EMOS/1: An Evolution Metrics Model for Open Source Software

Yi Wang
Center for Software Engineering
Shanghai Jiao Tong University
Shanghai 200240, P.R.China
+86-21-34202148
yi_wang@sjtu.edu.cn

Defeng Guo
Department of Computer Science and Engineering
Shanghai Jiao Tong University
Shanghai 200240, P.R.China
+86-21-34202146
gudfen@sjtu.edu.cn

ABSTRACT
Open Source Software (OSS) has become the subject of much commercial and academic interest of last. Providing quantitative metrics for OSS evolution has also become an urgent issue. However, most existing studies of software evolution have been performed on systems developed within a single company using traditional management techniques. These metrics models are not suitable for measuring OSS evolution. In this paper, we designed a preliminary evolution metrics model named EMOS/1 which contains a set of new metrics defined for evaluating OSS specially. The most significant novelty of this model is that it takes some properties of Open Source Community (OSC) into consideration. In another word, we measure the evolution of OSS and OSC together. We also provide a lightweight case study on Ubuntu project using EMOS/1. We find out the Open Source Community and its members also play essential role in OSS evolution. We expect our model can bring better understandings and explanations of phenomena in open source development and evolution.

Categories and Subject Descriptors
D.2.8 [**Software Engineering]**: Metrics—*Product Metrics* D.2.9 [**Software Engineering]**: Management—*Programming teams*

General Terms
Measurement, Design, Human Factors.

Keywords
Software Evolution, Metrics, Open Source Software.

1. INTRODUCTION
Backgrounds. Software systems just like the biological systems, both of they evolve. After comparing with biological evolution, Svetinovic *et al.* [19] define the software evolution as following:

Software evolution is change in the essential properties of a software product family over time.

The term *"evolution"* was first applied to software by Lehman and Belady [12] in the 1970s. As organizations become more dependent on software and the pace of both business and IT change increases, the effective management of software evolution becomes more critical to an organization's success. An important contribution to enabling that will be to demonstrate that software evolution and its drivers can be measured, and the impact on software evolution of changes in the drivers can be predicted.

Lehman *et al.* [12][13][14] have built the largest and best known works on research of the evolution of large, long-lived software systems. They are also the pioneers in using metrics [7][10] method to study evolution quantitatively. Till now, most publications in this area describes studies have been performed on systems developed within a single company using traditional *"in house"* development and management techniques.

Challenges. From last decades, more and more software systems are developed by Open Source Community (OSC for short in following section of this paper). Compared with traditional development methods, Open Source Software Development [3] and Open Source Software have lots of new characteristics. For example, the development and maintenance team are much more flexible than traditional development. Godfrey and Tu [9] studied the evolution of Linux by examining 96 kernel version. They found out that although Linux is very large (over two millions lines of code), it has been growing at a "super-linear" rate for several years. Given that the growth of large commercial systems tends to slow down when the systems become larger, Godfrey and Tu's results suggest that OSS systems have a growth rate that is much greater than that of traditional systems. However, their works are mostly done under the framework built for traditional software and can not provide efficient explanations to related issues, such as the reason why the growth of OSS is super-linear. Besides, No literature provides quantitative study on issues such as bugs and bugs fixing, which cost at least 1/3 efforts in the process of evolution and maintenance for Open Source Software.

To better measure and explain the evolution phenomenon in OSS, we designed a novel metrics model containing a set of metrics called EMOS/1 for Open Source Software in this paper. In this paper, we take the changes of Open Source Communities into consideration, for the Open Source Development is a Community-Driven Development in its natural. For OSS development, *John Kenneth Galbraith's "Only the community, reflects the wellbeing, maybe even the survival, of all people."*

tells all the truth about the relationship between OSS and OSC. The communities behind the software are in great differences with these independent software corporations. If we want to get comprehensive understanding of OSS, the OSC should not be ignored.

Design Goals. The ultimate goal of us is to bring better understanding to the process of Open Source Software. To achieve this goal, we decided to design a new evolution metrics model for quantitative measuring the evolution of Open Source Software. Therefore, our model must fulfill following requirements:

Simple: the evolution metrics model should be simple enough for use.

