An Empirical Study on the Evolution of Android Operating System in terms of Lehman's Laws

Nazifa Tasnim Hia
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh

Nishat Tasnim Mim
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh

Abdus Satter
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh

Kishan Kumar Ganguly
Institute of InformationTechnology
University of Dhaka
Dhaka, Bangladesh

ABSTRACT
Software evolution refers to the changes made to a software product to enhance its capabilities. In this phase, different software metrics are measured to ensure the maintainability status. Software evolution has some rules postulated by Lehman and his collaborators. Several empirical studies have been performed to analyze the trend of software evolution in different types of software. Observing the evolution and Lehman's laws applicability in Android source code is the main objective of this research work. Various types of software metrics have been calculated to measure the change among releases. After that, using those metrics, the changing pattern has been analyzed and six out of eight Lehman's laws have been found to be confirmed in Android source code evolution. The remaining law 4 and law 5 are difficult to determine as those require deeper empirical studies in the field of open-source software.

Keywords
Software evolution, software metrics, Lehman's laws.

1. INTRODUCTION
Software evolution is such attribute which controls the ability of the software system thus it can accommodate with different changes in software lifetime[1]. The term"evolution" describes the idea of anything developing over time to achieve various goals. Software undergoes a continual evolution process across the full development life cycle where various upcoming stakeholder or market requirements are addressed. The evolution process is useful to meet the current necessity and maintain the quality level of the software products.

Software evolution is highly observed in OpenSource Software (OSS) systems like industrial software. OSS refers to those kinds of software whose source code are easily accessible, modifiable and distributive. Linux, Libre Office, VLC media player, Mozilla Firefox, Android Open Source project (AOSP) etc., can be referred to as very common and popular OSS systems.

AOSP, has attracted its users with its continuously updated features in each release and achieved widespread use all over the world. It was started its journey in 2003 with Android Inc. now owned by Google. Android being open source the competitors in smartphone operating system industry Android has the largest market share in terms of units shipped worldwide and the number of android users [2]. It has gained vast popularity because its software system is maintained by the developers according to the different stakeholder and market requirements. It has released 22 versions till 2022, as lots of evolutionary changes happened to android, it is quite necessary to focus on its evolution practices to get insight about how it is coping with different changes, continuous growth rates, instabilities and many other complex situations. From this perspective, the following research question is addressed:

RQ: What are the Android operating system's evolution practices in terms of Lehman's Laws?

M M Lehman in his research work [3] [4] [5] observed the software evolution process and claimed that the evolutionary process of an industrial software has eight laws. He stated that these laws characteristics are commonly observed in software maintenance. Many researchers have experimented the evolution process on different types of industrial [6], open source [7], mobile applications [8], Linux kernel [9], web application [10], Eclipse IDE [11] based and many more to evaluate the Lehman's laws. To the best of our knowledge, there is no work to track android operating system's evolution based on Lehman's laws. So, the objective of this research is to analyze android source code based on different code metrics and get insight into the source code behavior and then analyze the Lehman's laws applicability in Android system.

458 android versions from 2009 to 2022 have been used. The only metrics that can be retrieved from the source code are code metrics. Because of this, decisions regarding these two laws—i.e., both the Conservation of familiarity and organizational stability could not be made. Aside from this confusion, it can be said that android adhere to all of Lehman's laws.

This source code analysis will assist both researchers and developers to understand how being an OSS system android is maintaining the evolution process, whether Lehman's laws have created impact on the evolution process, visualizing the trends and understand where the changes are required and where not and many more.

2. BACKGROUND
Lehman's Laws: Lehman observed the nature of software, evolving patterns, coping mechanism to deal with the radical software changes etc. Successful evolution of a software is not easy rather the evolution has some patterns and goes through several constraints. The laws that Lehman postulate are discussed below.

1. **Continuing Change:** Software system should be dynamic to adapt different changes. Specifically, an E-type software needs to be changed depending on its end users' requirements otherwise it cannot rival other new software and over time could lost the favor of its user [3].

2. **Increasing Complexity:**The more software changes its functionality, the more possibility arises to increase the complexity. This law is complimentary with the first one, but the complexity should be minimized as much as possible with continuous changes. Therefore, in maintenance phase, enough preventive mechanism is advised to include for decreasing the complexity rate [3].

3. **Self Regulation:**This law claims that the E-type software evolution process holds the self regulation characteristic. That means, the system keeps a trade-off between the changing requirements and the actual changeability [3].

4. **Conservation of Organizational Stability:**This law says that large organization are not seem to bring change in large scale to maintain the stability during software active lifespan. If lots of changes for example, budget increment, investments to the developers, change in strategies are accepted in large scale then the overall situation causes to the unstable situation in the organization [3].

5. **Conservation of Organizational Familiarity:** Lehman noticed the importance of software familiarity maintenance with its developer and end users. Massive changes in software system slow down the developers productivity rate of developers as well as decreases the interest of end users [3].

