Mobile Applications Evolution

Achyuta Timsina
Shree Dimna Prajapati

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Authors:
Achyuta Timsina
E-mail: actimsina@gmail.com

Shree Dimna Prajapati
E-mail: shreecim@gmail.com

University advisor(s):
Dr. Samuel A. Fricker
School of Computing

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden
Internet: www.bth.se/com
Phone: +46 455 38 50 00
Fax: +46 455 38 50 57
Abstract

Context. Due to improved computing power, connectivity and interaction capability of mobile devices, their popularity and general acceptance in mass population has increased in recent years. Mobile applications are software systems running on mobile hand-held devices such as smart phones and tablets. Due to obvious differences in mobile applications, the evolution studies on them is of high importance.

Objectives. The primary objective is to study and compare the mobile applications evolution with the Lehman’s laws of software evolution. Next is to identify and report how the software development team size influences mobile applications evolution.

Methods. The study is conducted on 9 different open source mobile applications among which 5 were developed by single core developer and 4 were developed by multiple core developers. The selected projects’ code repository is cloned into local copy and a number of tools are used on those repositories for extraction of relevant metrics from the artifacts. The Lehman laws are tested graphically, analytically and in some cases statistically.

Results. Six of the Lehman’s laws are tested for validation in sample mobile applications. Among the six laws, I-Continuing Change is found supportive, III-Self Regulation and VI-Continuing Growth are found partial supportive in mobile applications. The II-Increasing complexity and V-Conservation of Familiarity are inconclusive. The IV-Conservation of Organizational Stability is not supportive in our sample mobile applications. Moreover, mobile applications are developed by a single or a few developers. Small team mobile projects have less time between releases compared to large team projects.

Conclusions. Majority of Lehman’s laws are not supported in mobile applications evolution. The growth pattern of mobile applications is different than that predicted by Lehman laws.

Keywords: Software maintenance, Software evolution, Lehman’s laws, Software engineering, Mobile applications, Mobile applications engineering
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Commits per developer of projects with single core developer</td>
<td>22</td>
</tr>
<tr>
<td>4.2</td>
<td>Commits per author of projects with multiple core developers</td>
<td>23</td>
</tr>
<tr>
<td>5.1</td>
<td>Changes in lines of code in each release of multiple core developers</td>
<td>25</td>
</tr>
<tr>
<td>5.2</td>
<td>Changes in lines of code in each release of single core developer</td>
<td>26</td>
</tr>
<tr>
<td>5.3</td>
<td>LOC growth trend of projects with a single core developer</td>
<td>28</td>
</tr>
<tr>
<td>5.4</td>
<td>LOC growth trend of projects with multiple core developers</td>
<td>29</td>
</tr>
<tr>
<td>5.5</td>
<td>Incremental change in LOC of projects with a single core developer</td>
<td>31</td>
</tr>
<tr>
<td>5.6</td>
<td>Incremental change in LOC of projects with multiple core developer</td>
<td>32</td>
</tr>
<tr>
<td>5.7</td>
<td>Activity rate of mobile applications with single core developer</td>
<td>35</td>
</tr>
<tr>
<td>5.8</td>
<td>Activity rate of mobile applications with multiple core developers</td>
<td>36</td>
</tr>
<tr>
<td>5.9</td>
<td>Change rate of LOC of mobile applications with single core developer</td>
<td>38</td>
</tr>
<tr>
<td>5.10</td>
<td>Change rate of LOC of mobile applications with multiple core developers</td>
<td>39</td>
</tr>
<tr>
<td>6.1</td>
<td>Time between releases of mobile applications with single core developer</td>
<td>41</td>
</tr>
<tr>
<td>6.2</td>
<td>Time between releases of mobile applications with multiple core developers</td>
<td>42</td>
</tr>
<tr>
<td>6.3</td>
<td>Cumulative authors growth trend of projects with a single core developer</td>
<td>44</td>
</tr>
<tr>
<td>6.4</td>
<td>Cumulative authors growth trend of projects with multiple core developers</td>
<td>45</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Final set of Lehman laws [22, 26] ................................. 5
2.2 Summary of studies on Lehman laws of software evolution in commercial and open source software .......................... 9

4.1 Overview of the sample mobile applications ......................... 21

5.1 Summary results of testing Lehman laws on sample mobile applications ......................................................... 24
5.2 Quadratic correlation of LOC and time in days (since the first release) of sample mobile applications ......................... 27
5.3 Increment change of LOC in each releases of single core developer mobile applications ........................................ 33
5.4 Increment change of LOC in each releases of multiple core developers mobile applications .......................................... 34
5.5 Quadratic correlation on change rate of LOC of all sample mobile applications ......................................................... 37

