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

**Jordan McDonald - 40063974**

1. **ABSTRACT**

This paper studies the validity of Lehman’s laws of software evolution when applied to 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 from the API will be identified and attached to each law in turn, with the objective of visualising software evolution over time in order to support or contradict the laws devised by Lehman. At the end of the study the results are expected to be varied, with some laws holding and others floundering, due to the absence of research in this topic, this paper will attempt to fill this void and provide conclusions that contribute to the field.

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 for reason discussed prior. 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 untapped context (to my knowledge).

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 is 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 live 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 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 different programing languages and other mitigating factors 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 with 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. Research question two builds on RQ1 to examine the factors that affect software evolution across different paradigms such as language utilised, releases and collaborators.

**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 increase as software 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 not fluctuate (law 4)

**H5** – Total lines of code in the system increases incrementally at an average rate (law 5)

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

**H7** – As issue comments increase the rate of closing issues should also increase (law 8)

**RQ2** – What factors affect adherence to Lehman’s laws for projects hosted in GitHub?

**H8 –** The choice of core programming language affects a projects adherence to Lehman’s laws.

**H9 –** As a project produces more releases, Lehman’s laws become more adhered to.

**H10 –** The more collaborators a project has, the more it adheres to Lehman’s Laws

**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. In particular the choice of language needs to be considered to allow research question two to be answered appropriately, to ensure this is possible 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

Project Selection Process

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

Select another programming language

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? (Remove this step)

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 90% 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**

**4.4.1** Technologies Used

The system (coined as Darwin) will take the form of a web application which a backend supported by Java Servlets and MongoDB. The front end applies web technologies such as CSS, HTML, Google Graphs API and the Bootstrap Library in order to create functional and appealing graphical user interfaces. To interact with the view and the GitHub API JavaScript and by proxy the JQuery library are utilised in order to extract the relevant data from the HTTP response and process it for display on graphs and finally storage in the database. Communication from the Server and the API will make use of the AJAX functionality in the JQuery library which provides a framework to dynamically update webpages and integrate the decentralised processes utilised in Darwin.

**4.4.2** How the System Will Be Used

The project will be designed in a service based approach, the user will have to initially bypass a splash page which will require a login/registration in order to access the main systems query component. To ensure security the application will store the user details in the browser session and navigation between pages will refer to this object and redirect illegal actions to the splash page. To make the system user friendly it will be possible to login using social media details, which reduces the amount of data held on the Mongo database. Once the user has been authenticated a redirect will occur, the next page will allow the user to input one or more GitHub URL’s which can then be used to extract data from the API after submitting. The system will then collect a huge amount of data ranging from commits, stargazers, waters and pulls which will then be rendered in graphs for the user to evaluate. Each graph will be organised into a series of tabs which correspond to category for a set of metrics, inside each tab further actions will be presented in adjusting the time sample and data on the graph. To enable adaptation of queries a custom input tab will be created, the user will be able to select a number of metrics of their choice and plot the evolution of these data points over time. This will allow comparisons to be made between any of the extracted metrics and provide flexibility for future analysis or research questions.

**4.4.3** System Architecture

URL(s)

Webpage

GitHub API

Raw data

JSON data

Parsed data

Visualizer Module

Updated webpage

JSON extractor module

Draw Graph

MongoDB

Java Servlet

DB Query

Figure 2 – shows the general system processes

**4.5 Data Analysis Methods**

In order answer the research questions the data will have to be visualised in graphs which reflect the relationship between the metrics and the corresponding hypotheses. This will be achieved using Microsoft Excel, from this it will be possible to generate graphs for each project which explore how a metric transforms over time. This level of visualisation will allow comparisons to be drawn through a series of statistical methods. 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.
* **Contributor** - A contributor is someone who has contributed to a project by having a pull request merged but does not have collaborator access
* **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.
* **Pull Request** - Pull requests are proposed changes to a repository submitted by a user and accepted or rejected by a repository's collaborators.
* **Commit Comments** – Messages that a user has attached to a specific co4.mmit.
* **Tags** – Often created when a new version of the project is released.

**4.5.1** – Statistical Methods

In order to interpret the data extracted from the API various statistical techniques will have to be utilised. These are crucial in order to determine answers to the research questions and hypotheses, at this stage a few methods have been identified that may prove useful for any future analysis and each has been outlined in the list below, however there is further opportunity to use additional formula as required as the data becomes more known.

**4.5.1.1** Pearson’s Correlation

Among the established hypotheses are a subset that focus on relationships between two collections of metrics (see H1, H6, H7). This algorithm will quantify this relationship for each project over a sustained period of time, this will create one hundred values (one for each project). It will then be possible to perform an analysis and determine if the hypothesis holds and as a side effect law one as defined by Lehman.

Pearson’s Correlation -

X = dataset one

Y = dataset two

**4.5.1.2** Growth Rate

This equation has significant value in the context of software evolution, where values are analysed over a period of time. When hypothesis five is considered it is clear that average growth rate is an ideal measurement from which to draw conclusions in relation to incremental growth of a project over time. Using growth rate formulas it is possible to sample a sustained period of time into equal sections which will allow a comparison to be made which will reveal if lines of code do increase incrementally. The time slice will be varied to determine consistencies and determine if the law applies to the set of one hundred chosen projects. It will then become possible to find out what percentage of the projects lie within the standard deviation of average growth rates. If we consider hypothesis four and two this formula is also applicable as a method of analysis, if law four is abided then the growth rate should be consistent over time with minimal fluctuation.

1. Growth Rate -
2. Average Growth Rate –

X = current value

Y = past value

N = total samples

1. **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