Linux and Lehman - Literature Review of Open Source Evolution Analysis

Chris J Arges
University of Texas
Austin,Texas,USA
christopherarges@gmail.com

ABSTRACT

Analysis of Linux kernel source code evolution reveals how certain long-term open source projects grow. Using metrics such as lines of code and complexity, Lehman's software evolution laws can be validated or refuted using empirical evidence. This is important because many of these laws were formulated using only closed-source development models. Studying open source development as well improves the model making it applicable to a wider range of projects. This paper reviews existing studies and compares their methods and results using a general framework. By studying and analyzing a wide range of software development methods including open source, software evolution can be better understood resulting in more efficient and consistent development.

Categories and Subject Descriptors
D.2 [**Software**]: Software Engineering

General Terms
Reliability, Design

Keywords
Software Evolution,Open Source

1. INTRODUCTION

To understand the purpose of these papers and review them a few things must be considered. First, motivation of why these types of studies are important. Second, Lehman's laws must be understood as a basis of these studies. In addition, open source development and the Linux kernel are explained. Finally the metrics used are described to understand how the papers measured evolution empirically. Finally, a general framework is presented to provide a method to compare these papers.

1.1 Motivation

Understanding software evolution improves the ability to create processes for effective and consistent software development. E-type applications are embedded into their own domains, which results in a long-term project where there is no definite closure to the project. These programs can be thought of as feedback systems where the program and environment evolve simultaneously. [2] These E-type software projects continually evolve and change over time can either flourish or decay. By analyzing empirical metrics of existing software projects laws can be developed that track these phenomena. In turn, effective processes can ensure a greater chance of success for the project.

1.2 Lehman's Laws

Lehman's laws describe how E-type software projects evolve with respect to release numbers. The original study Lehman conducted studied IBM software development of OS/360 in the 1970s. After this study, additional papers were written which expanded and amended this set of laws. Table 1 shows a summary of Lehman's laws and a brief description of each law. For this particular review, subsets of these laws were examined by the papers studied.

For more in depth explanation of what these laws mean [6] is a good resource.

1.3 Open Source and Linux Development

For the particular papers that were reviewed, only papers examining the Linux Kernel were selected. The Linux kernel is an operating system based on the design of Unix, started by Linus Torvalds. It runs on many different platforms and architectures and has contributors across the world. [1] The first Linux kernel was released in 1991 and still actively developed. Since this is an open source project with many source releases being archived, much data is completely open and available to study and analyze. Since Lehman's studies mainly concerned closed-source software projects, the addition of this completely open trove of data using an alternative development method is a prime place to validate these laws.

1.4 Metrics

The papers that were reviewed measured lines of code and complexity for releases for the Linux kernel. Lines of code can be measured in many ways. SLOC refers to Source Lines of Code which includes non-commented code, whereas other measurements may only count non-blank lines. [5] This metric is used throughout the papers reviewed, but each paper has their own approach to counting. Complexity can be calculated using techniques such as Cyclomatic complexity or determining coupling of code modules. [5] For each paper reviewed an overview of what metrics used is useful in determining the differences in the study and perhaps explain variations in the results obtained.

1.5 Evaluation Framework

After reading the selected papers a general pattern emerged in which these papers could be compared. In all instances Lehman's Laws were used as the basis of evaluation, however various subsets were actually examined. The scope of

No	Year	Brief Name	Law
I	1974	Continuing Change	E-type systems must be continually adapted else they become progressively less satisfactory.
II	1974	Increasing Complexity	As an E-type system evolves its complexity increases unless work is done to maintain or reduce it.
III	1974	Self Regulation	E-type system evolution process is self regulating with distribution of product and process measures close to normal.
IV	1980	Conservation of Organizational Stability (invariant work rate)	The average effective global activity rate in an evolving E-type system is invariant over product lifetime.
V	1980	Conservation of Familiarity	As an E-type system evolves all associated with it, the average incremental growth remains invariant as the system evolves.
VI	1980	Continuing Growth	The functional content of E-type systems must be continually increased to maintain user satisfaction over their lifetime.
VII	1996	Declining Quality	The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
VIII	1996	Feedback System	E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.