Explicit: the metrics defined should have explicit meaning.

Reasonable: the metrics defined should be reasonable.

Expressive: the metrics should be expressive enough for uncovering some facts about OSS and related communities.

Contributions. The contributions of this paper are the following:

1. We present EMOS/1, which is a new evolution metrics model for quantitative measuring the evolution of Open Source Software.

2. We provide a brief case study using our evolution metrics model to analysis the Ubuntu (An Open Source Linux Distribution).

3. We explain some interesting facts and provide quantitative evidences for some assumptions on open source software's development and evolution.

Paper Organization. This paper is organized as following. We provide a brief introduction on characteristics of Open Source Software Development and Evolution in section 2. Then we present our evolution metrics model (EMOS) in section 3. In section 4, we provide a case study on Ubuntu using our model, and provide a brief discussion. The related work and concluding remarks are included in section 6 and 7 respectively.

2. CHARACTERISTICS OF OPEN SOURCE DEVELOPMENT AND EVOLUTION

The open source development can be dated back to 1984, when Richard Stallman founded the Free Software Foundation. Free Software Foundation is a tax-exempt charity that raises funds for work on the GNU Project. In last 90s, Open source software systems are becoming evermore important. Many large companies are investing in open source projects and many of them are also using such software in their own work. As a consequence, many of these projects are being developed rapidly and are quickly becoming very large. Besides, Open Source Projects received more and more attentions from the researchers. Open Source Software Development is an orthogonal approach to the development of software systems where much of the development activity is openly visible, development artifacts are publicly available over the Web, and generally there is no formal project management regime, budget or schedule. Open Source Software Development is oriented towards the joint development of community of developers and users concomitant with the software system of interest. Compared with traditional software development and maintenance, OOS has some characteristics in following aspects.

2.1 Development Process

For traditional software development, software companies first try to find the needs of their customers, and then use some process models for their development. The development process consist some strict phases, such as *Requirement Engineering, Testing* .etc. However, the developments of OOS do not follow this way. Usually, an OOS project begins with a single developer who has a personal goal or vision. Then he or she makes the code base available to others and development proceeds. Typically, anyone may contribute towards the development of the system and built Open Source Community to provide administration for the project. Ubuntu's development can serve as example for this point. Open Source Software Development commonly does not have formal document for *Reqiurement, Design, Testing*, and so on. For this reason, it is hard to provide a strict *SCHEDULE* for the whole project and provide a precise process model for it.

2.2 Developers

The traditional in-house software development rely on employees in one or several companies, they live in same country except these ones who server for multinational corporations. The open source development is totally different. It has contributors living all over the world. They speak different languages; have different culture and education backgrounds. Therefore, it is impossible for design a unify standard for code, documents. Besides, the OSS development teams are not as stable as corporation-dominated development during the whole life cycle of the software. More people leave, and even more people engage. This directly causes the code quality not so stable.

Meanwhile, most OSD developers have "day jobs" that take up most of their time. This may entail longer development cycles. However, this is a two-edge sword for the software development. It also can bring an advantage since OSD projects are largely immune from "time-to-market" pressures. The software would be released only when the Open Source Community satisfied the system's maturity and stability. This provides compensation to the "trade-off" of the lack of unify code and document standard.

2.3 Evolution and Maintenance

The Evolution and Maintenance of OSS may suffer from some problems. Open Software development encourages active participation of potential users but not pay enough attention on reflection and reorganization. Code quality is maintained largely by "massively parallel debugging" (*i.e.*, many developers each using each other's code) rather than by systematic testing or other planned, prescriptive approaches. Compared with traditional in house development, the open source development lacks planned evolution. For the unstable code quality and standard, the refactorings of existing system is extremely difficult [18].

Fixing existing bugs is also an important topic open source software evolution and maintenance. Compared with traditional software, the OSS is supported by people all over the world. Therefore, the efficiency of bug finding and fixing may be higher. However, the lack of compulsory management also brings some drawbacks in response to urgent circumstances.

