**Evaluating Lehman’s Laws of software evolution using the GitHub API**

**A dissertation submitted in partial fulfilment of**

**the requirements for the degree of**

**MASTER OF ENGINEERING in Computer Science**

**In**

**The Queen's University of Belfast**

**by**

**Jordan McDonald**

**\*ADD SUBMISSION DATE HERE\***

**\*** **Declaration Cover Sheet ADD\***

**\*Add acknowledgements here\***

1. **ABSTRACT**

This paper studies the validity of Lehman’s laws of software evolution when applied to one hundred open source projects hosted on GitHub. The data set that will be used to investigate this objective will be extracted from the GitHub API and focuses on the repository level which provides the novelty to this study. Metrics attained from the API have been extracted and attached to each law in turn as a means of quantifying the analysis and enabling the various hypothesis to provide insight into the validity of each law in this context.

…Add some more info on the conclusions\*\*

1. **INTRODUCTION**

The term software evolution represents the change of a software system as time progresses, factors that instigate this change include various forms of maintenance which can be categorised as adaptive, perfective, corrective and preventative[15]. To evaluate this change Lehman and Belady formulated the laws of software evolution, which attempted to outline the factors that drive growth and development of software, while also taking into account forces that lead reduced progress. Lehman theorised that most software is subject to change over the course of its existence and this change can be driven by a multitude of different events. The goal was to identify a set of laws that these changes would obey, or must obey in order for software to survive (Section 2.1).

The goal of this paper is to examine these laws in the context of open source projects hosted on GitHub, with a dataset mined from the GitHub API as the focal point for the study. GitHub is a hosting website designed for collaboration on a centralised repository of source code. Any user of the website can ‘Clone’ any public repository and read or alter the code, this serves as the backbone of modern open source development and helps facilitate the ‘fork and pull’ model of development. In addition to code hosting, collaborative code review, and integrated issue tracking, GitHub has integrated social features. Users are able to subscribe to information by “watching” projects and “following” users, resulting in a feed of information on those projects and users of interest. Users also have profiles that can be populated with identifying information and contain their recent activity within the site [2].

As of 2015, GitHub reports having over 9 million users and over 21.1 million repositories [3] making it the largest host of source code in the world [4]. This represents a period of rapid growth considering in 2010, announced on the official GitHub blog it was revealed that one million repositories were hosted on GitHub. These factors in tandem with the accessible GitHub API’s data on commits, code churn, issues, watchers and pulls among other metrics provide an excellent foundation to examine Lehman’s laws in a lesser analysed context and at a volume which in my current knowledge has not been addressed fully on another study.

This paper will perform a large scale analysis of open source projects hosted on GitHub, extracting data at the repository level in order to determine if Lehman’s laws hold or are contradicted by the findings. Each law will be represented by metrics taken from the API and the evolution of these metrics over time will provide an insight into software growth patterns, which in turn shall test the validity of the laws devised by Lehman.

1. **BACKGROUND AND RELATED WORK**
   1. **Background**

Initially devised in 1974 Lehman’s laws have undergone multiple changes as the years have progressed, with the latest alteration taking place in 1996. In his 1980 article [5] Lehman qualified the application of such laws by distinguishing between three categories of software:

* An S-program is written according to an exact specification of what that program can do.
* A P-program is written to implement certain procedures that completely determine what the program can do (the example mentioned is a program to play chess).
* An E-program is written to perform some real-world activity; how it should behave is strongly linked to the environment in which it runs, and such a program needs to adapt to varying requirements and circumstances in that environment.

It is evident that the laws reflect the E-program definition devised by Lehman, the emphasis on feedback and adaptations of software are key components of evolution. Each project in this study will in turn reside under the E-program umbrella and each law is applicable to this category, see below for a summary of each.

