# Open Source Software Development Methodologies

CP628: Open Source / Linux

Lecture: Sasha Ivkovic

Student Name: Zihong Chen

Student No: 2555470

Email: czihong@gmail.com

## Abstract

Open source and free software become more and more popular, because of the development of Internet and the success of some famous free softwares in commerce, such as Apache and Mozilla. Compare to the traditional software development, there is not definite user, function, deadline in open source software. It leads to the open source software projects run in an unstructed model, and it takes a  long time to develop open source software until it release an stable version. Even some open source software projects are cancelled before they released, many external conditions lead to it happened. Some are because of the disinterest of programmers, some are because of lack of supervision and some are because of low quality. So that, the paper focus on researching the open source software project and development methodology.

## Table of Contents

1. Introduction....4
2. Background of Open Source....5
2.1 Open Source Software and Free Software....5
2.2 The Importance Software in Open Source World....5
3. Software Development Methodology....7
3.1 The Life-Cycle Paradigm....7
3.2 The spiral model paradigm....8
3.3 Open source software development model....10
4. Open Source Software Process....12
4.1 Parallel Development....12
4.2 Globally Distributed Development Communities....14
5. Conclusion....17
References....18
Bibliography....19

## 1. Introduction

The purpose of the paper is giving the concept of open source software development methodology to the reader. There are three parts of this paper. Firstly, it gives the audiences and big map of open source software, and the definition of confusing terms, such as open source software and free software. In addition, it mentions the milestone software in the history of open source. And then, the second part introduces the traditional software development methodologies (waterfall model, spiral model) and open source software development methodologies. After that, the third part compare the traditional and the open source software development methodologies. Analyze the advantage of the open source software methodologies in parallel development and large distributed communities.

# 2. Background of Open Source

## *2.1 Open Source Software and Free Software*

Before going forward, it necessary to understand the confusing terms, open source software and free software. The goal of free software movement is make all software free of intellectual property restrictions. Because of the followers of the free software believe that property software obstruct the develop of software and improve technical. “The Open Source movement is working toward most of the same goals, but takes a more pragmatic approach to them. Followers of this movement prefer to base their arguments on the economic and technical merits of making source code freely available, rather than the moral and ethical principles that drive the Free Software Movement”(Hicks, 2000). The Free Software Foundation provides a special license that software need to under these special license, such as GNU GPL, FreeBSD. The Open Source Initiative do not provide their license, but this initiative seeks support for all open source licenses, including the one from Free Software Foundation. In other word, the big companies want to provide their source code of software, but do not want to under Free Software Foundation licesnes, so that they can join into Open Source Initiative with their own licenses, such as Nokia Open Source License.

## *2.2 The Importance Software in Open Source World*

In these days, open source software is gaining media attention, and is widely using in the world. There are some famous open source softwares we have to mention at this paper. The first one is Linux. In 1991, Linus Torvalds modeled his system on Minix, develop a Unix-based operating system, named “Linux”. The next milestone in OOS world is Apache HTTP server. Most of web server is running Apache rather than IIS. Mozilla is other important software. In January 1998, Netscape provided the source code of their browser, Mozilla was the name for the project. In these days, the Mozilla keep fighting with Microsoft IE in the browser market.

# 3. Software Development Methodology

## *3.1 The Life-Cycle Paradigm*

The life-cycle paradigm is a sequential software development model, because of the cascade from one phase to another, this model is also called "waterfall model". There are five phase of the model, requirements definition, system and software design, implementation and unit testing, integration and system testing, operation and maintenance (Sommeriville, L., 2001). This is illustrated in figure 1.