Table 1: Laws of Software Evolution (Summary) [7]

the study varied not only temporally (only a range of versions were used), but also could have included other projects than just the Linux kernel. The independent variable was essentially if the study used release numbers or release dates when evaluating metrics. The dependent variable was the actual metrics used in the paper to support the conclusions about Lehman's laws reached. Finally these conclusions are presented by showing if a particular law was supported or refuted.

1.5.1 Lehman's Laws Used

First the laws that are actually examined in the paper are determined. While some have a greater scope than others, the laws that are most commonly studied can be compared. In this section the importance of why a particular law was chosen is also explored.

1.5.2 Scope of Study

The papers chosen for review were chosen to be fairly recent when possible. Because Linux kernel development has occurred since 1991, there have been many changes and new data points happening all the time. All studies were obviously limited temporally (presuming no time machines were available), thus only certain Linux releases were considered. In addition some may have used more of the source tree, while omitting sub-directories such as 'drivers' from their analysis. Some studies only looked at stable releases, while others at both development and stable releases.

1.5.3 Independent Variable

How the particular independent variable used in comparing the metrics over time varied in the papers. Some choose release version numbers as the X-axis point in their comparisons, while others used the release date. One of the papers [10] was mainly concerned with this distinction, and showed that different conclusions could be reached depending on this particular criteria.

1.5.4 Dependent Variables (Metrics Used)

Various metrics were examined in the papers. The most common one was lines of source code (SLOC), however the definition of a line of source code varied. Complexity was measured in various ways including cyclomatic complexity as well as coupling in modules.

1.5.5 Conclusion reached

Using these metrics and the collected data, plots were generated to support or refute the chosen Lehman's laws in that paper. It is interesting how different conclusions could be reached depending on the selection of the above criteria, thus comparing these conclusions shows either flaws in the methods used, or a lack of a proper framework for evaluation.

2. LITERATURE REVIEW

2.1 The Linux Kernel as a Case Study in Software Evolution [4]

2.1.1 Summary

This paper evaluates Lehman's laws on just the Linux kernel by examining complexity and lines of code to explain all of

Lehman's laws. It focuses on the Linux kernel only and within a specific range. There is much depth and breadth in examining all of Lehman's laws; however some laws are not as easy to evaluate as shown in the paper. In addition it examines other metrics such as number of system calls and average complexity of functions to examine other laws. Overall it validates much of Lehman's laws when possible; however arguments for some laws are not very strong.

2.1.2 Evaluation

This paper examines all eight of Lehman's laws. While some are more directly related, others are less related and more difficult to make particularly strong conclusions.

The scope of this study includes just the Linux kernel. The paper states that it "use[s] 810 version of the Linux kernel, released over a period of 14 years." [4] This translates to a range of development and stable releases from March 1994 to August 2008.

Release date was used for the independent variable in any graphs showing a particular metric versus time.

This study included many metrics in evaluating the Linux kernel depending on the particular law being examined. This includes counting lines of code, system calls, configuration options, and functions. In addition cyclomatic complexity was examined for all files and per function. In addition Oman's maintainability index was used. Number of files added, deleted, grown, shrunk was also used. Releases per month was examined, intervals between releases, and release dates verses sequence numbers. In using all of these metrics a very in depth study was performed. The appendix in the paper reviews the software metrics used an how they were calculated.

The paper exhaustively goes through each law and applies various metrics to support or refute that particular law.

The first law (continuing change), is proven in that the architecture and driver parts of the code-base show continual change in supporting new platforms. This was shown by plotting the growth of source code files in all directory versus just architecture and driver directory against release date. The plot shows platform support is growing with the overall code, and thus is continually changing.

The second law (increasing complexity) has been examined using complexity measures and also determining the complexity perfection and how that changed. This showed that overall complexity is growing in the code, but the average complexity per function is decreasing. Thus, there is clear effort in reducing complexity per-function and this law is followed.