6. **Continuing Growth:**Continuing growth means adding functionality to enhance software feature according to the demand of its stakeholders. When a piece of software develops new capabilities, it helps to increase the overall usage of the software [5].

7. **Declining Quality:**A software system may lose its uses if it does not change with time, does not upgrade its functionality etc. This situation arises due to the failure of capability enhancement. To keep the evolution running a software needs to be consistent with in quality maintenance [5].

8. **Feedback System:**In software evolution process, to keep the regularity, feedback system has great importance. This system actually keeps the balance among different stakeholders' suggestions, demands and helps to maintain the system's self-regulation characteristics [5].

3. LITERATURE REVIEW

Researchers have worked on several domains related to software evolution to understand and justify the evolution process according to Lehman's laws. Different evolutionary analysis-based works on open source as well as industrial software system have been performed over the years.

Ayelet Israeli conducted an open source based case study over 800 versions of the Linux Kernel [9]. The objective was to see the systems evolving nature over time, how the system growth changes, the nature of complexity. In this study different software metrics were collected from large number of data set.Giovanni Grano in [12] experimented on the android applications. The objective was to infer the software evolution trends on different versions. A tool named" PAPRIKA" was introduced for monitoring the evolving mobile applications using anti patterns.

Software evolution analysis is not confined to the open and industrial boundary. Researchers are trying to analyze the evolutionary pattern in web-based applications [10]. This research conducted an empirical study on 30 PHP project to investigate whether Lehman's laws are applicable in web applications. They found Continuing Change, Self regulation, Conservation of organizational stability, Conservation of familiarity and Continuing growth are confirmed in that PHP projects.

Taranjeet Kaur performed a study [13] on two OSS (built in C++) named Graphic Layout Engine and Flight Gear Simulator. The core objective was to analyze the evolution pattern of these two software systems and finding the applicability of Lehman Laws using object-oriented code metrics (CBO, WMC, DIT, RFC, LCOM and NOC). In this study it is found that only three Lehman Laws (Continuous change, Continuous growth and Increasing complexity) are applicable in these two software system. They claimed, the applicability of Self-regulation, Organizational stability and Conservation of familiarity are difficult in terms of OSS. Two laws, declining quality and feedback system, are hypothetically related to the changing log of OSSsoftware. They advised for OSS to conduct deeper empirical study for finding more effective outcome of Lehman laws.

A case study [14] was performed by Kalpana Johari and Arvinder Kaur in two java OSS system, named JHot Draw (13 versions) and Rhino (16 versions). This study was conducted using Object Oriented Metrics to investigate the Lehman's Laws applicability's. They found the impact of continuing change, increasing complexity, and continuous growth according to the collected data and metric measurements. They mentioned self-regulation, organizational stability, familiarity conservation is hard to relate in terms of OSS.Their suggestion to conduct OSS related empirical study on large amount of data.

4. METHODOLOGY

In order to conduct the study, 458 tags were collected from GitHub. Then exploited the understand [20] tool to calculate each metric. Analysis was done to comprehend the source evolution based on those metrics. The complete breakdown of this study's phases is given below.

1.1 Data Collection & Processing

Since Android is an open-source project. The majority of its source codes are hosted and maintained on a Google Git repository. Additionally, the repository has a Git mirror that is synced with the primary repository. The sources were retrieved from the Git mirror. There are 715 available tags in the Repository. 458 tags were retrieved from 2009 - 2022. At first, the master branch was cloned, then using the Git information each tag was checked out [25]. Tags of the source are considered as the unit. The Understand [20] tool was used for calculating the required metrics for this study. Every tag is analyzed through the Understand tool. This tool gives a CSV containing a measure of the code metrics for every tag. The release frequency process metric was also calculated by retrieving the tags' id and their release date. Based on it, the monthly release frequency from 2009 to 2022 was calculated. Using the number of files for each version, the incremental change in number of files was calculated.

Software metrics are calculated in order to assess specific software [21] characteristics. Different code metrics are used in the software evolution process to conduct quantitative analysis.Software performance, quality, and team productivity can all be understood through the use of software development metrics, which are quantitative evaluations of a software product.