3. THE EMOS/1

Based on above OSS characteristics, our evolution metrics model provides metrics related to following aspects: *Module*,

Bugs, Bug Fixing, and Requirements Implementation. Please note that, we do not mean the totally abandon of the evolution metric of traditional software. The metrics in EMOS/1 provided should be treated as supplements of traditional metrics. The metrics can help to disclose more facts on Open Source Software and gain better explanations for phenomena occurring in Open Source Software's Growth and Evolution.

The EMOS/1's most significant change to traditional evolution metrics is that it combines the evolution of Open Source Software with the evolution of Open Source Community. This aims to uncover the fact about open source software evolution. The Open Source Community not only play an essential role in the development process, but equally important in the evolution and maintenance. EMOS/1's results can bring better understanding of OSS' evolution to us.

3.1 Metrics Related to Modules

3.1.1 Modules vs. Source Code

In measuring the evolution of traditional software, computing the change of source code is very important. However, calculating the source code and conducting analysis based on it are less meaning for in quantitative assessment of the OSS evolution. This because:

1. The open source development lacks unify standard for coding and documenting, and the quality of the code may vary a lot.

2. The open source developers' backgrounds vary a lot. It is hard for them to think and solve a problem in same way. For example, in an "in house" software development, all developers use similar data structures, algorithms, and so on, even they do not want to, they have to. In Open Source development, the programmers can finish the job using their own manners. No one will force them to do something and not to do something.

3. OSS sometimes uses the code of other OSS and this borrowed code may account great part of the whole system. For example, all Linux distributions use the code of Linux Kernel and other applications code. In this situation, discussions based on source code seem meaning less.

Based on above discussion, we decided to use Module as the atomic unit in our analysis. The meaning of "Module" varies a lot in different occasions. Generally, it refers the minimum unit assigned to the developers in the scope of Open Source Software, for instance, it refers "package" in Ubuntu[4][5] while it means "component" in Mozilla[2].

3.1.2 Metrics Definitions

The metrics related to modules including three indexes.

1. *M1*: The number of modules in software system at a series of specific moment.

2. *CD*: The number of developers (code contributors) at a series of specific moment.

3. *MC*: The correlation of *M1* and *CD*.

3.2 Metrics Related to Bugs and Bugs Fixing

Fixing bugs is one of the main tasks in the evolution of existing systems. With the change of release version, we can

We define the Metrics Related to Bugs and Bugs Fixing as following:

4. *B1:* The number of bugs in the software system at a series of specific moment.

5. *BF:* The number of Fixed bugs in the software system a series of specific moment.

6. *FBR:* The ratio of fixed bugs account for the existing bugs at a series of specific moment, it is defined in following way:

$$FBR = \frac{Fixed}{Existing} \qquad FBR \in [0, 1]$$

*"*Fixed*" means number of suggested features during one release cycle.

*"*Existing*" means number of implemented features during one release cycle.

7. *AU:* The number of active users (persons who provide bug reports and feedbacks) at a series of specific moment.

8. *BC:* The correlation of *B1* and *CD*.

9. *BA:* The correlation of *B1* and *AU*.

10. *FC:* The correlation of *BF* and *CD*.

11. *FA:* The correlation of *BF* and *AU*.

Among above 8 metrics, the B1, BF, AU can get though survey on the bug documentations and can use a graph to show their changes. The following 4 metrics and *MC* (defined in 3.1.2) can be easily gotten through some statistical tools such as SPSS.

3.3 Requirement Implementation Ratio

In Open Source Software Development, there is no strict Requirement Documentations [17]. Besides, it also does not have a traditional process of *Requirement Analysis*. The Faith of the OSS is "Features are the first". Generally, the Open Source Communities publish the features on the Internet, waiting for somebody take the duty of developing such features. As the develop assignments are not imperative, some features may not be implemented in time. So, we introduce the following index to adjust Feature Implementation Ratio (*FIR*).

$$FIR = \frac{implemented}{suggested} \qquad FIR \in [0, 1]$$

*"*Suggested*" means number of suggested features during one release cycle.