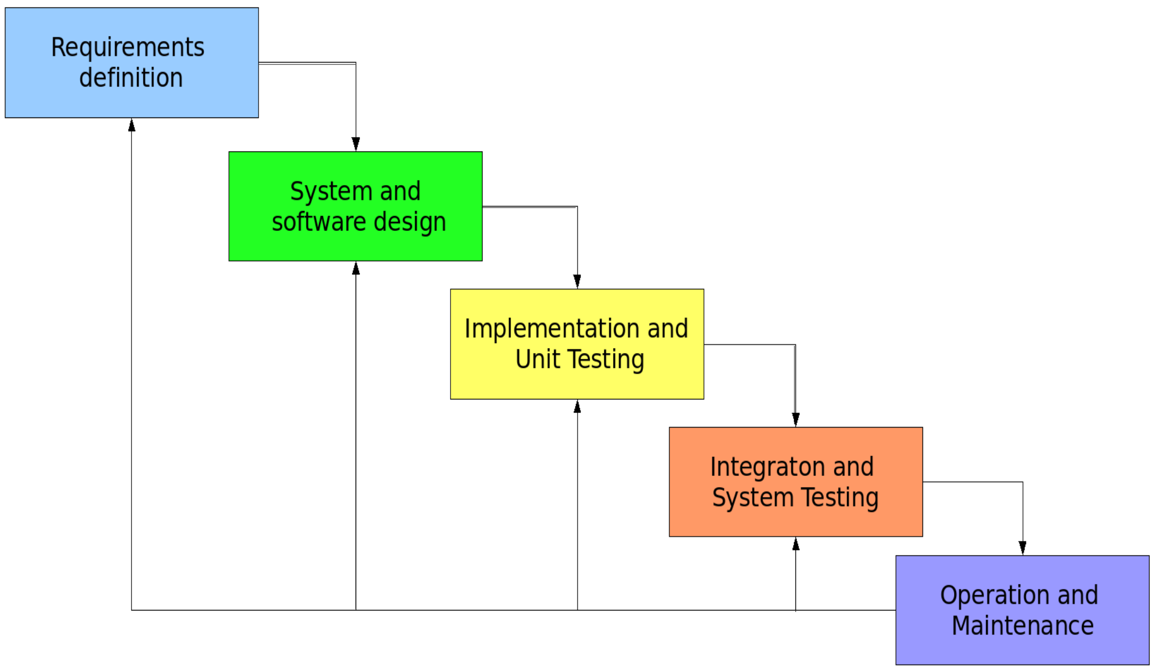

Requirements definition
System and software design
Implementation and Unit Testing
Integraton and System Testing
Operation and Maintenance

Figure 1

**Requirements analysis and definition**: This phase is usually consult with user. Ask for the users' requirements. Then defined the system's services, feature, goal in detail and serve as a system specification.

**System and software design**: This phase involves partitioning the requirement to either hardware or software systems, establishing an overall system architectural, describing the subsystems and their relational.

**Implementation and unit testing**: Transition the software design document into programming code. Unit testing ensure the units meet its specification.

**Integration and system testing**: To integrate each units into a complete system. And then test the complete system meet the requirements of software. Finally, the system is delivered to the customer.

**Operation and maintain**: This phase involves improving the system's feature, find the bug which do not find in the earlier stage, add new function of software as new requirements are discovered.

## *3.2 The spiral model paradigm*

Compare to the lify-cycle model, there are two extra components, risk analysis and prototype. The spiral model is presented as a  circuit which is broken into four primary parts: planning, risk analysis, engineering and customer evaluation. This is illustrated in figure 2.

**Planning**: Interact with customers, ensure the goals and constraints of the software project, and define the alternative.

**Risk analysis**: Analysis the alternative (prototype), identify the risk of alternative. If the risk is too great, throw the alternative.