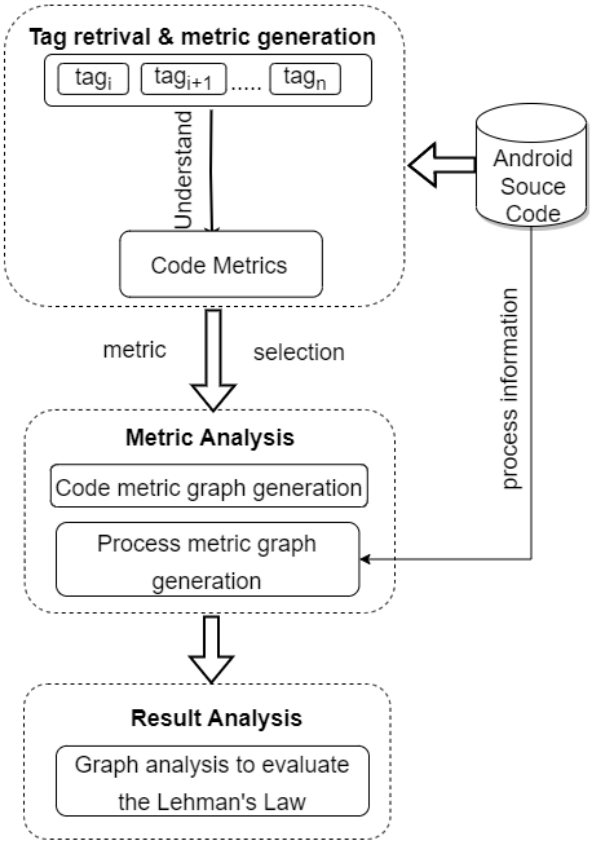


Figure 1: Android Source Code Evolution Process

The following list of metrics areused in this study:

1.1.1 Source Lines of Code (SLOC or LOC) [22]:

This metric is one of the traditional, simple and useful metrics which counts the total line number from a particular program source code. This line counting help to understand the size ofthe program, developers' productivity during development, required effort to maintenance, etc. SLOC measurement can be done from two perspectives. One is counting the physical LOC measurement where comment lines are included with other lines. Conversely, logical LOC is another perspective where only executable "statements" are get counted. In this study, LOC was used to understand the evolution.

1.1.2 Number of Methods (NOM) [22]:

In this metric number of the used or unused methods is counted. This metric helps to understand how many methods or functions are included in the source code.

1.1.3 Number of Files (NOF):

This is another simple metric where the total number of files included in a program is measured. This metric helps tounderstand the systems changes, growth, self-regulation capabilities, familiarity and organizational stability.

1.1.4 Cyclomatic Complexity (CC) [22]:

Cyclomatic Complexity (CC) is a quantitative measure of independent paths in a software program. An Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths

1.1.5 Response for a Class (RFC) [23]:

This metric counts the total method number which is executed in response to a message received by an object of a class.

1.1.6 Weighted Method per Class (WMC) [22]:

Generally, a class consists of several methods. Each method has individual complexities. In WMC the methods included in a given class are considered to do the sum of their individual complexities. The resultant value predicts the overall effort required during development and measures the maintenance effort. A higher value of WMC represents the more complex class.

1.1.7 Release Frequency:

This process metric tracks how frequently monthly version releases occur. Process metrics are those metrics that are employed to enhance the software development process.

As a result, the source code is transformed into a collection of metrics.

1.2 Data Analysis

This study examines the evolution of the source code empirically. Therefore, scatter plots are created from the processed data. Regression lines are fitted through the plots to help understand the pattern of those plots. Whether it obeys Lehman's Law or not was decided based on the coefficients and the standard error.

2. RESULT ANALYSIS

The result analysis performed on the Android source code is discussed in this section. These graphs were created using 458 tags from the years 2009 through 2022. A linear regression line was fitted for each metric to examine how the metrics changed over time. Here's an illustration:

$$\boldsymbol{Y = c + mx} \qquad \textbf{(1)}$$

The general equation for any straight line is equation (1) where m is the gradient (or degree of steepness) and c is the y-intercept (the point where the line crosses the y-axis).The variables x and y are related to coordinates on the line in the linear equation (1).

The formula yields a result for y when a value for x is entered. The tag's serial number (in accordance with the release) is used as an independent variable, and the calculated metrics served as the dependent variable. The gradient of the equation, M, was predicted using these variables. The regression's standard error (S), which measures how far the observed values deviate from the regression line on average, was also determined. Utilizing the units of the response variable, conveniently informshow consistently off the regression model is.

2.1 Law-1: Continuing Change

Continue change means the frequent changes of software with time with different requirements.

Used metrics: Lines of Code, Number of Methods and Number of Files.

2.1.1 Lines of Code (LOC)

The Figure 2is showing the change of Lines of Code over the versions. To understand the pattern of the change, a regression line was fitted through the chart. The equation is given below:

$$Y = 6450.42 + 523944.36x$$

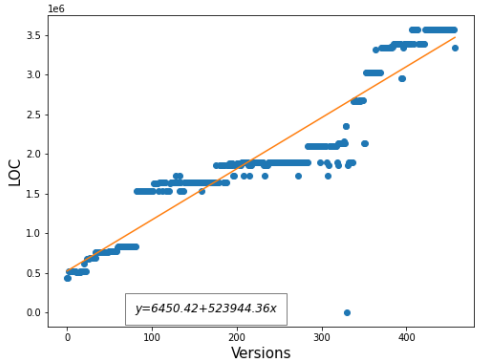


Figure 2: Lines of Code (LOC)

from the equation, the coefficient is found to be positive. So, it indicates a linear increase of LOC over the versions. The standard error should also be considered, which is, in this case108.785. This means a 95% prediction interval would be roughly 2×108.785 = +/- 217.56units wide, which is not too wide and thus this model is sufficiently precise [24].