6.1 Calculated values of mean, median and standard deviation from time between releases of each mobile applications ........ 43
6.2 Quadratic correlation of cumulative authors of each mobile applications ................................................................. 45

8.1 Comparision of mobile apps evolution with the published open source and commercial software evolution .................. 56
Abstract

1 Introduction

2 Background and Related Works
   2.1 Evolution Studies on Commercial Software
   2.2 Evolution Studies on Open Source Software
   2.3 Studies on Mobile Applications Engineering

3 Research Methodology
   3.1 Aims and Objectives
   3.2 Research Questions
   3.3 Methodology
   3.4 Study Design
      3.4.1 Sampling Criteria
      3.4.2 Data Collection
      3.4.3 Data Analysis
   3.5 Validity Threats
      3.5.1 Construct validity
      3.5.2 Internal validity
      3.5.3 External validity

4 Sample Mobile Applications

5 Lehman Laws in Mobile Applications
   5.1 I-Continuing Change
   5.2 VI-Continuing Growth
   5.3 II-Increasing Complexity
   5.4 III-Self Regulation
   5.5 IV-Conservation of Organizational Stability (Invariant Work Rate)
   5.6 V-Conservation of Familiarity
6 Characteristics of Mobile Applications

6.1 Time between releases ........................................ 41
6.2 Growth of application with respect to growth of developers .... 43
6.3 Major/ minor releases and developer involvement .............. 46

7 Analysis and Discussion

7.1 Continuing Change ............................................ 48
7.2 Increasing Complexity ........................................ 49
7.3 Self Regulation .................................................. 51
7.4 Conservation of Organizational Stability (Invariant Work Rate) 52
7.5 Conservation of Familiarity ................................. 53
7.6 Continuing Growth ............................................. 53
7.7 Characteristics of Mobile Applications ........................ 54

8 Conclusion ....................................................... 56

9 Future Work ..................................................... 57

References ........................................................ 58
Due to improved computing power, connectivity and interaction capability of mobile devices, their popularity and general acceptance in mass population has increased exponentially in recent years [5]. With the rapid innovation and development in mobile devices, mobile business companies and independent developers have created various mobile applications (or apps), platforms and distribution mechanisms [1]. A number of centralized mobile apps distribution systems are developed and established by large mobile companies around their target mobile platforms, for example: Google Inc. has Android based apps market place named GooglePlay\(^1\), Apple Inc. has iOS based apps market place named AppStore\(^2\), etc.

Mobile applications are software systems running on mobile hand-held devices such as smart phones and tablets. The mobile devices are not only supporting voice communications but also enriching the usability of mobile devices by capturing the increasingly sophisticated capabilities incorporated into such devices [10]. The apps running on mobile devices are implemented on languages and frameworks provided by the target mobile platform, for example: Java for Android, Objective-C for iOS, C# for Windows phone etc [28]. A particular platform provides mobile app developers with a specific set of software development kits (SDKs), frameworks, and languages. There are a number of open source as well as closed source mobile platforms (or mobile operating systems). Some of open source mobile platforms are Android\(^3\), Tizen\(^4\), etc. and closed source mobile platforms are iOS\(^5\), Windows mobile\(^6\), etc. These mobile apps are developed on a particular platform and are distributed through specific market places, such as Android apps in GooglePlay and iOS apps in iOS AppStore.

Some of the characteristic differences of mobile applications are [5, 10, 35, 37, 1, 28]:

- Limited hardware capabilities (in terms of processor, memory, screen size, etc.) of mobile devices

---

1See www.play.google.com
2See www.apple.com
3See www.android.com
4See www.tizen.org
6See www.windowsphone.com
Chapter 1. Introduction

- Variability of platforms, connectivity, and usage contexts of mobile devices
- Centralized apps distribution channels
- Apps need to be sold in large quantity with cheaper prices
- Developed by relatively small team or by an individual
- Small applications with few specific functionalities

These differences in mobile applications makes it important to study how these applications are designed, developed and evolved [35]. Moreover, the popularity and wide spread use of mobile applications makes it necessary to study mobile application engineering principles and practices [28]. Despite the importance of studying mobile application engineering principles and practices, there are very few scientific studies in mobile applications evolution.