**Engineering**: Evolution the software product, develop a better production.

**Customer evaluation**: Interact with customers, evaluation the product.

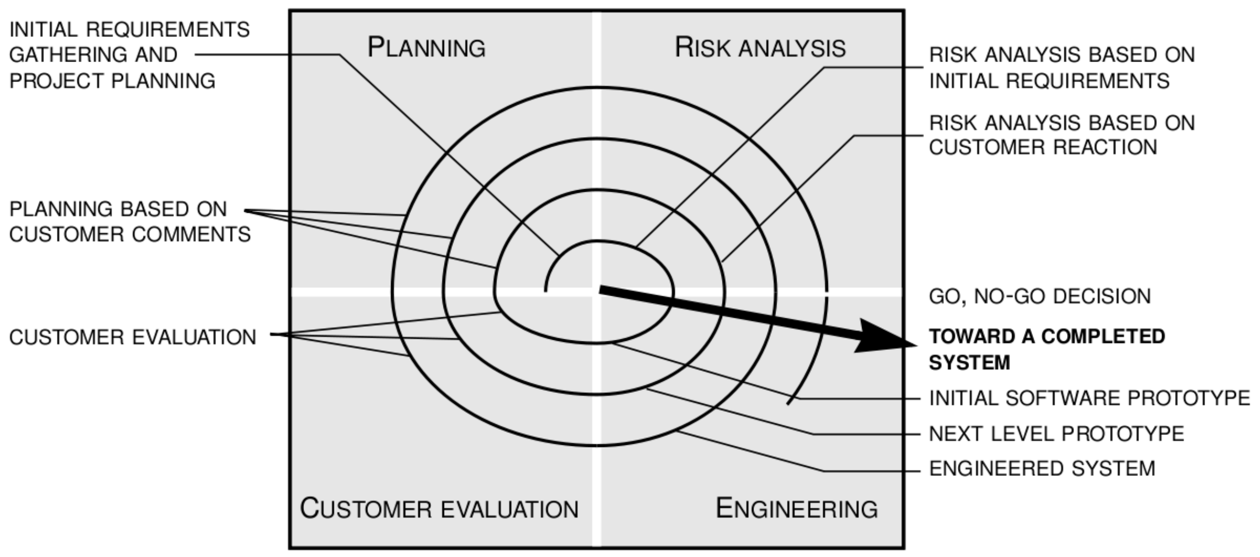

INITIAL REQUIREMENTS GATHERING AND PROJECT PLANNING
PLANNING
RISK ANALYSIS
RISK ANALYSIS BASED ON INITIAL REQUIREMENTS
RISK ANALYSIS BASED ON CUSTOMER REACTION
PLANNING BASED ON CUSTOMER COMMENTS
GO, NO-GO DECISION
CUSTOMER EVALUATION
TOWARD A COMPLETED SYSTEM
INITIAL SOFTWARE PROTOTYPE
NEXT LEVEL PROTOTYPE
ENGINEERED SYSTEM
CUSTOMER EVALUATION
ENGINEERING

Figure 2

In the spiral model, the developed team produces a more integrate and better product after customer evaluation and risk analysis. The prototype of the product maybe throwed, if the risk is too high. In the engineering part, it could use some based software development model, such as life-cycle model and prototype model. The remarkable advantage of the spiral model is, risk analysis. The risk analysis make the developed team of the project find the risk of the product earlier before the delivery of the product.

## *3.3 Open source software development model*

There is not a existing model to develop an open source software. Why? The whole process of develop an open source is more complex than traditional, because of its thousands of talented developers, the large global communities. In open source software, there are no definite requirements of customers, but the user can report any new suggestion or idea. The new suggestion or idea can be treated as customers' requirement.

Several researchers have investigated the software development methodology of OOS. Ballinger et al. (1999) suggest that it follows a very intensive spiral model (Boehm, 1988), albeit without any real risk analysis. This model is illustrate in figure 3. As the figure show, there are no risk analysis. “New suggestion or idea” from users instead of risk analysis. In fact, the real risk of an open source software is not enough talented developer to support it. This maybe occur because of the low attract of the software's feature, the maintainer of the project  lose passion, the project is not active enough.