So, it can be concluded that lines of code have a linear growth as the version number increases.

2.1.2 Number of Methods (NOM)

Figure 3is a plot that shows the change in the Number of Methods over the versions. For understanding the pattern of the change, a regression line was fitted on the data. The equation is given below:

$$Y = 76.32 + 8712.75x$$

It can beinferred that the coefficient's value is positive from the equation. So, it suggests that the number of methods has been growing linearly. The standard error has also been taken into account.It is 1.268 in this instance. This indicates that a 95 percent prediction interval would be approximately 2 ×1.268 = +/- 2.536 units wide, which is not excessively wide and indicates that this model is accurate enough [24].

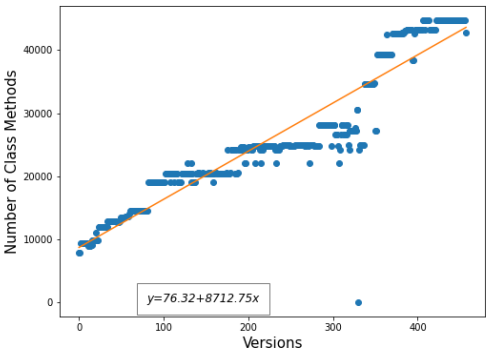


Figure 3: Number of Class Methods

The trend observed here and in the LOC is comparable. The number of class methods also rises as the tag version in a brief period of time.

2.1.3 Number of Files (NOF)

Figure 4 is a graph that displays how the number of files has changed over the versions. A regression line was fitted through the chart to help explain the pattern of the change. The formula is provided below:

$$Y = 24.75 + 2014.20x$$

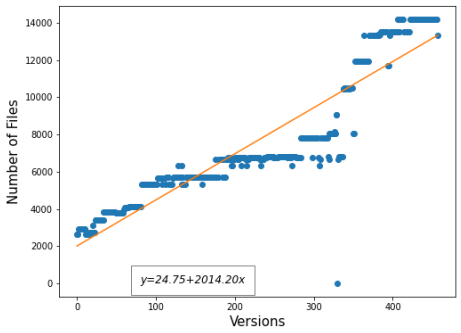


Figure 4: Number of Files

The equation shows that the coefficient is positive. Therefore, it implies that the number of files has increased linearly. As Android is a large system. So, for ease of maintenance, it is necessary to modularize the source code properly. As a result, for incorporating new features or new changes most of the time they introduced new files. Additionally, the standard error should be considered. In this scenario, it is 0.4797. This proves that a 95 percent prediction interval would be approximately 2 ×0.4797 = +/- 0.959units wide, which is not too wide and reveals that this model is sufficient [24].This chart also shows an increasing trend in the number of files with the versions.

The increasing trend in each of the aforementioned charts of the code metrics indicates that changes have occurred during the course of this project. The law of continuing change is supported by Android project.

2.2 Increasing Complexity

When changes are made to software there might be enough risk of increasing the overall complexity. Increasing complexity may become a great loss in software behavior understanding, software testing, and overall software maintenance.

Used metrics: Cyclomatic Complexity (CC), Weighted Method per Class (WMC) and Response for a class (RFC).

2.2.1 Cyclomatic complexity

An increase in Cyclomatic complexity can be a result of the growth of the system or a result of lack of maintenance. The change in cyclomatic complexity is depicted in Figure 5. The complexity in this case was normalized based on the number of classes. As it can be seen from these sections 2.1.1, 2.1.2, 2.1.3show an increasing trend as the releases occur. Additionally, the cyclomatic complexity is a cumulative sum of all class complexities for a release. Consequently, a normalization based on the number of classes is necessary. A regression line is fitted to the graph in order to understand the true pattern of the entire chart.

$$Y = 0.00 + 21.30x$$

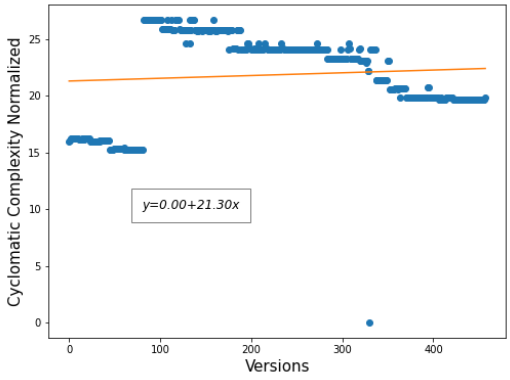


Figure 5: Cyclomatic Complexity

Figure 5shows a sharp rise between 4.x.x and 5.x.x, which is the year 2013 when Android became the most widely used tablet operating system. Though the coefficient is positive. Since 2013 it followed a decreasing trend. It symbolizes that the code was properly maintained to handle the complexity issue.

