. In order to enable a unified toolchains IBM established the Eclipse consortium in 2001 and gave Eclipse to the open-source community under an open-source license. This way the community could deal with the code while the consortium would deal with the commercial relations. But tool vendors were reluctant to contribute to Eclipse while IBM was in control. So IBM relinquished any control.

The Eclipse Foundation was formed in February 2004 as a non-profit organization with its own professional staff and support of over 50 member companies [10]. The purpose of this organisation is to "cultivate an ecosystem of complementary products, capabilities, and services" [11]. Thus, the Eclipse project is not only about creating a state-of-the-art IDE. Eclipse professionally hosts and coordinates many sub-projects that extend the platform. These are some of the most prominent projects:

- Business Intelligent and Reporting Tools (BIRT)
- Eclipse Web Tools Platform (WTP)
- Graphical Editing Framework (GEF)
- Visual Editor (VE) Project
- C/C++ Development Tools (CDT)
- Mylar
- Eclipse Communications Framework (ECF) Project

As the purpose of Eclipse is to foster new projects it is made fairly easy to create a project (enter incubation). Yet, it is very hard to get a project to ship with the IDE. This is because the foundation is very much aware of their "Collective Reputation", meaning that the quality level Eclipse is perceived at depends also on the quality of the sub-projects, and vice versa. A failing

[3] according to `http://marketshare.hitslink.com` "Browser Market Share for May, 2007"

project under the Eclipse banner consequently detracts from that reputation. This is why Eclipse has a well-defined process for identifying and dealing with failures when they occur. These processes are documented as guidelines and checklists and published on their website[4] [11].

As part of the Eclipse Development Process a project is constantly reviewed and guided by mentors. The process defines a Project Lifecycle of project stages (see figure A.3): Pre-proposal, Proposal, Incubation, Mature, Top-Level, Archived. A project can move on to the next stage once it passes the corresponding review.

The projects are managed by an hierarchy named the Project Leadership Chain, consisting of the Project Management Committee (PMC), the project leaders of the Top-Level projects and regular project leaders. Eclipse is a meritocracy.[5] Leadership roles in Eclipse are also merit-based and earned by peer acclaim [11].

Each project has a development team composed of Committers and Contributors. Since Eclipse is an extensible platform with well documented APIs there exists a third interesting role: Adopters. Adopters are plug-in developers loosely affiliated with the project. They can extend the platform with their problem specific tools, which are often times also made available to the public.

To maintain a reliable and accessible development roadmap and to support teams in major design decisions, a set of councils (Requirements, Architecture and Planning) guides the development done by Eclipse Open Source projects.

3.3 Gentoo Linux Distribution

There are many well known Linux distributions of high quality. Gentoo is just one good example for a well documented and coordinated, non-commercial open-source project. The Gentoo Linux Distribution efforts focus on system tool development, packaging and documentation.

System details and installation instructions are systematicly documented for developers and end-users in several languages on the gentoo website[6].
An active community provides help to users in the online forums[7] and the gentoo wikis[8]. Even non-gentooers find valuable information here.

The project targets server as well as desktop systems on 10 different architectures[9]. For each architecture a stable and an unstable release is maintained. Gentoo issues official releases (CDs and installer software) once or twice a year, while software updates through the package management system come in on a daily bases. Furthermore, on gentoo systems every program is compiled

[4] `http://www.eclipse.org/org/documents/`
[5] The more you contribute the more responsibility you will earn.
[6] `http://www.gentoo.org`
[7] `http://forums.gentoo.org`
[8] `http://gentoo-wiki.com`
[9] x86, sparc, amd64, ppc, ppc64, alpha, hppa, mips, ia64, arm

from source, enableing the compiler to optimize the code for the exact target platform. Because of this it is also possible to obtain fresh software releases fast.

As coordination of such a large project with such a diverse code-bases is a non-trivial task, the project maintains a series of tools. The process of compiling and installing the software is made easy by so called *ebuilds*. One of the outstanding tools is the package management system "portage" making it easy to download and manage the ebuilds. *Java-config*, *eselect*, *use*-flags and an intelligent *init* system are other highlights in the gentoo toolchain.