These special characteristics suggests that mobile applications are likely to evolve differently than conventional software. The impact of these significant characteristics on their evolution is not foreseeable. On one side, the business model of mobile applications tends to shorten the release cycle and their relatively small size tends to reduce the planning and coordination effort for enhancing or maintaining them. On the other side, the variability of mobile devices and platforms increases the number of mobile apps versions to be maintained, hence likely to increase the maintenance cost [29]. To extend and maintain such applications practitioners need to have a good understanding of how they evolve.

Belady and Lehman initially proposed laws of software evolution in '70s [3] and the researchers gradually evolved the laws by additional empirical studies in closed source proprietary software [24, 22, 27]. A number of other researchers [11, 2, 9, 36] have evaluated the software evolution laws postulated by Lehman as well. Some of the studies were in closed source commercial software (for example [3, 24, 22, 27, 9, 19]) and some of them were in open source software (for example [11, 32, 33, 17, 20, 8, 13, 36, 16]). The results of the studies are not consistent with the predictions of Lehman’s laws. For example: Linux operating system was found to be growing at a super-linear rate rather than the sub-linear rate as predicted by Lehman laws [11]. Such selective applicability of Lehman laws makes it unpredictable how mobile applications evolve.

Due to increasing importance and functionality of mobile apps, large number of apps are being developed and put into marketplaces that are being widely used by peoples from all walks of life [12]. In this light, this study fulfills an important gap in understanding the mobile applications evolution in terms of Lehman’s laws and the influence of team size in mobile applications evolution. It also helps to identify or suggest the factors in mobile applications engineering that has influenced the observed agreements or deviations to the Lehman laws of software evolution.
Chapter 1. Introduction

We tested validity of 6 Lehman’s laws in this study. The laws tested are I-Continuing Change, II-Increasing Complexity, III-Self Regulation, IV-Conservation of Organizational Stability, V-Conservation of Familiarity and VI-Continuing Growth. We further studied influence of applications development team size in our sample mobile applications. The characteristics we studied are percentage of commits per contributor, time between releases, cumulative contributors and major/minor releases of sample mobile applications, to suggest the factor that may have influenced the observed deviations or agreements to the Lehman’s laws.

We found the first law (I-Continuing change) to be supportive, and partial support for III-Self regulation and VI-Continuing growth. The second (II-Increasing complexity) and fifth law (V-Conservation of familiarity) are found to be inconclusive. The fourth law, that is, IV-Conservation of organizational stability was found not to be supportive at all for our sample mobile applications. Moreover, we found that the mobile applications are developed by a single or few developers, time between releases is small, and partial correspondence in growth pattern of developers with the growth pattern of LOC.

The background and related works are presented in Chapter 2. The chapter 2 explains important concepts in Lehman laws and provides a brief description of related works in open source as well as commercial software evolution. Chapter 3 has aims and objectives, research questions, research methodology, study design, and threats to validity of this study. The selected sample open source mobile applications in this study are described in Chapter 4. Results of testing the Lehman laws in our sample apps are reported in Chapter 5, and Chapter 6 provides the results on the influence of development team size in mobile applications evolution. Analysis and discussion on obtained results are presented in Chapter 7. The thesis report is ended up with conclusion and future works in Chapters 8, and 9 respectively.
Chapter 2

Background and Related Works

A significant portion (more than 60%) of software life-cycle is spent on maintenance till the software is no longer used [24]. Software maintenance is an important process because it consumes a significant part of the overall software life-cycle costs [4]. Besides, business opportunities can be lost if the software could not changed or adapted quickly and reliably [4]. As software system must evolve to satisfy new demands, software evolution studies can lead to increase in understanding of the nature and dynamics of the programming process [9]. Study of behavioral changes in software systems due to continuous maintenance and enhancement over their lifetime is referred to as software evolution [19].

Software evolution consists of activities such as fixing, adaptation, enhancement and so on to maintain stakeholder satisfaction during the lifetime of a software system [23]. Difficulty in capturing empirical data about software projects and longitudinal nature of the field makes it challenging to study the software evolution [19]. But such studies gives us the ability to plan and control software development and maintenance activities effectively [3].

As early as 1970’s Belady and Lehman proposed three laws of software evolution, initially described as “program evolution dynamics” [3] with evolutionary data of IBM OS/360 system. Later on, Lehman categorized the software systems into SPE scheme [24] to acknowledge selective applicability of the software evolution laws. S-type programs can be completely derived from requirement specification, and can be tested for correctness, for example: Calculator. P-type programs can be formally specified completely but can be computationally expensive to reach to the correct result. Hence these programs use heuristics or probability theories to reach more appropriate result. For example: Chess Game. E-type programs emulate the real word processes and becomes a part of the world it models, for example: Operating System. The E-type systems must be continually enhanced, adapted and fixed if they are to remain relevant in a changing world and an evolving application environment [26].