2.2.2 *Max Cyclomatic Complexity*

Max Cyclomatic Complexity (MCC) is a measure that helps to identify the tolerance level of the system. When the project is in the early phase it tends to grow and adapt to new requirements. As an example, the I/O module has to tackle several types of Input and output commands. Therefore, it is natural to have a lot of conditional clauses in those classes. They learn about all the possibilities, though, once the system has stabilized. Then, those conditions may be reorganized and combined into a single structure or something analogous.

It can be seen inFigure 6 that for some versions, the maximum complexity is high and then in the subsequent version the complexity starts to normalize.

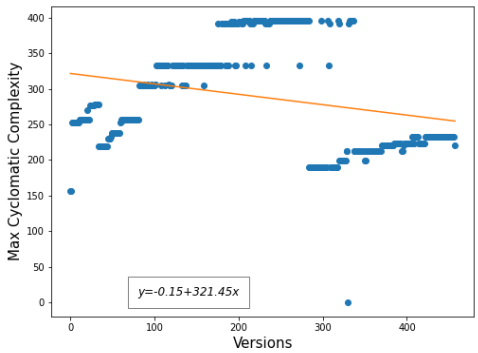


Figure 6: Max Cyclomatic Complexity

So, it resembles the example that when they develop something new it's natural to become complex. However, the change was refactored after it stabilized. The regression equation for this graph is:

$$Y = -0.15 + 321.45x$$

This equation has a negative slope, and the regression line also slopes downward. 0.4797 is the standard error. This demonstrates that the model is adequate because the 95 percent prediction interval would be about 2 ×0.0253 = +/- 0.0506 units wide.This proves that the code is being maintained.

2.2.3 *Weighted Method per Class (WMC)*

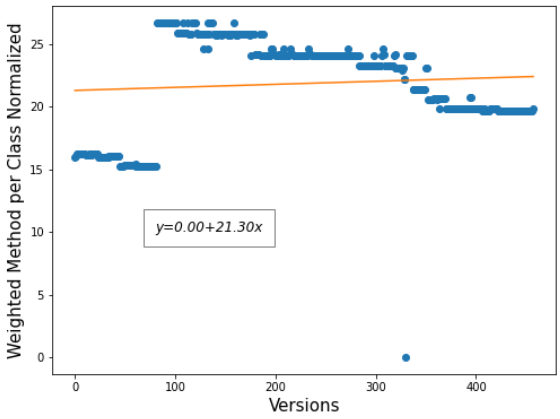


Figure 7represents the change of the complexity per class or weighted method per Class(WMC). Therefore, this plot resembles the plot Figure 5.

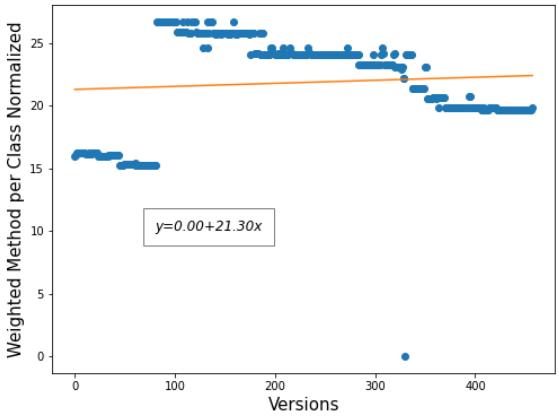


Figure 7: Weighted Method per Class (WMC)

2.2.4 *Response for a Class (RFC)*

The class's overall design complexity increases and becomes more difficult to interpret as RFC increases. Pressman claims that as RFC increases, so does the testing effort necessary because the test sequence grows. On the other hand, a low RFC value denotes greater polymorphism. The RFC value for a class should fall between 0 and 50; in some cases, the higher number maybe 100. It varies depending on the project.

$$Y = 0.02 + 65.37x$$

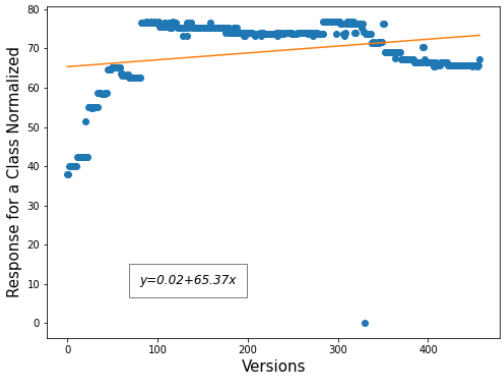


Figure 8: Response for a Class

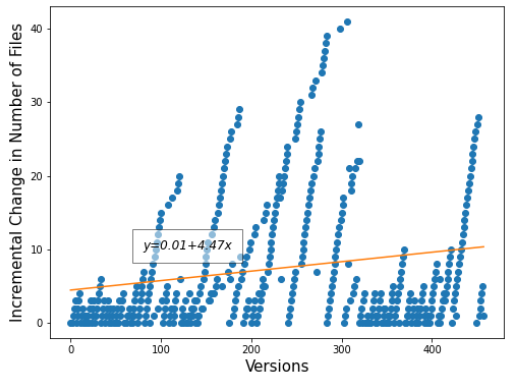


Figure 9: Incremental Change in Number of Files

For the early ages of the Project, a sharp increase in RFC is seen in Figure . A project's size tends to increase more quickly in the first few years as new requirements are more frequently incorporated. After 2013, however, the rate of RFC growth was incredibly slow. Additionally, the slope for the most recent years is down. Due to the size of the Android project and the fact that RFC's value can reach upto 100. It is therefore evident that the project is being maintained and the design complexity is still under control.