* **(1974) "Continuing Change"** - an E-type system must be continually adapted or it becomes progressively less satisfactory[5]
* **(1974) "Increasing Complexity"** - as an E-type system evolves, its complexity increases unless work is done to maintain or reduce it[5]
* **(1974) "Self-Regulation"** - E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal[5]
* **(1978) "Conservation of Organisational Stability (invariant work rate**)" - the average effective global activity rate in an evolving E-type system is invariant over the product's lifetime[5]
* **(1978) "Conservation of Familiarity"** - as an E-type system evolves, all associated with it, developers, sales personnel and users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.[5]
* **(1991) "Continuing Growth"** - the functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime
* **(1996) "Declining Quality"** - the quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes[6]
* **(1996) "Feedback System"** (first stated 1974, formalised as law 1996) - 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
  1. **Related Work**

Attempts at general data mining from GitHub has been prominent in recent years, Kalliamvakou et al [2] published a paper that highlighted the ‘promises and perils of mining GitHub’. This paper has a focus on avoiding common pitfalls in GitHub mining and concluded that there is valuable data to be found if these are avoided. M.M. Mahbubul Syeed [11] has previously performed a systematic literature review into the evolution of open source projects, the authors examine the data sets utilised, sources of the data and research trends in recent years. The author found that Lehman’s laws do not hold in certain cases, with individual laws in the research yielding contradicting results in regards to open source projects.

Additional papers have provided much more focused studies, Jyoti Sheoran et al [7] investigate the watcher mechanic on GitHub, which provides notifications to user who watch a repository each time an event occurs such as a commit or creation of an issue. The paper hones in on the contributors of a project, tracking to process of a user becoming a watcher to finally contributing to a project, finding that this process accounts for a huge bulk of the tested projects eventual contributors. Another study on this topic was conducted by Xu Ben et al [9] which performed visualisation on metric related to commits, low level code statistics and lines of code on a single project, this restriction limits the usefulness of the research. Georgios Gousios et al [4] look in depth at the GitHub ‘fork and pull’ model of development on a sample of 291 projects. The metrics utilised are among the widest ranging in previous literature, considering feature sets for the pull request itself, the project and the developers involved. An analysis was made on what projects utilise this model, the turnover rate of pull request and why requests are rejected. [11] Provides insight into what constitutes a projects popularity on GitHub using the starring mechanic, the paper theorised that this could be tracked over time to show the evolution of popularity. [13] Analyses issues (bugs) as part of open source software, correlating the data with watchers, forks and other metrics.

A similar study to that presented in this paper in regards to evolution was performed by Jesus M. Gonzalez-Barahona et al [8] was conducted on a long running FLOSS project, glibc inside a SCM repository with over 20 years of history. The paper also approaches the research through reference to Lehman’s laws. The metric utilised has a focus on commits, lines of code and files changed to represent evolution – a downside to this study is single project focus, this paper hopes to consider a much larger dataset in order to draw novel findings. [17] [18] take a single and seven project approach respectively with a focus on long running projects such as SQLite and the open source browser Firefox. [19] Has a sample size of nine projects and utilises code level metrics such as KLOC [10] also delves into software evolution and Lehman’s law, however from the context of databases.

**3.3 Novel approaches in this paper**

On conclusion of the literature review gaps in the research were identified from which novel contributions to the field could be made. Evaluating Lehman’s laws according to data from the GitHub API has not yet been fully investigated. This paper plans to represent each law with relevant metric and quantify the evolution of these data points. Prior studies that are similar to the approach in this paper have flaws – A) only investigating one project B) looking at evolution from the stand point of databases. This study will encompass a large data set with variation in the language of choice for the repositories, from this it will be possible to determine if a pool of different programing languages will support or contradict Lehman’s laws.

1. **PROPOSED METHODOLOGY**

**4.1 Research Questions & Hypotheses**

In order to provide scope to the research presented in this paper it is critical to set clear and defined research questions. Research question one will focus on the validity of Lehman’s laws in the context of open source GitHub projects, with multiple hypotheses that will attempt to draw out the relationship between each law and the metrics extracted from the API. A caveat of law three that has to be considered is a reduced scope due to the huge amount of possible metrics available that need considered, in this case we have restricted it to three data points.

**RQ1** - Is it possible using data extracted from the GitHub API to determine if OS software evolution over time reflects Lehman’s laws?