The researchers then gradually evolved the software evolution laws by additional empirical studies in closed source proprietary E-type software [22, 27]. Current form of software evolution laws are stated as in Table 2.1 [27, 26].

After the pioneering work on software evolution by Belady and Lehman [3],
there has been numerous empirical studies conducted on varying commercial as well as open source software. Evolution studies on commercial software systems are summarized in Section 2.1 and that on open source software systems are described in Section 2.2. Even though mobile software engineering is a recent filed, there are some heterogeneous studies in the field. An overview of such studies pertaining to mobile applications evolution are in Section 2.3.

2.1 Evolution Studies on Commercial Software

Belady and Lehman [3] used uni- or multi-variate regression techniques to study data trends of IBM OS/360. They proposed three laws of software evolution

<table>
<thead>
<tr>
<th>Lehman Laws</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Continuing Change</td>
<td>“A program that is used must be continually adapted else it becomes progressively less satisfactory.”</td>
</tr>
<tr>
<td>II-Increasing Complexity</td>
<td>“As a program is evolved its complexity increases unless work is done to maintain or reduce it.”</td>
</tr>
<tr>
<td>III-Self Regulation</td>
<td>“The program evolution process is self regulating with close to normal distribution of measures of product and process attributes.”</td>
</tr>
<tr>
<td>IV-Conservation of Organizational Stability (Invariant work rate)</td>
<td>“The average effective global activity rate on an evolving system is invariant over the product lifetime.”</td>
</tr>
<tr>
<td>V-Conservation of Familiarity</td>
<td>“During the active life of an evolving program, the content of successive releases is statistically invariant.”</td>
</tr>
<tr>
<td>VI-Continuing Growth</td>
<td>“Functional content of a program must be continually increased to maintain user satisfaction over its lifetime.”</td>
</tr>
<tr>
<td>VII-Declining Quality</td>
<td>“Programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment.”</td>
</tr>
<tr>
<td>VIII-Feedback System</td>
<td>“Programming process constitute multi-loop, multi-level feedback systems and must be treated as such to be successfully modified or improved.”</td>
</tr>
</tbody>
</table>

Table 2.1: Final set of Lehman laws [22, 26]
from the observation of 21 releases of IBM OS/360 over 9 years. They have used release sequence number (RSN) as pseudo-time measure, the program size is measured using number of modules, and the system complexity as the fraction of the released system modules that are handled during the course of the release with a particular RSN. Later the researchers evolved the laws of software evolution by additional empirical data on additional commercial systems. In [24], System X’s data from 18 releases over the period of 4.3 years was analyzed and a new set of five laws were proposed. In [22] the current set of eight laws (See Chapter 2.1) were proposed with additional data from Logica FW system. Further, [27] applied inverse square model (initially proposed by Turski [34]) on software evolution studies based on the data from IBM OS/360 and Logica FW system. Twenty six releases of IBM OS/360 and twenty one releases of Logica FW over the period of 8 years were studied in paper [27] as a part of FEAST project. Further extension of the FEAST/1 project, Lehman et al. [25] studied three other E-type software with more than 15 years of evolutionary data over 17 to 29 releases. This study also provides further evidence towards laws of software evolution. The discontinuities observed in Lehman software evolution growth models in seven commercial software projects were examined proposing alternative discontinuous models by Lawrence [21]. However these studies were conducted in large commercial software. There was no evidence that the law also support to the small scale commercial software or small team size software.

Gall et al. [9] studied product release history of Telecommunication Switching System (TSS) over 10 million lines of code and 20 different releases. The studied the overall system evolution behavior as well as its constituent sub-systems evolution behavior. Evolution behavior of systems and sub-systems were completely different in TSS such that it is important to consider evolution of sub-systems to understand complete evolution behavior of the system [9]. From this study, if we take the module B and module C of sub-system C than it can be seen as small scale software system. According to Gall et al. these module B and C are high candidate of redesign. This concludes that the small scale software might behave differently than large scale commercial software system. Another study on software evolution have applied methodologies from social psychology to examine group problem solving processes such as flexible phase mapping and gamma sequence analytic methods in 23 closed source commercial software over the period of 20 years to understand the phases through with a software systems evolves [19].