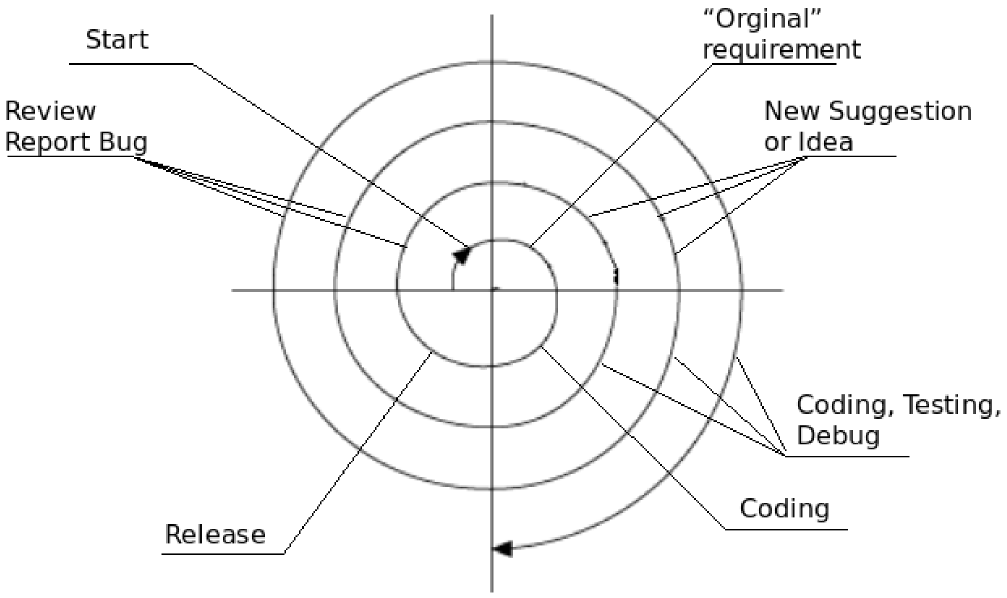

Start
"Orginal" requirement
Review Report Bug
New Suggestion or Idea
Coding, Testing, Debug
Coding
Release

Figure 3

# 4. Open Source Software Process

There is not a standard of OOS process, each team or company use different methods and tools, which depending on the size and other aspect of the software project. Here just brief OOS development in a general way.

## *4.1 Parallel Development*

In software development, parallel development, a large software sperates into small components, and individual or small group work on each separating component. Parallel development means not linear development, each indivaul or small group can keep development their own part of large software product, rather than waiting on each other.

The developers can join in the OOS project at anytime and any phase, whatever initialize phase or testing. So that, the talented developers can contribute to the open source software using their own special and professional skill. In conventional software development process, Brooks(1995) argue that “adding manpower to a late software product makes it later.” In OOS development process, it is a big different to traditional software development. In fact, each developer work in highly cohesive and loosely coupled model, it do not effect to each other. Contrarily, more talented developer to participate in the project, it will speed up the software project.

The other problem of parallel development is, two or more individual and small group may work on the same aspect of the software system (Figure 4). In open source, it is hard to stop this case occur. According to experience of traditional software, it is waste time, cost and human resource, if there are two same teams try to solve the same aspect of software system. In the process of development an open source software, it sounded not a bad thing. Why? “Good Programmers know what to write. Great ones know what to rewrite (and reuse)” (Raymond, 2001 CatB). More than one teams solve the same aspect of the system, in other word, there is not just one proposal of the same problem. So that, we can merge the outstanding part of each proposal, and make an better proposal. It is helpful to improve the quality of the software system. As above said, it waste time, cost and human resource to work on the same component. To the close source project and small project team, this argument is right. To the open source and the huge development community, probably it is not right.

Thus, the parallel development reduces the time of development and improves the quality of software.