**H1** – If the amount of commits decreases the amount of star gazers will also reduce (law 1 + 6)

**H2** – Total lines of code increases as software system evolves (law 2)

**H3** – Issues, additions and deletions over time for will be normally distributed (law 3)

**H4** – As software evolves changes to lines of code should remain invariant over time (law 4)

**H5** – As Lines of code increases the amount of issues will also increase (law 5)

**H6** - Project issues will increase as code churn decreases (law 7)

**H7** – As the number of issue comments increases the number of issues should decrease (law 8)

**4.2 Project Selection**

To provide scope to the research performed in this paper, a process of identifying the volume and variation of the projects attained from GitHub needs to be defined - figure 1 demonstrates the selection process. The ten programming languages of choice have been chosen based on a ranking system seen in the GitHub blog post [14] which shows the top ten used languages (based on total active repositories) on the site in public and private repositories (excluding forks) as of August 2015.

Top 10 languages on GitHub [14]

1. JavaScript
2. Java
3. Ruby
4. PHP
5. Python
6. CSS
7. C++
8. C#
9. C
10. HTML

Select another programming language

Search each language for the most popular project (sorted by stars)

Identify the top ten languages

If total projects for the current language equals ten?

Has the current project been on GitHub for five years or more?

Does the project have a fifty percent plus affinity to the target language & metrics fully populated?

Add the project to the selection for analysis

Figure 1 - Flow chart showing the project selection progress

It is crucial to apply restrictions to the projects selected for each programming language in order to visualise the evolution of the software effectively and maintain the integrity of the target programming language requirement. The GitHub advanced search facility on the site allows the descending ordering of the ‘most stars’ for a programming language, each sequential project is then evaluated against two criteria.

1. Duration of project life on GitHub, with a set five year threshold which is chosen to ensure evolution can be mapped over a sustained period of time.
2. It is very common for most projects to use multiple programming languages, however GitHub allows users to examine a project for the breakdown of languages utilised. Using this each project prior to analysis has to meet the 50% target language affinity requirement.

**4.3 Data Collection**

GitHub provides a robust API which is ideal for mining the data associated with a project. The current version of the API is version three and all requests are performed over HTTPS, the data is returned in a JSON format which allows simplistic parsing of the metric required. Disadvantages to the API include the pagination system which restricts the amount of data that can returned in one request, which may lead to multiple similar requests taking place. The method utilised to collect this data will be AJAX as implemented in the JQuery JavaScript library, then once processed stored in MongoDB database.

**4.4 System Design**

To enable the research a workbench has been devised which will handle the automated collection of the data for each of the one hundred projects and to execute the statistical functions. The interact with the GitHub API the JQuery library will be leveraged in order the extract the relevant data via a HTTP call using Ajax, the response from the API will take the form of JSON which can then be parsed as required. In order to answer the hypotheses various statistical methods would need to be applied, to handle this the R environment was integrated in the workbench in order to reliably get results using the built in libraries of R.

**4.4.2** Overall System Architecture

URL(s)

Webpage

GitHub API

Raw data

JSON data

JSON extractor module

MongoDB

Java Servlet

DB Query

R Environment

Figure 2 – shows the general system processes

**4.5 Data Analysis Methods**

Now it would be prudent to discuss the structure of the parsed data, each metric is associated with an accompanying time series that signifies the start of a weekly interval. The dataset itself is organised into a vector with each point containing weekly counts of the frequency of the metric in that particular time period. some data points may have gaps between frequencies that exceed the weekly structure, therefore padding has been introduced to fill the gaps in a project as required, in this case each padded weekly interval will be assigned a zero to signify no activity in that period. To ensure the integrity of the research the first six months for each projects have been ‘trimmed’ this is to account for projects that have origins that outlive the GitHub platform. The reason for this is to remove the possibility of initial ‘dump’ of data from a pre-existing polluting the results with the potential for significant statistical outliers. The metrics that will be extracted from the API in order to quantify the analysis are listed below, the relationship between these and the hypotheses has been covered in a previous section (with additional metrics added for the flexibility of the workbench).