Another study by Lehman and Ramil [26] revised the prior studies on software evolution by the authors and acknowledged that software evolution laws can be applicable in wide range of software systems from various application areas or environments of their implementation, suggesting a hint that mobile applications could also be studied based on the software evolution laws. Further, Lehman et al. [23] suggest that use of simple growth modeling approaches to software systems can be useful for software evolution and planning.
Chapter 2. Background and Related Works

A study on 23 commercial software systems over the period of 20 years were reported in [2] and found that evolution seems to be happening at a very controllable pace, without significantly increasing or decreasing the complexity and quality of portfolio of the software projects under study.

Law of continuous change, law of increasing complexity and law of continuous growth seems to be supportive in large commercial software systems [27, 25, 9, 21]. While Law of self regulating, law of conservation of organizational stability, law of conservation of familiarity cannot be varified as there are not so many studies done to validate the law. However, [25, 24] support the law.

All the laws were studied in large commercial software system. The evolution of commercial software seem to be validated only in large commercial software systems. Our analysis found that the Lehman laws need to be validated in small scale software system as well. There is no such study that discuss about the evolution behaviour of small scale software systems. Since the large commercial team have the planning about the releases, therefore there seem to have fix time to market. But the time between releases of small scale software system is not known.

2.2 Evolution Studies on Open Source Software

Godfrey and Tu [11] performed software evolution studies on Linux operating system kernel and has reported stark contradiction to the predictions of Lehman laws. They found that the Linux was growing at super-linear rate for a number of years till their study. But other extensive study on large libre software projects has reported that the super-linearity is not the norm but the exception as a few number of software has shown sustained super-linearity [32]. Another contradictory study to that of Godfrey and Tu’s study has reported that the growth of FreeBSD and Linux was liner upper bound and are similar to the growth models of commercial software systems [17].

A large scale empirical investigation on evolution of open source software projects (8791 projects hosted at www.sourceforge.net) were studied by S.Koch [20]. The study found out that the mean growth rate of the projects were linear or decreasing over time according to the laws of software evolution, whereas a significant percentage of the projects were growing in super-linear fashion.

Another large scale investigation on the libre software was reported in [13]. The study used a data set of 12,010 open source projects from FreeBSD packages and projects hosted at SourceForge.net. The study reports that a small subset of basic size metrics are enough to characterize a software system (for example SLOC), software system are self similar, and software evolution is a short range correlated process. This study combines prior empirical studies by the authors which were reported in different venues [14, 15].

Extensive study of Linux kernel on 810 versions released over 14 years has
reported that the average complexity of functions were decreasing with time due to the addition of many small functions into the software systems [16]. The study also explores various alternative metrics to describe the system evolution behavior.

Evolution studies on software systems that are developed in constraint environment was performed with 21 Eclipse (IDE) third party plug-ins over 6 years period [7]. The study shows that the law of continuing change, self regulation and continuing growth are valid whereas, the law of conservation of familiarity was not confirmed and the declining quality was inconclusive. Two open source software systems developed using object oriented languages (for example: Java) were studied for evaluating Lehman laws of software evolution [18]. They reported that the increasing complexity and continuous growth were supportive and others inconclusive.

In close proximity to our study in terms of software size, a study on small size open source software projects were studied and reported in [33]. They studied 2 small open source software projects and reported that the predictions of Lehman laws hold valid in small size open source projects.

Despite a large of number of studies are conducted on open source software systems, they are incoherent so as to establish the validity of the Lehman’s laws unequivocally in such projects. Such incoherency of the reported results makes it necessary to conduct more studies similar to our study.

Law of continuous change seems to be supported in open source software system as well [16, 36]. Not only for large open source software systems, but also with medium-size and small size software systems, the law seem to be supportive [18, 33]. The law of continuing growth also seems to be supportive in open-source software systems. As the number of studies suggests that the open-source software systems undergo continuous growth [16, 36, 7, 15, 32]. However, the growth pattern is not fixed. Few studies have also been conducted in small scale open-source software [18, 33]. While law of self regulating, law of conservation of organizational stability, law of conservation of familiarity cannot be validated because both supportive as well as not supportive literatures were reported [16, 36, 7].