Although some metrics of complexity increased over the course of the project, the analysis report of the above chart shows that preventive measures are taken to keep complexity from rising. Therefore, the Android source is consistent with the law of increasing complexity.

2.3 Law-3: Self Regulation

According to this law, the changes in software are accepted in a certain scale when necessary. This law ensures the self regulation of the evolution process to make a steady trend. This law implicitly controls the organization's stability and reserves the familiarity of the software product.

Used metrics: Line of Code (LOC), Number of Method and Number of File.

Result Analysis: There is no exact metric for self regulation measurement. In this work, the self regulation is interpreted by the growth of LOC Figure 2, Number of method Figure 3 and Number of Files Figure 4[22]. From those figures, it is observed that all these measures exhibit a steady increasing trend and slight jumps in major releases. Then the following subsequent minor releases and patches follow a steady growth.

Thus, the self-regulation law is reflected in the Android project.

2.4 Law-4: Conservation of Organizational Stability

According to this law, the average effective global activity rate of an evolving system remains invariant over the life time of the system [6]. Organizational stability refers to the concept that there will not have lots of changes in software product that might be a reason for lots of unstable situations in the organizational environment.

Used Metrics: Incremental Change in Number of files, Releases per month.

2.4.1 Incremental Change in Number of Files

The number of files added or removed from the previous version is depicted in this **Error! Reference source not found.**. The work rate is more stable the fewer variations there are between the versions. To understand the pattern of the change, a regression line was fitted through the chart. The equation is given below:

$$Y = 0.01 + 4.47x$$

The equation gives a positive coefficient and the value is very low. So, it depicts a steady work rate. Though the graph shows some sudden increase that could be a result of the refactoring.

Thus, the familiarity of the project among developers is conserved.

2.4.2 Release Frequency

This Figure shows the release frequency of Android per month over 13 years. For Understanding the work rate, it is an important measure.

Though from the graph and the regression line it is predictable that the work rate was not always consistent. However, there are some points that need attention. This plot is showing a release frequency of 157 months or 13 years. Over these years Android had to cope with many internal and external influences and structural and organizational changes. Under those circumstances, the fluctuation of the release frequency is very low. Most of the time the frequency was below 50.

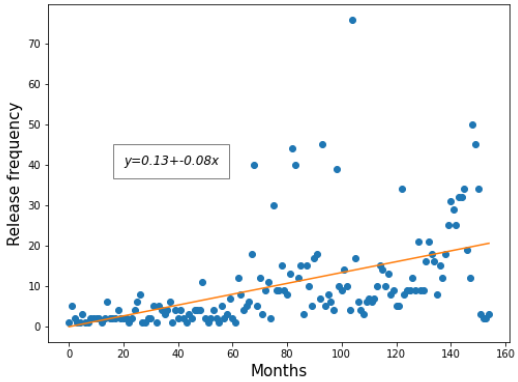


Figure 10: Release Frequency

Though from the graph and the regression line it is predictable that the work rate was not always consistent. However, there are some points that need attention. This plot is showing a release frequency of 157 months or 13 years. Over these years Android had to cope with many internal and external influences and structural and organizational changes. Under those circumstances, the fluctuation of the release frequency is very low. Most of the time the frequency was below 50.

In this study, the invariant work rate is interpreted by Incremental changes in the Number of Files and Releases per month. **Error! Reference source not found.**and Figure showed that the work rate is nearly constant in terms of adding new files or new releases. Only slight inconsistency occurs in releasing major versions. Thus, the law of organizational stability can be interpreted as valid in the Android source Code project.

2.5 Law-5: Conservation of Familiarity

According to this law, the content of successive releases is invariant in an evolving system [1]. In this metric, the changes are scaled in such a manner that the familiaritylevel with the software product remains in the feasible range.

Used Metric: Releases per Month, Incremental Changes in Number of files.

Result Analysis: This law suggests that the change between releases should be limited which allows the developer to maintain familiarity with the code. This law can be validated by observing incremental growth. If it seems to be constant or declining on average, familiarity is conserved.In this study, Releases per Month 2.4.2and incremental changes in number of files are considered to assess the law 2.4.1.

So, the law of conservation of familiarity is supported by Android.