* **Stargazers** - Repository Starring is a feature that lets users bookmark repositories. Stars are shown next to repositories to show an approximate level of interest [16].
* **Commits** - A commit, or "revision", is an individual change to a file (or set of files).
* **Additions & Deletions** – represent modified, added or removed lines of code.
* **Issues** - Issues are suggested improvements, tasks or questions related to the repository.
* **Fork** - A fork is a personal copy of another user's repository that lives on your account. Forks allow you to freely make changes to a project without affecting the original. Forks remain attached to the original, allowing you to submit a pull request to the original's author to update with your changes.
* **Commit Comments** – Messages that a user has attached to a specific commit.
* **Tags** – Often created when a new version of the project is released.
* **LOC** – total lines of code at a certain point
* **Growth Rate** – how much a certain metric changed per time interval

**4.5.1** – Statistical Methods

**4.5.1.1** **Growth Rate**

This equation has significant value in the context of software evolution, where values are analysed over a period of time. In particular in tandem with an LOC metric to answer to hypothesis two which is an ideal use case for this statistic. In addition to this it will have value when applied to hypothesis five, but in this case it will be mapped to a time series and cross correlation with issues. This statistic will be utilised to generate a percentage showing the amount of growth,

1. Growth Rate -
2. Average Growth Rate –

X = current value

Y = past value

N = total samples

**4.5.1.2 Shapiro Wilks Test**

This particular test will be applied to the three metrics stated in hypothesis three in order to determine the distribution of the data and evaluate the normality. This particular statistic utilises the null hypothesis principle using a set alpha (0.05 in this case) if the p value is below this threshold then for each metric it will be considered normally distributed population data.

**4.5.1.3 Cross Correlation**

To adequately answer hypotheses one, five, six and seven a cross correlation will be performed which will quantify the relationship between two time series by identifying lags of series x that will be useful predictors of series y. In the case of this research, a lag of -2 will be considered to determine if a change in one metric two weeks prior will have an impact on a series two weeks in the future, in other words to determine if x leads y.

**4.5.1.4 Variance & Standard Deviation**

Law four concerns itself with an invariant work rate, this can be interpreted to applying a variance on the growth rate of the projects LOC. The growth rate will become a series of growth rate values between each weekly LOC, the variance will be applied to this series and a medium operation will be applied to the priori generated one hundred variances in order to facilitate discussion.

**5. RESULTS**

**5.1 – Hypothesis One**

The results generated for each hypothesis will now be examined in sequence, HP1 which represents laws one and six will be initially examined. A lagged cross correlation was performed with multiple different values in order to determine if and when the impact of making a change i.e. a commit will have a direct effect on stargazers and in particular what duration is of time after a commit is the change felt most significantly. The results of this experiment are shown in figure three which shows the results with a lag ranging from -5 to no lag applied.

|  |  |
| --- | --- |
| Lag Amount | Percentage of positive correlations |
| 0 | 59% |
| -1 | 61% |
| -2 | 57% |
| -3 | 60% |
| -4 | 60% |
| -5 | 54% |

Figure 3 – percentage positive cross correlation at different lags

The results in figure three show that there is a consistency in the amount of correlation between the two metrics despite the variation in lag values. However it is worth noting that a -5 lag for commits has the lowest percentage, it is possible that the further apart the commit frequency in a particular week from the present time the less impact it will have as its changes could already have affected the stargazers at a prior point in time. In terms of reflecting the results back to the laws it represents, the percentage of instances that commits and stargazers correlating is not conclusive to say whether or not the law holds in both cases, however the evidence in this case would suggest the law is supported more than refuted. To visualise the distribution of coefficients for each lag a series of graphs have been provided in figure four, the points in each graph generally gravitate around the zero point with no strong trends to assist in providing a conclusive outcome.

Figure 4 – graphs showing the distribution of correlation values for each of the 100 projects on different lags

**5.2 Hypothesis Two**

For each of the one hundred projects in the dataset the growth rate algorithm define previously was applied in order to determine the amount of growth from the projects first week and the last week attained. See figure five for a visualisation showing the amount of projects that either had a positive or negative growth. Based on these results the majority of projects show a positive increase in LOC over the course of its life time, this evidence would suggest that law two has been supported using these one hundred projects.