From the evolution studies of open-source software systems it can be said that the number of studies conducted are few. Some of the Lehman laws of software evolution are not applicable in open-source software systems. Moreover, there are very few studies in small scale open-source software. No evidence was found that suggests about the time between releases of small scale software system/team size. It is also unknown whether the growth pattern of application size depends upon the growth pattern of developers in small scale software systems. The relation between major/minor releases between developers is also unknown in small scale software systems.
Table 2.2: Summary of studies on Lehman laws of software evolution in commercial and open source software

<table>
<thead>
<tr>
<th>Lehman Laws</th>
<th>Commercial Software</th>
<th>Open-source Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Continuing Change</td>
<td>Supportive</td>
<td>Supportive</td>
</tr>
<tr>
<td>II-Increasing Complexity</td>
<td>Supportive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>III-Self Regulation</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>IV-Conservation of Organizational Stability (Invariant work rate)</td>
<td>Supportive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>V-Conservation of Familiarity</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>VI-Continuing Growth</td>
<td>Supportive</td>
<td>Supportive</td>
</tr>
<tr>
<td>VII-Declining Quality</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>VIII-Feedback System</td>
<td>Supportive</td>
<td>Inconclusive</td>
</tr>
</tbody>
</table>

2.3 Studies on Mobile Applications Engineering

There is no such studies that explicitly tests the Lehman laws validity in mobile applications. This significant lacking of the study on mobile applications evolution is one of the major motivation factor for our study. However, we found one study conducted on mobile applications engineering [28]. This study is closely related to our study on open source mobile apps hosted at www.f-droid.org. The study reported that apps are significantly different in a number of ways from traditional software systems [28]. The sample apps were studied from a structural and historical perspective. Some of the important findings are:

- Most apps are short-lived single developer projects
- Apps are smaller than conventional software systems (20 KLOC)
- Many apps have a small set of functionalities, thus few classes are required to implement them.
- Due to heavy use of external APIs, apps are complex and consequently difficult to maintain.

From the above characteristics, if we match the characteristics with the Lehman laws, than we can assume that the mobile apps does continuously evolve but for short period. We need to see whether the mobile apps evolve for short period because of small set of functionalities which make the apps to freeze at certain point or because of increasing complexity that make the mobile apps to stop evolving. Hence, the law of continuous change cannot be confirmed here.
As the mobile apps gain complexity due to heavy use of external APIs, it seems that the law of increasing complexity to be satisfied. However, we need to check it further in order to confirm the law of increasing complexity.

Since the apps have small set of functionalities, it seems that the law of continuous growth to be bounded. However, we need to check further to confirmed the assumption for this law.

With no clue for the rest of the law, we will further analyze from our study. However, from the study of Minelli et al.[28], we can consider that the mobile apps are developed by single or few developers. The apps are small scale apps. We will also analyze the influence of development team size in the mobile applications evolution. Therefore, study on relation of time between releases with small scale software system/team size is of importance. Also, the relation between growth pattern of application size and the growth pattern of developers in small scale software systems need to be analyzed. The relation between major/minor releases between developers is also unknown in small scale software systems.
Chapter 3

Research Methodology

This chapter provides the aims and objectives in Section 3.1, and research questions we formulated to fulfill stated aims and objectives in Section 3.2. The study design including the data collection and analysis is presented in Section 3.4. Finally, potential validity threats and mitigation strategies we took to minimize them is discussed in Section 3.5.

3.1 Aims and Objectives

Our aim is to study, analyze and report evolution in mobile applications. The study will fulfill an important gap in understanding mobile applications evolution, and enables us to understand how development team size can influence mobile applications evolution.

The objectives to fulfill the stated aim are:

- To study mobile applications evolution
- To compare the mobile applications evolution with the Lehman laws of software evolution
- To study the influence of development team size on mobile applications evolution

3.2 Research Questions

To achieve the stated aims and objectives, we formulated following research questions:

RQ1 Does the evolution of mobile applications confirm Lehman laws?

RQ2 How does the size of development team influence mobile applications evolution?

   RQ2.1 Do the time between releases correspond with the team size?
RQ2.2 Do the growth pattern of developers corresponds with the growth pattern of application size?

RQ2.3 Are major releases created by more developers and minor releases by few developers?

3.3 Methodology

As this is the empirical research, we first set the research context within which we developed research questions. The details of how we performed retrospective artifacts analysis [31] to extract relevant data from the sample mobile application are described in Section 3.4. Thus extracted data are further analyzed to answer the research questions using statistical and graphical methods. Finally the conclusions are drawn from the result (See Chapters 5, 6), analysis and discussion (See Chapter 7).

The evolution studies on software systems are based on the historical aspect [19, 23], which naturally makes retrospective artifact analysis [31] a suitable research methodology among other alternatives such as experiments, surveys etc. Besides, majority of related works in software evolution studies have used retrospective artifact analysis (For example: [13, 20, 36, 2] etc.).

Chapter 5 answers the research question RQ1. To answer the research question RQ2, we further divided the question into a number of subquestions. The answer to the research question RQ2 and its sub-questions (See Section 3.2) are presented in Chapter 6. Analysis and discussion on the obtained results are presented in Chapter 7.