The Gentoo project is organized in 30 sub-projects dealing with package mangement, documentation and infrastructure. Many sub-projects themselves have up to two levels of sub-sub-projects. All projects maintain a dedicated website on gentoo.org. Documentation, guidelines and process descriptions[10] can be found here. Each sub-project is represented by a project leader. Decisions are usually made by majority consensus.

On the management side the gentoo team defines processes in order to ensure the high quality of their work. Gentoo assures process quality through guidelines, checklists and regression tests on the different platforms[11]. A good example of a Gentoo process is how new developers are introduced into the project:

Recruitment starts by either a user himself replying on an opening or an invitation to a very active users. The candidate is then assigned a personal mentor with whom he works with for at least one month. During this time the apprentice has to pass a quiz, showing that he has gained enough knowledge of the gentoo guidelines. After developer status is granted the new team member has to endure a probationary period of 30 days during which his mentor is fully responsible for his actions. This way it is made sure that the newcomers have enough skills to successfully participate in the project.

[10] Example: Retirement Process `http://www.gentoo.org/proj/en/infrastructure/retire-process.xml`

[11] Example for Gentoo guidelines: release process description `http://www.gentoo.org/proj/en/releng/docs/release_guidelines.xml`

A

Figures

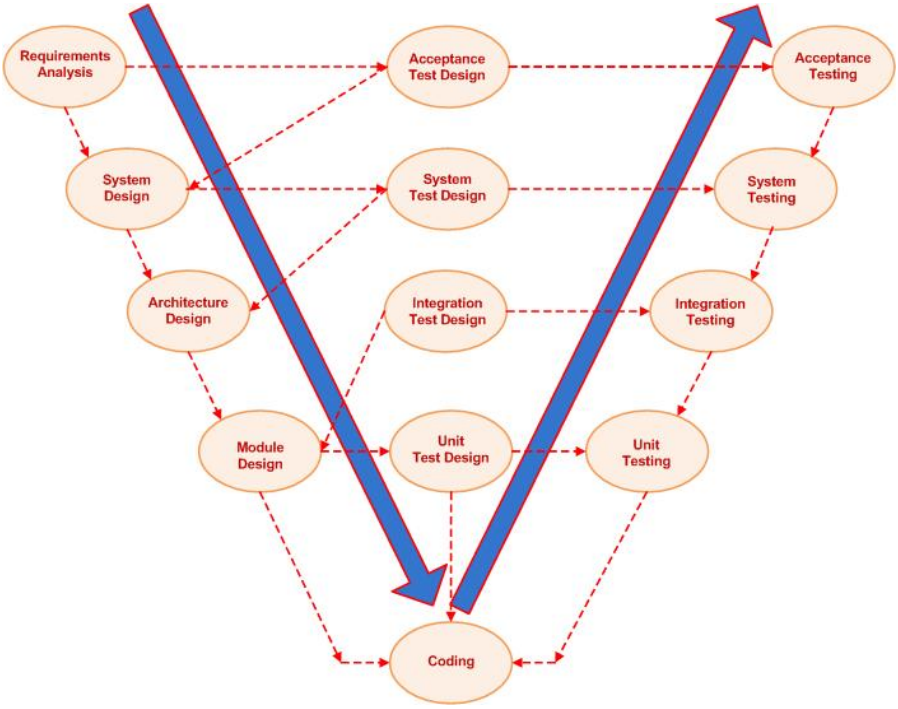

Fig. A.1. V-Model, by Ajith Kumar (public domain) [12]