The third law (self regulation) was shown by examining growth rates of code against release dates. It can be shown with this data that large changes are followed by periods of less changes, in another words growth and stabilization alternate. Thus indirectly, self-regulation is shown using these observations.

The fourth law (conservation of organization stability) was shown by examining the number of modifications made to the kernel over release dates as well as release intervals. The author makes the distinction that the "work" performed on the actual project really needs to be measured somehow. With Linux kernel code changes, the amount of work per change in code isn't a linear relationship; thus this is a difficult metric to examine. Intuitively, the paper makes the argument that this law is false for Linux. Since the number of people working on the code base keeps growing as well as the code size its-self. However, the paper then goes on to consider the interpretation that rate corresponds to the fraction of the code-base that is changes per release date. If this is examined then it can be shown that the changes per release date are invariant but with a very high variance that may or may not mean this is a conclusive result.

The fifth law (conservation of familiarity) can be shown as stable minor versions have little functionality changes, while changes between successive stable versions are quite large. Thus this is inconclusive as it depends on what is compared.

The sixth law (continuing growth) can be shown by examining metrics of lines of code and features. Many other studies have already shown growth in terms of lines of code and it can be shown that there is super-linear growth up to version 2.5. After this the growth seems to be closer to linear. To measure growth of features, the number of system calls and configuration options were calculated. Growth in system calls is slowing down greatly, while the configuration options are still increasing quickly. All of these data show that indeed there is continuing growth.

The seventh law (declining quality unless prevented) has been shown by the increasing usefulness to user while showing improvement in maintainability metrics. Thus quality has been maintained because there is conscious effort to do so. This is another difficult thing to observe as the definition of "quality" is hard to quantify. Thus Oman's maintainability index is used to show measured quality versus release date. This shows increasing quality. In addition anecdotal evidence suggests that quality is good because more and more people are using Linux.

The eighth law (feedback system) was shown by using anecdotal evidence of the open source development model.

Overall laws 1, 2, and 6 seemed to be more conclusively tested and proven, while the other laws involved some anecdotal evidence and/or stretching of definitions.

2.2 Evolution and growth in large libre software projects [8]

2.2.1 Summary

This paper evaluates Lehman's Laws for the Linux Kernel and other large Open Source projects as well. It motivates the need for this study using 'libre' or open source software by mentioning the lack of studies evaluating Lehman's laws. Previous research highlights a particularly relevant paper by Godfrey and Tu [3], which studied the Linux kernel evolution in 2000. This paper showed that the Linux kernel showed a super-linear growth rate in lines of code and that the fourth Lehman's law did not apply. This particular law states that development is constant and independent of the resources devoted to it. The rest of the paper is devoted to determining if open source E-type software violates Lehman's fourth law and compares its results with Godfrey and Tu's study. The conclusion showed that indeed the Linux kernel showed super-linear growth as well as some other open source projects. However, many open source projects exhibited just linear growth and supported Lehman's fourth law. The paper hypothesis that this extraordinary growth may be a result of the self-organizing structure of open-source projects, and that further studies are needed.

2.2.2 Evaluation

This particular paper only evaluated Lehman's fourth law which is the conservation of organizational stability.

The scope of the paper encompasses the following projects: the Linux kernel, {Net,Open,Free} BSD, GNOME, KDE, Apache, Mono as well as other open source projects. The time frame used in this study used as much data as was possible at the time of writing. The entire system was used, as well as splitting the system into its various subsystems. In addition both development and stable releases were considered, and care was taken to understand the context of these releases.

The independent variable used in these studies was release date rather than version number. This makes sense since releases and their corresponding version numbers aren't consistent intervals of time.

Metrics used in the study were source lines of code only. A tool called SLOCCount was used to compute the SLOC. For this program SLOC is defined as "a line that finishes in a mark of new line or a mark of end of file, and that contains at least a character that is not a blank space nor comment." [8]

The conclusion reached was that the Linux kernel does exhibit super-linear growth and thus violates Lehman's fourth law. In particular devices drivers and inclusion of architecture dependent code resulted in large chunks of added code. Other projects exhibited linear-growth only, and the paper stated that super-linearity only exists in some cases.