**"implemented"* means number of implemented features during one release cycle.

3.4 Summary

We summarize the metrics we define in this section in the following table.

Table 1.The 12 Metrics defined in this paper.

Categories	**Name**	Meaning
Module Related	*M1*	Number of modules
	CD	Number of developers (code contributors)
	MC	Correlation of *M1* and *CD*
Bugs and	*B1*	Number of bugs

Bugs Fixing Related	*BF*	Number of Fixed bugs
	FBR	Ratio of fixed bugs
	AU	Number of active users
	BC	Correlation of *B1* and *CD*
	BA	Correlation of *B1* and *AU*
	FC	Correlation of *BF* and *CD*
	FA	Correlation of *BF* and *AU*
Others	*FIR*	Feature Implementation Ratio

There are still some factors we do not take into consideration in EMOS/1, for example: issues related the documents. We will add them into our metrics model in following editions to achieve the "evolution" of our model.

4. CASE STUDY

We choose Ubuntu [4] as our research target, and our method based on the release history of Ubuntu. For most of metrics we defined in section 3 rely on some specific time point, we choose the time of every main release of Ubuntu, the releases of Ubuntu are shown in section 4.1. Though Ubuntu community promises regular release, the release of Version 6.06 is late for two months. Therefore, we use data of April 2006 in our discussion.

4.1 Ubuntu and Its Community

Ubuntu is a completely free Linux distribution. It derives from Debian, and is supported by Canonical, which is a software company. For it can get financial support from this company, the Ubuntu community promise the six month regular release. The first official release of Ubuntu was made in October 2004 and was duly named Version 4.10, followed by several other releases. Till now, there are 5 main desktop versions and 4 server versions (The Version 7.04 Beta has released during we write this paper.).

Table 2. Some facts about the Ubuntu

Version No.	Release Date	Server Included?
4.10	October 2004	N
5.04	April 2005	Y
5.10	October 2005	Y
6.06	June 2006*	Y
6.10	October 2006	Y

*Please notice that the period between the release of 5.10 and 6.06 is not 6 months but 8 months.

Ubuntu has a community to drive and manage its development and maintenance. The Ubuntu community consists of people living in all over the world. Ubuntu community takes charge of works such as bugs fixing, localizations, and so on. The Ubuntu community also provides some infrastructure for distributed development and maintenance, such as Bazaar (a distributed version control system), and bug report system.

4.2 Analysis

4.2.1 Metrics Related to Modules

We get the information of Modules from the Launchpad[1] which is a distributed collaborative infrastructure for Ubuntu and other open source software development. Ubuntu mainly grows though a manner of adding new packages. So, we treat packages as the modules we mention before. In Ubuntu community, some members are in charge of auditing the packages and add them to Ubuntu, here developers are refers them.

Table 3. The changes of *CD*.

Version No.	Release Date	*CD* (Number of developers)
4.10	October 2004	13
5.04	April 2005	77
5.10	October 2005	120
*	April 2006	192
6.10	October 2006	280

*We use the number of April 2006 for substitution. That is the reason why we left the fourth Version No. blank.

The following table points out the packages. There are over 17000 packages totally for Ubuntu. However, only a few packages receive the official support. We only consider these packages here.

Table 4. The changes of *M1*

Version No.	Release Date	*M1* (Number of Modules)
4.10	October 2004	269
5.04	April 2005	436
5.10	October 2005	817
*	April 2006	1225
6.10	October 2006	1658

*As we mentioned in table 3.

We use SPSS13.0 and choose Pearson Correlation Coefficients and Two-tailed test of significant for the judgment of correlation. The result is: *(CD: 0.99, BF: 1, Sig.:0.001).* Obviously, these two are significantly correlated at the lever of 0.01.

This result tells us the increases of Modules correlate to the increase of development team obviously. The growth speed of developers is not much slower than the speed of software growth; this can provide part explanations to the *linear (super-linear)* growth of OOS.