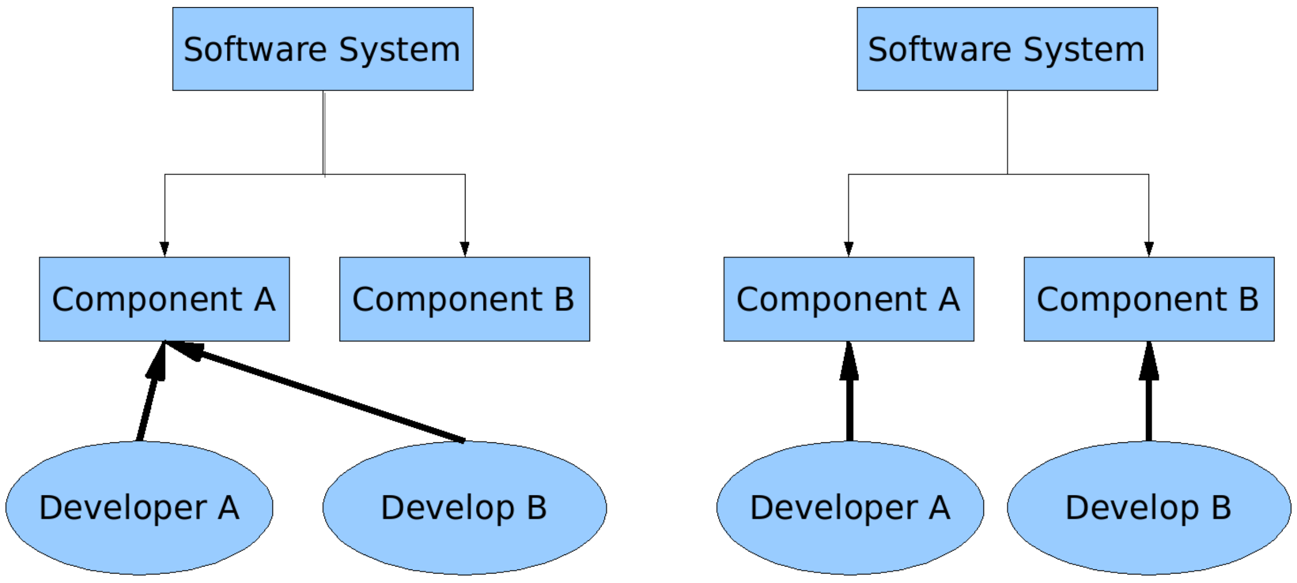


Figure 4

## *4.2 Globally Distributed Development Communities*

As we know, the open source software development communities are setted up for sharing knowledge and improving quality of software in the worldwide scope. For example, the Apache HTTP project, has core developer from Canada, Germany, Italy, the US, and the UK (Fielding 1999), and studies of the Linux kernel development community show activity in at least 28 countries (Hermann et al., 2000). These worldwide communities share the knowledge (source code) and interacts each other via Internet technologies.

In traditional software development, the developed team focuses on meeting the requirements of local users. Such as an Australia team is going to develop a proprietary software, the feature and function of the software is from a Australia-centric perspective. The team struggle to develop an localized software, which meet the habit of Australian. In contrast to proprietary software, at the beginning of develop an open source software, weather or not the leader of project is conscious, its potential users is worldwide. This is because, once the source code is released, everyone can download and modify it. The open source software project is naturally in the user-developer model. The user-developer model is showed in figure 5.