2.3 Impact of release intervals on empirical research into software evolution, with application to the maintainability of Linux [10]

2.3.1 Summary

This study examined a previous paper [9] that concluded that common coupling was growing exponentially and thus Linux needed to be re-factored. This paper refutes this argument by showing the importance of using release dates versus release numbers. Thus we use the basis of Lehman's second law to show increasing complexity; however the difference is that using an independent variable of release date versus release number will result in completely different predictions.

2.3.2 Evaluation

More indirectly Lehman's second law is shown, however an actual comparison was not accomplished.

The scope of this paper is limited to Linux kernels from 1996 until 2008 which covers 2.0 until 2.6. An observation is that the authors of this paper referred to the 2.X kernels as 2.0X series; however this is incorrect. [1] For other sections of the paper only stable kernels were considered with the omission of the 2.6 kernel. It wasn't clear if the authors believed this to be a development version or not. This paper was published in 2008 which means the 2.6 series data was indeed available.

The independent variable used in this study was release date. Specifically, this paper looked in depth into this phenomena. It concluded that since open source development doesn't usually have a strict interval between releases that release numbers may be misleading in drawing conclusions using it as an independent variable.

Metrics used in this study were counts of common coupling instances to determine the complexity. In addition the minor release version number was plotted against release date to show the progression of stable and development versions for 2.0 through 2.5 kernels. In additional the average NCNB (Non commented, non blank lines of code) was counted to determine the growth in size of the kernel.

This paper concludes that different results can be observed if release date is used versus release number. Specifically the study in [9] showed Linux needed re-factoring in 2002, otherwise it would become unmaintainable. However, the Linux kernel is still actively being maintained with no issues, and in addition this paper exposes the flaws with the previous study. Thus this paper is validated in this regard.

3. CONCLUSION

3.1 Comparison of Studies

All three of these papers looked at the Linux kernel and how the code is evolving. While many aspects changed, there was some commonality in these studies as expressed by the evaluation framework given at the beginning of the paper. All papers provided different insight into the evolution of the Linux kernel.

"Impact of release intervals on empirical research into software evolution" [10], with application to the maintainability of Linux showed that release date needs to be used as the independent variable in any analysis of open source software. This intuitively makes sense because release cycles are not as rigid as in a completely closed-source operation. This study also looked at its own set of data which indirectly validated Lehman's second law of increasing complexity. In fact the complexity was increasing but there are instances of active effort to reduce it, resulting in a maintainable long term project. This study was done in 2008, while the 2.6 kernel had been out for a while, it would have been nice to see some data corresponding to this release as well. In addition more metrics should have been evaluated to measure complexity versus release date to further support this study.

"Evolution and growth in large libre software projects" [8] concluded that the Linux kernel exhibited super-linear growth and therefore contradicted Lehman's fourth law. This paper assumes that "growth" increases in source lines of code, and equates to "work" being done. This is an elusive thing to evaluate since calculating the person-hours of work that has gone into the Linux kernel is nearly impossible. "The Linux Kernel as a Case Study in Software Evolution" [4] argues that by defining "growth" and "work" differently, by looking at changes in the code one could conclude near-linear growth is occurring. However, [4] also says that intuitively this law is false since more developers are working on the project, and yet more and more features and platforms are being supported as quickly. This could be because much of the code is orthogonal, and thus it needs to be evaluated by subsection. In addition, this paper could have used additional metrics to evaluate growth. It would have also been nice to see other laws examined as well, because much of this data could also help argue for or against other laws as well.

The most robust study was "The Linux Kernel as a Case Study in Software Evolution." [4] This study evaluated all of Lehman's rules against a fairly large Linux release data-set. It concluded that many of the rules were indeed valid; however it seemed that some results were inconclusive or needed to be stretched a bit to actually make a comparison. Overall, I think this was the most complete paper and included many metrics and data to back up many of its assertions.