2.6 Law-6: Continuing Growth

According to this law, the functional content of a system must be continuallyincreased to maintain user satisfaction over time [6]. Growthcan be interpreted as an increase in the size of code in orderto add new features. New software is developed from initialrequirements that are the foundation of the system. Sooneror later the changing requirements need to be adapted tomaintain the satisfaction of users that leads to growth.

Used Metrics: Lines of Code, Number of Methods and Numberof Files.

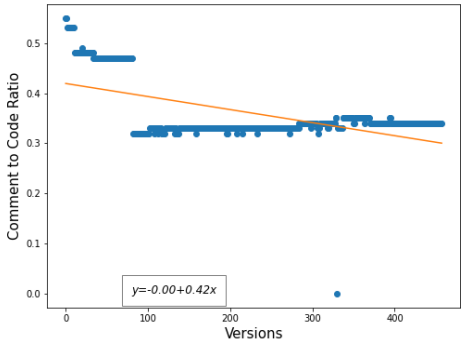
Result Analysis*:*Above metrics measures are used in this study to capture the growth as the increase of these measures reflects the growth of a system by size and functionality. In the above sections 2.1.1, 2.1.2, 2.1.3, it was proved that these metrics grow as the version releases. So, the law of continuing growth is supported by this project.

2.7 Law-7: Declining quality

According to this law, the quality of an evolving system degrades over time unless it is rigorously maintained and adapted to a changing operational environment [6].

Used Metrics:Comment to Code Ratio.

2.7.1 Comment to Code Ratio



This

Figure 8: Comment to Code Ratiodisplays theRatio of Comment Lines to Code Lines across versions. Earlyon, there is a noticeable sharp decline, but after that, the ratioremains constant.

$$Y = -0.00 + 0.42x$$

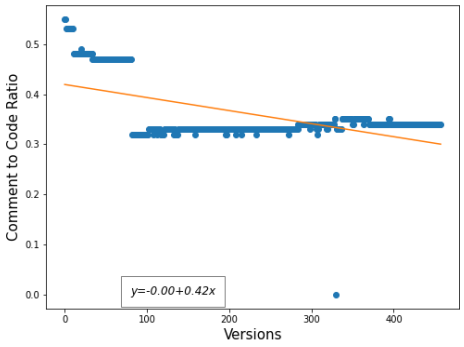


Figure 8: Comment to Code Ratio

The early phase's sudden decline has an effect on the regression line, which is why the coefficient is also negative. However, the ratio is still around 35 %. Though there is no standard value for comment-to-code ratio still it's believed that excellent code has >25% comment-to-code ratio. So, according to this metric, the code is properly maintained. Deciding on this law is difficult as the quality cannot be measured based on some code metrics. Still, the validation of previous laws and the raising popularity of Android indicates that the quality is maintained. So, the law of declining quality is valid for Android.

2.8 Law-8: Feedback System

This law states that asystem's evolution is made up of multi-level,multi-loop, andmulti-agent feedback devices.

Result Analysis: As there is no particular metric to validate this law. It is difficult to evaluate it. This law seems to apply on most to open-source projects accurately because community-driven feature requests, bug reports, and issues are what drive open-source projects. As an open-source project, Android's feature requests and bug reports are managed through GitHub. The opinions of users and the development community are used to guide the development of Android. So, the fact is that the Android project follows the feedback system law.

3. THREATS to VALIDITY

Internal validity: Android source code metadata was collected from the git mirror repository of the main Google git repository. There are 715 available tags of Android source. Due to time constraints, all of the tags could not be analyzed. There are some missing tags from 2009, 2018 and 2020 respectively. This research conducted an analysis of 458 tags from 2009 to 2022. Android source is written in both C++ and Java. However,the focus of this research ison the Javapart as most of the source code is written in Java. As a whole, it can be concluded that some data have been missed that might give more insightful ideas regarding the analysis.

External validity: External validity is such measurements where the findings of the research are tried to generalize with other relative factors such as other people, other settings or other compatible factors. However, in this research, the investigation of the Android source code produced the results that are best fitted in this study.

Construct validity: The understand tool [20] was used for the source code analysis. Only the code metrics are measured by this tool. For the analysis of the data, the CK metric suite [22] was used. However, two metrics from the CK metric suite could not be measured, namely coupling between objects (CBO) and lack of cohesion in methods (LCOM). Only the

release frequency was used for the process metrics. It would be very beneficial to include more process metrics to better understand the project's development

4. CONCLUSION

Software evolution happens to make the software lifespan longer and ensure the software product quality. There are two types of software: Open and industrial. Unlike industrial software, OSSis influenced by different types of stakeholders. In spite of that those open source projects become successful when they can adapt to real world changes. Generally, software evolution follows several significant laws identified by Lehman after his thirty years of software industry level experience. Though Lehman's laws were established on the basis of industrial software (mostly closed), researchers are trying to analyze the impacts on different OSS. Many researchers found these laws strengths in much closed software as well as in OSSand found that these laws have almost equal impact in both cases. However, this research is conducted to see the evolutionary changes in the Android source code and found that in the Android source code evolutionary pattern there is a significant impact of Lehman's laws. The research is conducted using some traditional code metrics and the future plan is to use more process metrics and git repository statistics to do further research.