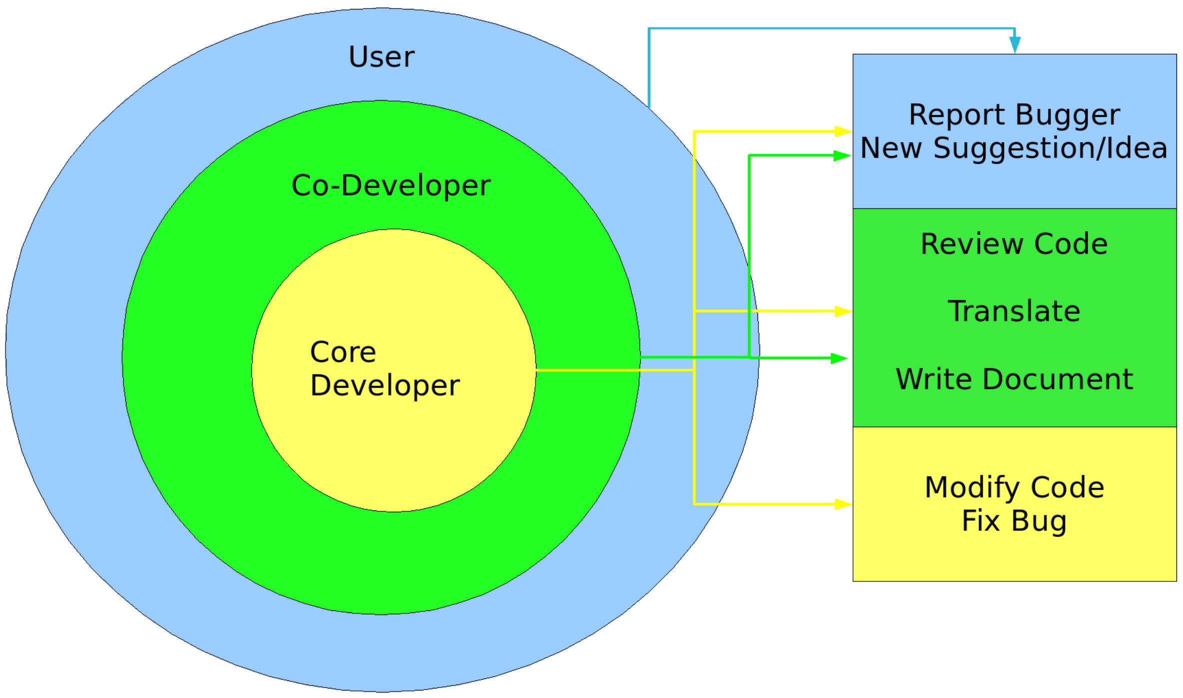
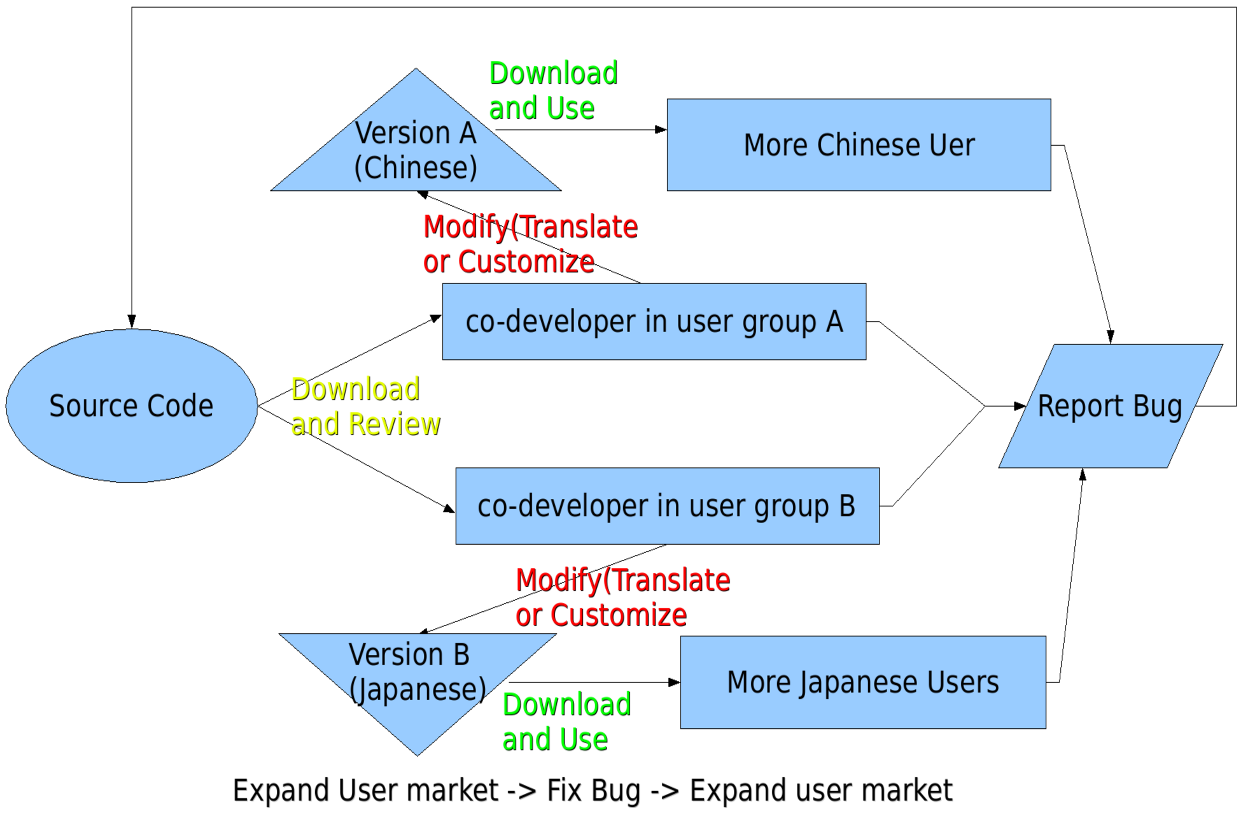


Figure 5

The pure users who have no any programming experience, they can report the bug, report new suggestion and idea. The users who have programming experience, they can report the bug, report new suggestion and idea, review code, translate into other language and write document. To the core developer, they still can do what users and co-developer do, and they have the privilege to modify code and fix bug.