3.2 Future Work

Following Lehman's work and expanding these studies into the world of open source could yield more ideas for future work. While some papers seemed more complete by analyzing many open source projects, others restricted their study to just the Linux kernel. I think it would be interesting to also compare and contrast the various development models used and how those projects in aggregate follow these particular laws. Another factor that seemed to affect the studies was how releases were handled by each particular project. Commercial software has more of a unilateral direction where design, development, and testing occur in a controlled environment. Open source software has a distributed structure where developers start a project to "scratch an itch". These projects "release often", and testing is done by users and consumers of the software. Thus the feedback system model should be expanded and updated to account for this different model.

Many of these papers continue to prove or refute Lehman's laws and continue this thread of study which has tracked over thirty years of software development history. While I find these particular laws interesting in their own right, I believe future studies should seek out new patterns without just constraining scope to these particular laws. It seems that many of these studies had to stretch interpretations and definitions to have something conclusive in examining a particular law. For instance, the fourth Law which examines the work invariance is extremely difficult to measure in the open-source world. One cannot measure the amount of time all Linux kernel contributors spend writing any particular line of code. Thus methods such as lines of code, or changes in code are examined. While some authors acknowledged this difficulty, others shoe-horned in a particular metric to support or refute this law.

It also seemed that metrics could be gathered slightly differently, or interpretations of the laws could be varied such that a conclusion was less objective and more dependent on too many small variables. Ideally, a bias free method of examining software development data in all domains of development over large periods of time could be used to create new models of development.

With the advent of more ubiquitous data-mining techniques, large amounts of data could be stored, retained and updated to create a more dynamic, accurate, and conclusive model. These data troves could contain a particular company's own development data, and in addition could be combined with data from academia as well as the trove of data available for open-source projects.

By using more complete data sets, general rules could be determined that would track a wider range of projects. These rules could build on Lehman's existing work and perhaps help to either prove further these laws or create new laws. In addition continually updating these models with new information would refine these rules and further improve software engineering processes.

3.3 Final Thoughts

The impact of these papers shows that open source development presents a unique model for the creation of E-type software projects. The data for these projects are widely available and easy to obtain. Thus, larger scale analysis can be performed to eek a greater understanding of patterns in this system. Lehman's law provide great insight and a history of experiences, yet its unclear if these apply to this new paradigm of development. Overall these laws can be applied to many software development processes given more coverage into additional domains of development. This research is valuable in that better processes can be understood and used to make software engineering more efficient, consistent, and stable.

4. REFERENCES

[1] The linux kernel archives.

[2] S. Cook, R. Harrison, M. Lehman, and P. Wernick. Evolution in software systems: foundations of the spe classification scheme. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(1):1–35, 2006.

[3] M. Godfrey and Q. Tu. Evolution in open source software: A case study. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 131–142. IEEE, 2000.

[4] A. Israeli and D. Feitelson. The linux kernel as a case study in software evolution. *Journal of Systems and Software*, 83(3):485–501, 2010.

[5] H. Krasner. Software system measurement and metrics (class notes).

[6] M. Lehman. Laws of software evolution revisited. *Software process technology*, pages 108–124, 1996.

[7] M. Lehman, J. Ramil, P. Wernick, D. Perry, and W. Turski. Metrics and laws of software evolution-the nineties view. In *metrics*, page 20. Published by the IEEE Computer Society, 1997.

[8] G. Robles, J. Amor, J. Gonzalez-Barahona, and I. Herraiz. Evolution and growth in large libre software projects. In *Principles of Software Evolution, Eighth International Workshop on*, pages 165–174. IEEE, 2005.

[9] S. Schach, B. Jin, D. Wright, G. Heller, and A. Offutt. Maintainability of the linux kernel. In *Software, IEE Proceedings-*, volume 149, pages 18–23. IET, 2002.

[10] L. Thomas, S. Schach, G. Heller, and J. Offutt. Impact of release intervals on empirical research into software evolution, with application to the maintainability of linux. *Software, IET*, 3(1):58–66, 2009.