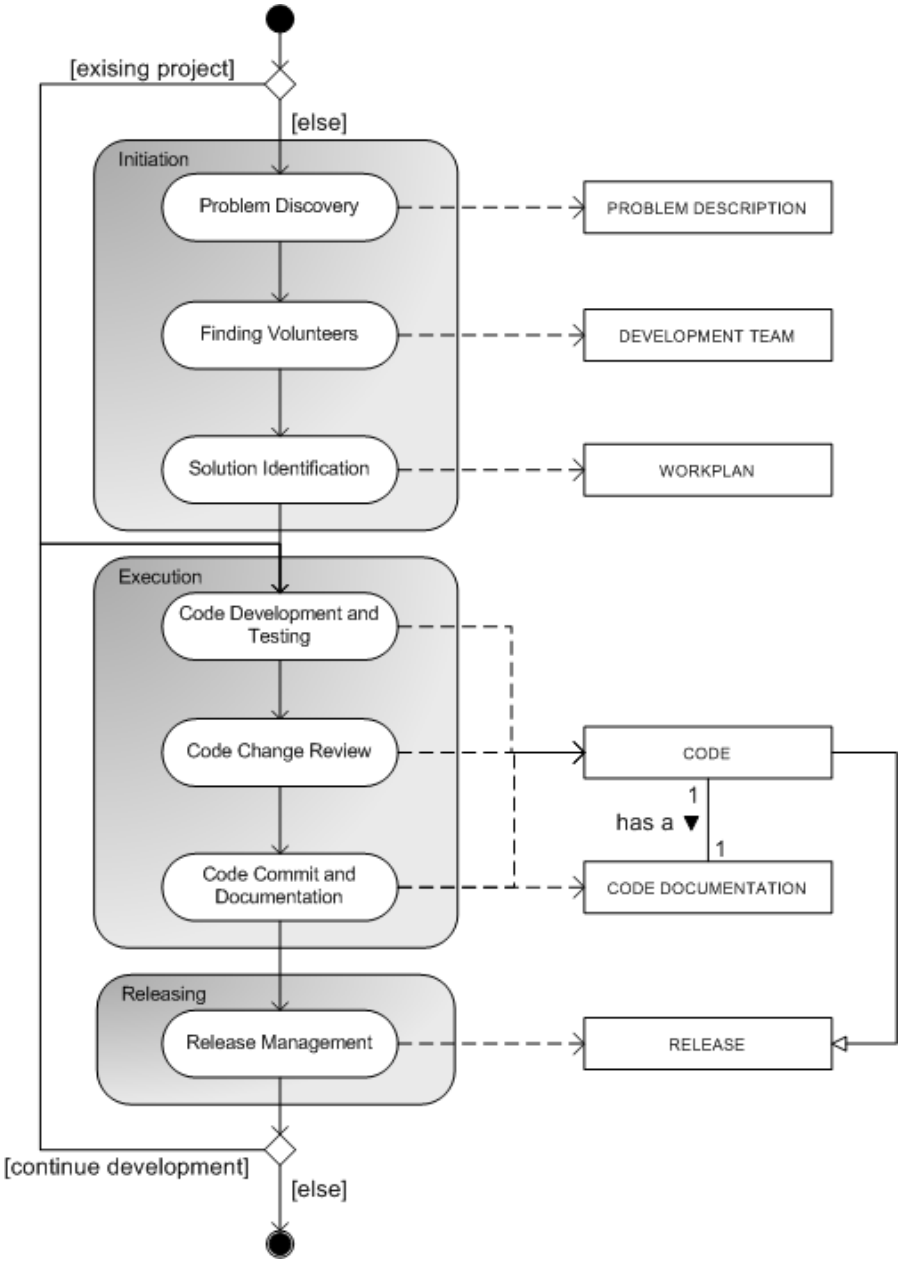

Fig. A.2. Process-data diagram of Open source software development, by M. Abbing (Creative Commons Attribution ShareAlike License version 2.5) [9]