4.2.2 Metrics Related to Bugs and Bugs Fixing

Ubuntu bug reports are also tracked in Launchpad. Till March 2007, there are totally 88259 bugs reported and near 30 thousands are confirmed. The numbers of bugs at each release moment are listed in following table. Please note that we just provide these numbers in the precision of thousand and we exclude all redundant reports.

Table 5. The changes of *B1*.

Version No.	Release Date	*B1* (Number of Reported bugs)

4.10	October 2004	0
5.04	April 2005	12000
5.10	October 2005	25000
*	April 2006	44000
6.10	October 2006	70000

*As we mentioned in table 3.

Following figure shows out the change of No. of Reported Bugs. From this figure, we can find the speed of the reported bugs' growth is at lease linear or even much quicker.

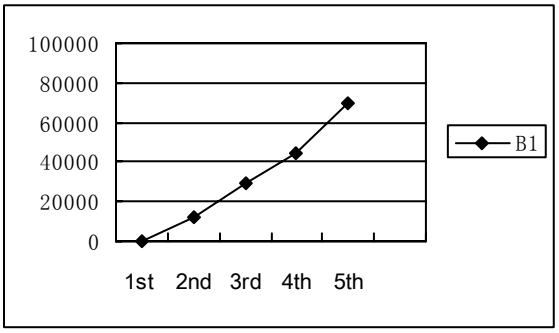

Figure 1. The change of *B1*

We also list the number of fixed bugs (BF), we also choose five time point and the period between them is exact 6 month (common release cycle).

Table 6. The changes of *BF*.

Version No.	Release Date	*BF* (Number of Fixed bugs)
4.10	October 2004	0
5.04	April 2005	2300
5.10	October 2005	7000
*	April 2006	10000
6.10	October 2006	14000

*As we mentioned in table 3.

We also use a figure to illustrate the change of number of fixed bugs.

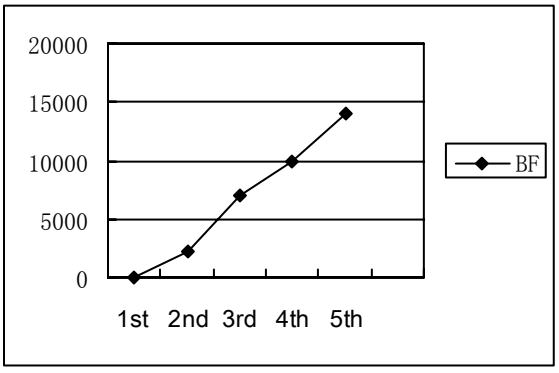

Figure 2. The change of *BF*

The numbers of fixed bugs are increasing slower than the increasing of bugs. We can find the unfixed bugs become more and more. How to find enough contributors for bug fixing is an urgent problem proposed to all members of Ubuntu community. This is a crucial issue for the further development of Ubuntu.

Table 7. The changes of *FBR*.

Version No.	Release Date	*FBR*
4.10	October 2004	0
5.04	April 2005	0.192
5.10	October 2005	0.241
*	April 2006	0.227
6.10	October 2006	0.200

*As we mentioned in table 3.

We can find that although both the B1s and BFs increasing quickly, the FBRs do not vary a lot. This accounts for the fact that fixing bugs need much more professional knowledge and skills. The team of bug fixing can not increase as quickly as users do.

The active users are the persons who provide the bug reports and other related document. We get the number of active users through use a program mining the bug record on the Launchpad according the username of submitter, and find how many people submit bug reports to the Launchpad system. The numbers of Active User are listed in following table.

Table 8. The changes of *AU*.

Version No.	Release Date	*AU*
4.10	October 2004	Not valid
5.04	April 2005	1600
5.10	October 2005	4000
*	April 2006	7000
6.10	October 2006	12000

*As we mentioned in table 3.

The growth of the active users is so quick; this reflects a fact that the Ubuntu is more and more popular. More people begin to use it and try to make contribution to the whole community through submit the bug they encounter during their use.