3.4 Study Design

This section describes the research design for this study. Sampling criteria we used in this study to select open source mobile applications is described in Subsection 3.4.1. Data collection and analysis methods applied on the sample mobile applications are described in Subsections 3.4.2 and 3.4.3 respectively.

3.4.1 Sampling Criteria

As our main aim is to study mobile applications evolution, historical data extracted from mobile apps repositories is used as a primary source. For this study open source mobile apps for Android platform are chosen, because:

- Open source mobile apps repositories are accessible to everyone, which helps to make this study repeatable
• A number of open source mobile apps are freely available for Android platform

A number of open source mobile apps are available in different Online repositories\(^1\). We have used https://www.f-droid.org/ as our source of selecting open source mobile apps sample because it hosts more than 500 open source mobile apps for Android platform and it is growing quickly. The selection criteria of mobile applications for this study is listed as follows:

• Applications must have minimum of 11 release versions
• Applications must have their source code in git repository
• Applications must not be a dead project

The application containing minimum of 11 versions were selected to make the reliable correlation between the metrics used [15]. The samples were limited to git repository, because homogeneous method for sample selection and data extraction could be used. Git is one of the widely popular source code management system today therefore only limiting to git may not create bias in sample population. Dead project are not selected because they are not evolving.

We selected 9 open source mobile applications projects satisfying the stated selection criteria. The selected apps and their descriptive statistics is provided in Chapter 4.

3.4.2 Data Collection

After the completion of sample selection, next phase was to collect the required data to perform the correlation. Since all the selected sample mobile applications were from git repository, we used git commands to extract the data from the repository. The required data for the purpose of the study were LOC (Lines of code), added LOC, deleted LOC, number of authors, release date, number of release versions, and commits per author.

First, we cloned each sample mobile applications from git repository to local repository. Each of the sample mobile applications were cloned to local repository using git command `git clone git:<host address>`. Then using git command `git log`, the log was viewed. This log contains all the information about each commits in the selected projects code base. It contains information such as date of the commit, name of the contributer, version number etc. Each sample application in the git repository contains number of folders, each folder representing each release version. This version folder contain a unique code named ”SHA-1” code. SHA-1 is the unique identity code given to each commits. This SHA-1 code is generated on each commit. This SHA-1 code was noted down. Then the same

\(^1\) See https://www.f-droid.org, http://opensource.ngphone.com/
SHA-1 code was searched in the log of the application using $git\ show\ SHA-1$ command. After matching the code, the information of the whole commit was displayed. From this information, date of the commit was noted. This date is the date of the release version. As a result, released date of each version was obtained. For counting LOC, $git\ diff$ command was used. "$git\ diff\ --shortstat\ [SHA-1\ of\ version\ i-1]\ [SHA-1\ of\ version\ i]$" generates the information such as added LOC, deleted LOC of version $i$ in comparison to version $i-1$. Therefore, using git command $git\ diff\ --shortstat\ [SHA-1\ of\ version\ i-1]\ [SHA-1\ of\ version\ i]$ added LOC and deleted LOC were recorded. For the first release the SHA-1 code of initial commit and the first release is compared. The obtained result is the added and deleted LOC of that release version. Therefore, to obtain the exact increased LOC in the version we need to subtract the deleted LOC. By doing so, the exact added LOC was obtained after previous released version. This total increased LOC of each version were calculated by subtracting deleted LOC from added LOC. For example, total increased LOC of version $i$ is as shown in 3.1.

$$\text{increasedLOC}_i = \text{addedLOC}_i - \text{deletedLOC}_i$$  \hspace{1cm} (3.1)

The total number of LOC of each release version is the sum of increased LOC till that particular release.

Thus resulted LOC was total number of LOC of that particular version which was recorded. To calculate the total number of LOC of first version the git command $\text{git\ diff}$ was used between SHA-1 of first commit and SHA-1 of first official release version, that is, $\text{git\ diff\ --shortstat\ [SHA-1\ of\ first\ commit]}\ [\text{SHA-1\ of\ first\ release}]$ assuming that the first commit of the sample applications were the official project kick off declaration in the git. After using the above command added LOC and deleted LOC of first release version was obtained. Then subtracting deleted LOC from added LOC, the total number of LOC of first release version was obtained. We can also say that for first release version, increased LOC is the total number of LOC of first release.

Version numbers, commits per author and number of authors contributing in each mobile apps were obtained from the respective git repository itself. All the release versions were official releases reported in git repository. All the extracted information was noted in excel sheets and plain text files for further analysis and reporting.