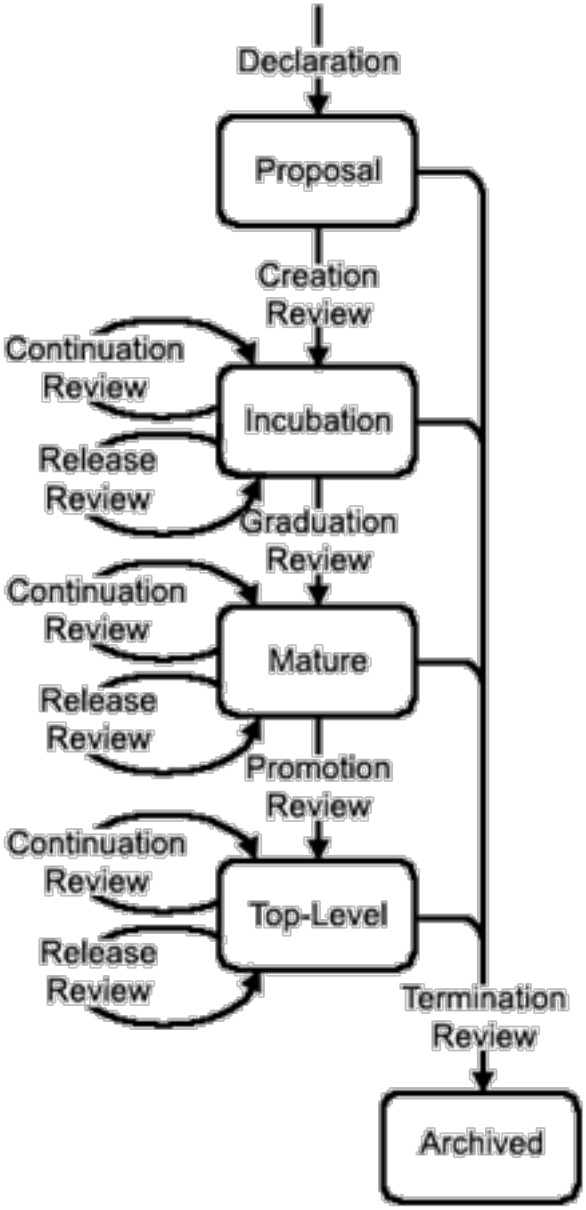

Fig. A.3. Eclipse Development Process - Project Lifecycle [11]

B

Abbreviations

API	Application Programming Interface
APT	Advanced Packaging Tool
CVS	Concurrent Versions System
GCC	GNU Compiler Collection
GDB	GNU Project Debugger
IDE	Integrated Development Environment
IRC	Internet Relay Chat
RPM	Red Hat Package Manager
SVN	Subversion
TCO	Total Cost of Ownership

References

1. Davor Čubranić und K. S. Booth. *Coordinating Open-Source Software Development.* IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (1999).
2. David C. Lawrence. *Internetnews Server: Inside An Open-Source Project.* IEEE Internet Computing **2** (5), 49–52 (1998).
3. Eric Steven Raymond. *The Cathedral and the Bazaar.* (2000).
4. Audris Mockus, Roy T. Fielding und James Herbsleb. *A Case Study of Open Source Software Development: The Apache Server.* ACM (2000).
5. Jason E. Robbins. *Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools.*
6. James W. Paulson und Armin Eberlein. *An Empirical Study of Open-Source and Closed-Source Software Products.* IEEE Transactions on Software Engineering **30** (4) (March 2004).
7. Roy T. Fielding. *Software Architecture in an Open Source World.* ACM **5** (2005).
8. Linus Torvalds. *The Linux Edge.* Open Source: Voices from the Open Source Revolution (1999).
9. *Open source software development method.* `http://en.wikipedia.org/wiki/Open_source_software_development_method` June 2007.
10. *Eclipse project resources - project information for Eclipse developers.* `http://www-128.ibm.com/developerworks/opensource/top-projects/eclipse-s%tarthere.html` June 2007.
11. *Eclipse Development Process.* `http://www.eclipse.org/projects/dev_process/development_process.php` June 2007.
12. *V-Model (software development).* `http://en.wikipedia.org/wiki/V-Model_%28software_development%29` June 2007.