Based on above descriptive data, we can easily calculate the following 4 metrics in this section. These metrics are summarized in table. We use Pearson Correlation Coefficients and Two-tailed test of significant. Following table show the results we gather by using SPSS13.0

Table 9. The results of *BC, BA, FC, FA*

Metrics	Results
BC	*(CD: 0.99, B1:1, Sig.:0.001:)**
BA	*(AU: 0.99, B1:1, Sig.:0.000)**
FC	*(CD: 0.98, BF: 1, Sig.:0.002)**
FA	*(AU: 0.98, BF:1, Sig.: 0.003)**

* Correlation is significant at the 0.01 lever.

Though above table show us , However, we do not think there are natural relationships between B1 and CD, this may partly because of the Ubuntu is a young project, nearly everything is increasing. Therefore, the high values of and may just by accident.

4.2.3 FIR

The Ubuntu holds two main conferences annually. The main task of these conferences is to discuss which features should be implemented in the next release of Ubuntu, and then write specifications for these features. These specifications are published through the We have examined the historical documents of Ubuntu. By using this information, we calculate the FIR for every release of Ubuntu. We show this in following figure.

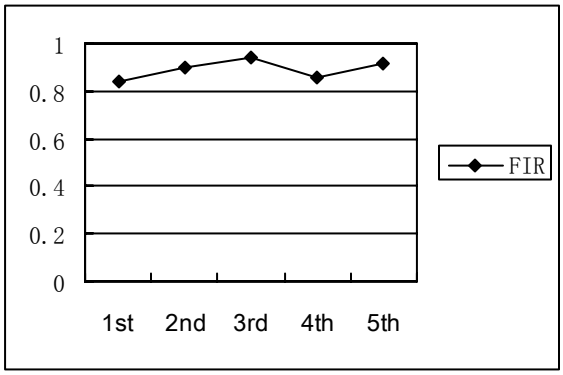

Figure 3. The change of *FIR*

4.3 Discussion

From the data we gather through above analysis, we can find some interesting facts about Ubuntu.

1. The increase of introduced packages shows great coherence with the increase of members.

2. The increase of reported bugs shows great coherence with the increase of Active Users (AU). This fact reflects the importance of Active Users to the Quality of OSS. Besides, Active users also contribute to the project through many other channels such as promoting the use of Ubuntu.

However, the situation of bug fixing is not very positive. For Ubuntu, there are over 10 thousand confirmed bugs which have not been assigned for revising. This many cause some threats to its quality in the long run.

3. The FIR is much higher than the Requirements Implementation Ratio using traditional development method. As we know, in traditional "in house" software development, the percentages of fulfilled requirements are no more than 70% generally. However, as we show in figure , all of these five releases implemented 90% suggested features. This phenomenon may lie on the power of distributed intelligence.

5. RELATED WORK

5.1 Evolution Metrics for traditional software

Lehman *et al.* [12][13][14] have built the foundation of research on the evolution of large, long-lived software systems. The FEAST/1 project, beginning in 1996 and ending in 1998, aimed to construct black- and white-box models of software system evolution, with special attention to feedback phenomena. The results of studying several data series from their industrial collaborators support, or at least do not contradict, the laws of software evolution formulated in the 1970s. Moreover, three new laws have been identified: Continuing Growth, Declining Quality and Feedback System. The project FEAST/2 focuses on control and exploitation of process behavior. In [14], Lehman provided over 50 rules based on the eight laws, for application in process planning, management and implementation, and suggested tools to be developed to support these activities.

Turski [20] suggests that system growth (measured in terms of numbers of source modules and number of modules changed) is usually sub-linear, slowing down as the system gets larger and more complex.

Gall *et al.* [8] examined the evolution of a large telecom switching system both at the system level and within the top-level subsystems. They noted that while the system-level evolution of the system seems to conform to the traditionally observed trend of reduced change rates over time, they noted that the major subsystems may behave quite differently from the system as a whole. In their case study they found that some of the major subsystems exhibited "interesting" evolutionary behaviors, but that these behaviors cancelled each other out when the full system was viewed at the top level. They argue that it is not enough, therefore, to consider evolution from the topmost level; one must also be concerned about the individual parts as well.