5. ACKNOWLEDGEMENTS

This research has been partially supported by the University Dhaka, Bangladesh under theCentennialResearch Grant, University of Dhaka, Reference No: Regi/Admin-3/1414.

6. REFERENCES

[1] Rowe, D., Leaney, J., & Lowe, D. (1994). Defining systems evolvability-a taxonomy of change. *Change*, *94*, 541-545.

[2] Haris, M., Jadoon, B., Yousaf, M., & Khan, F. H. (2018). Evolution of android operating system: a review. *Asia Pacific Journal of Contemporary Education and Communication Technology*, *4*(1), 178-188.

[3] Lehman, M. M. (1979). On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software*, *1*, 213-221.

[4] Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, *68*(9), 1060-1076.

[5] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., &Turski, W. M. (1997, November). Metrics and laws of software evolution-the nineties view. In *Proceedings Fourth International Software Metrics Symposium* (pp. 20-32). IEEE.

[6] Lehman, M. M. (1996, October). Laws of software evolution revisited. In *European Workshop on Software Process Technology* (pp. 108-124). Springer, Berlin, Heidelberg.

[7] Xie, G., Chen, J., &Neamtiu, I. (2009, September). Towards a better understanding of software evolution: An empirical study on open source software. In *2009 ieee international conference on software maintenance* (pp. 51-60). IEEE.

[8] Saifan, A. A., Alsghaier, H., &Alkhateeb, K. (2018). Evaluating the understandability of android applications. *International Journal of Software Innovation (IJSI)*, *6*(1), 44-57.

[9] Israeli, A., &Feitelson, D. G. (2010). The Linux kernel as a case study in software evolution. *Journal of Systems and Software*, *83*(3), 485-501.

[10] Amanatidis, T., &Chatzigeorgiou, A. (2016). Studying the evolution of PHP web applications. *Information and Software Technology*, *72*, 48-67.

[11] Businge, J., Serebrenik, A., & Van Den Brand, M. (2010, September). An empirical study of the evolution of Eclipse third-party plug-ins. In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)* (pp. 63-72).

[12] Grano, G., Di Sorbo, A., Mercaldo, F., Visaggio, C. A., Canfora, G., &Panichella, S. (2017, September). Android apps and user feedback: a dataset for software evolution and quality improvement. In *Proceedings of the 2nd ACM SIGSOFT international workshop on app market analytics* (pp. 8-11).

[13] Kaur, T., Ratti, N., & Kaur, P. (2014). Applicability of Lehman laws on open source evolution: a case study. *International Journal of Computer Applications*, *93*(18), 0975-8887.

[14] Johari, K., & Kaur, A. (2011). Effect of software evolution on software metrics: an open source case study. *ACM SIGSOFT Software Engineering Notes*, *36*(5), 1-8.

[15] Yu, L., & Mishra, A. (2013). An empirical study of Lehman's law on software quality evolution.

[16] Alenezi, M. (2021). Internal Quality Evolution of Open-Source Software Systems. *Applied Sciences*, *11*(12), 5690.

[17] de Oliveira, R. P., Santos, A. R., de Almeida, E. S., & da Silva Gomes, G. S. (2017). Evaluating Lehman's Laws of software evolution within software product lines industrial projects. *Journal of Systems and Software*, *131*, 347-365.

[18] Sousa, B. L., Bigonha, M. A., & Ferreira, K. A. (2019, September). Analysis of coupling evolution on open source systems. In *Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse* (pp. 23-32).

[19] Saini, M., Arora, R., & Adebayo, S. O. (2022). In-Depth Analysis and Prediction of Coupling Metrics of Open Source Software Projects. *Journal of Information Technology Research (JITR)*, *15*(1), 1-16.

[20] Scientific Toolworks, Inc, "Understand," Scientific Toolworks, Inc, [Online]. Available: https://www.scitools.com/. [Accessed 10 December 2022]

[21] Li, H. F., & Cheung, W. K. (1987). An empirical study of software metrics. *IEEE Transactions on Software Engineering*, (6), 697-708.

[22] Chidamber, S. R., &Kemerer, C. F. (1994). A metrics suite for object oriented design. *IEEE Transactions on software engineering*, *20*(6), 476-493.

[23] Cook, S., Harrison, R., Lehman, M. M., &Wernick, P. (2006). Evolution in software systems: foundations of the SPE classification scheme. *Journal of Software Maintenance and Evolution: Research and*

Practice, *18*(1), 1-35.

[24] J. Frost, "Statistics By Jim," [Online]. Available: https://statisticsbyjim.com/regression/standard-error-regression-vs-r-squared/. [Accessed 15 December 2022]

[25] Mahir Mahbub. Automated tool for matrics generation. https://github.com/MahirMahbub/Automated-Tool-For-Matrics-Generation, 2022