In the user-developer model, the users and co-developers come from different countries. So that, according to different users group, they may have their own habits to use software. The co-developer in the users group can modify the source code, release new version to fit the various requirements of users. In additional, they can translate the software into their mother language, so that the software can be rapidly used in their country, if the software is really excellent. The new version of the software make the "original" software looks more customize, and get more local user attention. The process is showed in figure 6.



Download and Use
Version A (Chinese)
More Chinese Uer
Modify(Translate or Customize
co-developer in user group A
Source Code
Download and Review
Report Bug
co-developer in user group B
Modify(Translate or Customize
Version B (Japanese)
More Japanese Users
Download and Use
Expand User market -> Fix Bug -> Expand user market

Figure 6

Large, globally communities is powerful tools. On the other hand, communities can seen as way to prompt feedback and fix bug, and leads to what Raymond (2001 CatB) has famously called Linux's Law of " Given enough eyeballs, all bugs are shallow."

## 5. Conclusion

The research paper focuses on open source development methodology. In traditional software development, the project can use structual model, for example waterfall model, spiral model. Unlike traditional software development, there is not a standard and structural model to develop open source software. To make the open source software project more structural, it can combine the traditional models, such as using waterfall model in the prototype model, or add the new requirements in the risk analysis  phase of spiral model. This does not mean the open source software is messy. Contrarily, open source development has its own advantage, such as parallel development, large distributed communities, parallel debugging. These advantages are helpful to improve the quality and reduced the time of open source project.

## References

Boehm, B. (1988) “A spiral model of software development and maintenance,” *IEEE Computer*, 21, 5, 61-72.

Bollinger, T., Nelson, R., Self, K.M. and Turnbull, S.J. (1999) “Open-source methods: peering through the clutter,” *IEEE Software*, Jul/Aug.

Brooks, F. (1995) *The Mythical Man-Month, Reading*, MA: Addison-Wesley.

Feller, J. (2002) *Understanding Open Source Software Development*, MA: Addison-Wesley.

Fielding, R.T. (1999) “Shared leadership in the Apache Project,” *Communications of the ACM*, April, 42,4.

Hermann, F. (1999) “Setting up shop: the business of open-source software,” *IEEE Software*, Jan/Feb.

Hicks, A., Lumens, C., Cantrell, D. and Johnson, L. (2000), *Slackware Linux Essentials,* MA: Walnut Creek CDROM

Raymond, E.S. (2001 CatB) “The Cathedral and the Bazaar,” in RayMond, E.S. (2001), *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, Sebastapol, CA: O'Reilly, pp. 19-64.

Sommeriville, L. (2001), *Software Engineer,* MA: Addison-Wesley.

## Bibliography

Boehm, B. (1988) “A spiral model of software development and maintenance,” *IEEE Computer*, 21, 5, 61-72.

Bollinger, T., Nelson, R., Self, K.M. and Turnbull, S.J. (1999) “Open-source methods: peering through the clutter,” *IEEE Software*, Jul/Aug.

Brooks, F. (1995) *The Mythical Man-Month, Reading*, MA: Addison-Wesley.

Feller, J. (2002) *Understanding Open Source Software Development*, MA: Addison-Wesley.

Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. (2005) *Perspectives on Free and Open Source Softwar,* Cambridge: the MIT press.

Fielding, R.T. (1999) “Shared leadership in the Apache Project,” *Communications of the ACM*, April, 42,4.

Hermann, F. (1999) “Setting up shop: the business of open-source software,” *IEEE Software*, Jan/Feb.

Hicks, A., Lumens, C., Cantrell, D. and Johnson, L. (2000), *Slackware Linux Essentials,* MA: Walnut Creek CDROM

Highsmith, C. (2002), *Agile Software Development,* MA: Addison-Wesley.

Raymond, E.S. (2001 CatB) “The Cathedral and the Bazaar,” in RayMond, E.S. (2001), *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*, Sebastapol, CA: O'Reilly, pp. 19-64.

Sommeriville, L. (2001), *Software Engineer,* MA: Addison-Wesley.