Kemerer *et al.* [11] conducted an empirical study of software evolution using the systems of a large United States retailer. The investigation focused on the types of changes, costs and effort to evolve the software. It then analyzed these factors from the point of the view of Lehman's laws. The authors detailed the analysis methods used in the study, such as comparing the time series and sequence analysis of data, as they believed that at this stage clarifying research methods is more important than obtaining a large volume of results.

At Hewlett-Packard, Coleman et al. [6] applied two models of maintainability metrics to the daily development and maintenance processes. HPMAS (Hierarchical Multidimensional Assessment Model) assesses the maintainability of software based on three dimensions: *Control structure*, I*nformation structure* and T*ypography, naming and commenting*. The polynomial regression model uses a polynomial equation to express maintainability in terms of the associated metric attributes. These two models were calibrated against HP engineers' subjective evaluation of 16 software systems. The models were used in monitoring maintenance activities, comparing software systems, and assisting the decision of buy-versus-build.

5.2 Evolution Metrics for OSS

Godfrey and Tu [9] provide first studies on the quantitative metrics for the evolution of Open Source Software. In their case studies, they measured the metrics which is commonly used in evaluating traditional "in house" software evolution. They measured the average size of implementation files (`.c`) and head files (`.h`). Besides, they also mentioned the issues related to the growth of subsystem and core subsystem. They found the growth of OSS in a linear or super-linear pattern. Recently Robles et al. [15] conclude that Lehman's laws, especially the 4*th* law is not well fitted to large scale open source software systems.

Roy and Cordy[16] presents an analysis of the evolution behavior of two small size open source software systems, the *Barcode Library* and *Zlib*, and got a surprising result. That is,

unlike large scale open source software systems, the evolution behavior of these small size open source software systems appears to follow Lehman's laws for software evolution.

Their main efforts are trying to find "why" the open source systems evolve through the "White Box" analysis of the software artificial. The difference between our approach and theirs is that we try to find "why" the open source systems evolve through the combined consideration of the software and the communities behind it. The things we interested are what make the OSS evolve in its unique pattern. We do not treat the OSS as separated system but try to link OSS with the unique environment (the community) where it evolves.

6. CONCLUDING REMARKS

In this paper, we provide a preliminary study on the evolution metrics of Open Source Software. As the Open Source Software Development and Maintenance are unique to traditional "in house" software development, traditional metrics models have some defects in measuring the evolution of OSS. To avoid we suggest a metrics model called EMOS/1 to measure the evolution of Open Source Software. EMOS/1 takes the Open Source Communities which drive the development and maintenance of OSS into consideration. It defines a set of new metrics for quantitatively assessment of the evolution of OSS. The new metrics related to issues such as Modules, Bugs, and Bugs Fixing.

We provide a lightweight case study on the evolution of Ubuntu according its official release cycle. Through our investigation on this popular Linux distribution, we find out that the Open Source Community and its members also play essential role in OSS evolution. This is verified by the results of our case study. The coherence between Ubuntu evolution and the evolution of its community is obvious. The super-linear and linear growth of OSS accounts great for the growth and maturation of the Open Source Community.

We want to conclude this paper with some words from *the J.F.K Yale University graduating class speech in June 11, 1962.*

> *"For the great enemy of truth is very often not the lie -- deliberate, contrived and dishonest -- but the myth -- persistent, persuasive and unrealistic. Too often we hold fast to the clichés of our forebears. We subject all facts to a prefabricated set of interpretations. We enjoy the comfort of opinion without the discomfort of thought."*

We hope our work can bring some new thoughts to the understanding of Open Source Software.

7. FUTURE WORK

Till now, the EMOS/1 is still a preliminary model in measuring the evolution of Open Source Software. It only refers to several factors in the Open Source Software and Communities evolution. We will refine EMOS/1 in future to make it more reasonable and comprehensive. The EMOS itself also needs to evolve; we will provide succeeding versions for EMOS model in future. Besides, we consider that doing case study is an important method in the study of large software system evolution. It can help to verify our assumptions and find the relationship under the cover. The case study in this paper is very rough in a way. We will execute more precise quantitative investigation in future. We hope that our efforts will encourage further investigation into the evolution of OSD software systems, as well as comparisons with systems developed mainly using traditional *"in house"* approaches.

