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

Jordan McDonald

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 maintenance or adapting to user requirements. 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, possibly to adapt to consumer requirements or through continuous refinement. 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.

2.1 A summary of Lehman’s Laws

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. BACKGROUND AND RELATED WORK
   1. Background
   2. Related Work – add issues paper

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.

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. [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 I identified gaps in the research 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 papers 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 support or contradict Lehman’s laws.

1. PROPOSED METHODOLOGY

RQ1 –

HP1 – Does the programming language of a GitHub project affect a projects adherence to Lehman’s laws?

HP2 –

HP3 -

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