Figure 5 – shows the amount of projects whose LOC increased or decreased over time

**5.3 Hypothesis three**

After applying the Shapiro Wilks test to the three metrics on each of the one hundred projects a p-value was generated to determine the significance of the normality for each series. These p value were then applied to an alpha (0.05) and the percentage of each metric that is encapsulated with this threshold is then generated, see figure five. The distributions of deletions and additions unanimously support the law, issues is not as conclusive but remains a significant enough percentage to consider this law fully supported.

|  |  |  |
| --- | --- | --- |
| Percentage of Issue | Percentage of Deletions | Percentage of Additions |
| 93.548386 | 100% | 100% |

**5.4 Hypothesis four**

Once the variance growth rate was collected for each project a median value was taken - **30.290**. The independent variance values are plotted in figure 6 in raw and logarithmic form to account for a significant outlier in the dataset. In this particular hypothesis to support Lehman’s fourth law the work rate should remain invariant and therefore close to the mean growth rate for each particular project, from these graphs it is difficult to determine a conclusion, to add value to these variance results standard deviation of the data will also be considered. To do this the standard deviation for each project was calculated, form this it would be useful to find out how many of the weekly growth rate instances were within one standard deviation from the mean- see figure seven. This graph indicates that a large proportion of each projects growth rate vector lies within one standard deviation, therefore remaining close to an invariant LOC growth rate – based on this interpretation of the data Law four appears to be supported.

Figure 6 – distribution of LOC growth rate variance for each of the one hundred projects

Figure 7 – % of each projects growth rate values within one standard deviation

6. Threats to Validity

**6. REFERENCES**

* 1. Lehman, Meir M. (1980). "Programs, Life Cycles, and Laws of Software Evolution". Proc. IEEE 68 (9): 1060–1076.
  2. Eirini Kalliamvakou et al, The Promises and Perils of Mining GitHub, MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Pages 92-101
  3. "GitHub Press Info". github.com. GitHub. Retrieved 2015-03-30.
  4. Georgios Gousios et al, An Exploratory Study of the Pull-based Software Development Model ,MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Pages 384-387
  5. Lehman, M. M. (1980). "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle". Journal of Systems and Software 1: 213–221.
  6. Liguo Yu and Alok Mishra (2013) An Empirical Study of Lehman’s Law on Software Quality Evolution in International Journal of Software and Informatics, 11/2013
  7. Jyoti Sheoran et al, Understanding "watchers" on GitHub, MSR 2014 Proceedings of the 11th Working Conference on Mining Software Repositories, Pages 336-339
  8. Jesus M. Gonzalez-Barahona et al, Studying the laws of software evolution in a long-lived FLOSS Project, JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS J. Softw. Evol. and Proc. 2014; 26:589–612
  9. Xu Ben, Shen Beijun,Yang Weicheng, Mining Developer Contribution in Open Source Software Using Visualization Techniques, Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference, 16-18 Jan. 2013, 934 – 937
  10. Ioannis Skoulis et al, Open-Source Databases: Within, Outside, or Beyond Lehman's Laws of Software Evolution?, 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), At Thessaloniki, Hellas
  11. M.M. Mahbubul Syeed, Imed Hammouda, Tarja Syst¨a, Evolution of Open Source Software Projects: A Systematic Literature Review, Journal of Software, Vol 8, No 11 (2013), 2815-2829, Nov 2013
  12. Hudson Borges, Marco Tulio Valente, Andre Hora, Jailton Coelho, On the Popularity of GitHub Applications: A Preliminary Note
  13. Tegawende F. Bissyand ´ e, Got Issues? Who Cares About It? An Investigation of Issue Trackers of 105 Projects, DOI: 10.1109/ISSRE.2013.6698918 Conference: Software Reliability Engineering
  14. Language Trends on GitHub, https://github.com/blog/2047-language-trends-on-github, August 19, 2015
  15. WG\_LCP - Working Group for Life Cycle Processes, Oversight Committee: C/S2ESC - Software & Systems Engineering Stand, 14764-2006 - ISO/IEC International Standard for Software Engineering - Software Life Cycle Processes – Maintenance
  16. <https://developer.github.com/v3/activity/starring/>
  17. Yan Dong, Shahab Mohsen, Does Firefox obey Lehman’s Laws of software Evolution?
  18. Guowu Xie et al, Towards a Better Understanding of Software Evolution: An Empirical Study on Open Source Software, Software Maintenance, 2009. ICSM 2009. IEEE International Conference on 20-26 Sept. 2009
  19. Iulian Neamtiu et al, Towards a better understanding of software evolution: an empirical

study on open-source software, JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS

J. Softw.: Evol. and Proc. 2013; 25:193–218