8. ACKNOWLEDGMENTS

The authors want to show their appreciations to Miss Fan Li for her suggestions in statistical models. We also want to show our thanks to the Ubuntu community's help for some historical data. The first author also wants to thank Ms Bei Fan from Leiden University, Netherlands.

9. REFERENCES

[1] Launchpad: http://www.Launchpad.net/ubuntu

[2] Mozilla: http://www.mozilla.org

[3] Open Source: http://www.opensource.org

[4] The Ubuntu Project: http:// www. ubuntu.com

[5] Benjamin, M.H. et al. *"The Official Ubuntu Book"*, Prentice Hall, 2006.

[6] Coleman, D.M. Lowther, B. Oman, P.W. and Ash, D. "Using metrics to evaluate software system maintainability", *IEEE Comp.*, vol. 27, no. 8, pp. 44-49, 1994.

[7] Fenton N.E. and Pfleeger, S.L. *Software metrics: a rigorous and practical approach*, 2nd ed. Boston, MA. PWS Publishing Company, 1997.

[8] Gall, H. Jazayeri, M. Kloesch, R. and Trausmuth, G. Software evolution observations based on product release history. In *Proc. of the 1997 Intl. Conf. on Software Maintenance (ICSM'97)*, Bari, Italy, Oct 1997.

[9] Godfrey, M. W. and Tu, Q. Evolution in open source software: A case study. In *Proc. of 2000 Intl. Conference on Software Maintenance (ICSM 2000)*, San Jose, California, October 2000.

[10] Ji, H. *A set of metrics for software evolution*. University of Reading, Department of Computer Science Technical Report RUCS/2000/TR/013/A, 2000.

[11] Kemerer, C. F. and Slaughter, S. An empirical approach to studying software evolution. *IEEE Transaction. on Software Engineering*, 25(4), July/August 1999.

[12] Lehman, M. M. and Belady, L. A.. *Program Evolution: Processes of Software Change*. Academic Press, 1985.

[13] Lehman, M. M. Perry, D. E. and Ramil, J. F. Implications of evolution metrics on software maintenance. In *Proc. of the 1998 Intl. Conf. on Software Maintenance (ICSM'98)*, Bethesda, Maryland, Nov 1998.

[14] Lehman, M. M. Ramil, J. F. Wernick, P. D. Perry, D. E. and Turski, W. M. Metrics and laws of software evolution—the nineties view. In *Proc. of the Fourth Intl. Software Metrics Symposium (Metrics'97)*, Albuquerque, NM, 1997.

[15] Robles, G., Amor, J. J., Gonzalez-Barahona, J. M., Herraiz, I. Evolution and Growth in Large Libre Software Projects. In Proc of the International Workshop on Principles of Software Evolution(IWPSE05), pp.165-174, Lisbon, Portugal. 2005.

[16] Roy, C.K. and Cordy, J.R. "Evaluating the Evolution of Small Scale Open Source Software Systems", *Special issue on CIC 2006, 15th International Conference on Computing, Research in Computing Science* 23, pp. 123-136 Mexico City, Nov. 2006.

[17] Scacchi, W. Understanding the Requirements for Developing Open Source Software Systems *IEE Proceedings--Software*, 149(1), pp24-39, February 2002.

[18] Stamelos, I. Angelis, L. Oikonomou, A. Bleris, G. L. Code Quality Analysis in software development. *Information System Journal*. pp43-60 Dec 2002.

[19] Svetinovic, D. and Godfrey, M.M. Software and Biological Evolution: Some Common Principles, Mechanisms, and a Definition. *In Proc. of the 8th International Workshop on Principles of Software Evolution (IWPSE05)*, Lisbon, Portugal. 2005

[20] Turski, W. M. Reference model for smooth growth of software systems. *IEEE Trans. on Software Engineering*, 22(8), Aug 1996