### 3.4.3 Data Analysis

To answer RQ1, quadratic correlation and graphical analysis was performed in 9 different sample mobile apps. We used size in LOC as dependent variable and time in days since the first release as independent variable. The resulted data and plots have helped us to observe the size growth pattern, which is crucial for the study. The research question in RQ2 is subdivided into 3 sub research questions.
For RQ2.1, we analyzed the time between releases of all sample mobile apps. For RQ2.2, we used number of authors added in each releases. The cumulative authors was plotted against the time in days of releases and is compared with the growth pattern of releases. Finally to answer RQ2.3, we analyzed major and minor releases of 9 different sample mobile apps then compared each of the major and minor releases with respective number of authors involved in that particular release version.

Different metrics have been used by different authors in their studies [32, 33, 16, 36, 7]. Use of number of files, number of modules etc are especially for large scale open source software systems. According to Lehman, the use of module is better, because its consistent compared to LOC [32, 26]. However, Godfrey and Tu [11] used LOC metrics to examine the Linux kernel and observed that both metrics give same evolution pattern. Harraiz et al. also used LOC against time as a metric [15]. Herraiz has suggested that a small subset of basic size metrics such as LOC are enough to characterize a software system [13]. Among all other metrics like number of modules, number of files, etc., we also used LOC to measure size because in small scale system like mobile apps measuring growth with uncommented non blank-LOC may or only be the best metrics [33].

I-Continuous change

For I-Continuous change, we have used changedLOC as a metric for size (dependent variable) and time in days since the first release as independent variable. Changes in LOC is the sum of added, modified, and deleted LOC for a particular code base. But git-scm\(^2\) stores one modification as one addition and one deletion, which makes it difficult to extract net modified and deleted lines of code [6]. Moreover, the deleted lines of code is of less importance in this study due to the fact that deletion takes significantly less time and effort in comparison to the addition of LOC. Hence we assumed that changed LOC is approximately equivalent to added LOC for all code bases.

VI-Continuous growth

For VI-Continuous growth, we have used LOC metric to measure the size of particular project at particular release version. For the graphichal analysis, plots are drawn for size in LOC of each release version with time in days since the first release. To study growth pattern of mobile applications, we performed quadratic correlation on the size in LOC as dependent variable (y) and time in days since the first release as independent variable (x).

We have used the quadratic equation 3.2 to study the growth pattern of sample mobile applications, where, \(y\) denotes LOC of the application and \(x\) denotes time

\(^2\)http://git-scm.com/
in days of the application since its first release.

\[ y = ax^2 + bx + c \] (3.2)

The number of LOC and time in days of each application is correlated. The quadratic equation along with Pearson’s correlation coefficient was calculated. The quadratic coefficient value \( a \) obtained from the quadratic equation (See Equation 3.2) is termed as the measurement for the growth rate of the application. The growth rate of the application is categorized in three parts as super-linear, linear and sub-linear. If the growth rate (value of \( a \)) is positive (greater than 0.05) than super-linear, else if the growth rate is negative (less than -0.05) than sub-linear. If the growth rate is zero (0) or near zero (-0.05 < \( a \) > 0.05) than linear [15].

II-Increasing complexity

For II-Increasing complexity, Lehman tested growth of the application using least square linear fit [27]. However, Koch [20] suggested that evolutionary behavior of software projects were better modeled using quadratic equation compared to linear equation. Prior study has suggested that a small subset of basic size metrics such as LOC are enough to characterize a software system [13]. Hence, this study also uses LOC growth pattern as an indicator of software complexity to study II-Increasing complexity.

III-Self regulation

For the law of self regulation, this study uses incremental change in LOC of each release as dependent variable and time in days (since first release) as independent variable. Incremental change in LOC is the difference in LOCs of two consecutive releases. It is calculated using the equation 3.3:

\[ \text{incrementLOC}_i = \text{LOC}_i - \text{LOC}_{i-1} \] (3.3)

where, \( \text{LOC}_i \) is the current LOC, and \( \text{LOC}_{i-1} \) is the LOC of preceding release of a sample mobile apps. For the initial release, \( \text{LOC}_{i-1} \) is ignored.

Mean(\( m \)) and standard deviation(s) of the incremental changes in LOC is calculated. Thus calculated \( m \) and \( m + 2s \) values along with incremental changes in LOC over time in days since the initial release are shown in Figures 5.9 and 5.10.

IV-Conservation of organizational stability

For the law of conservation of organizational stability, we calculated the average number of changes per day using the total number of changes in \( \text{release}_i \) divided by the days in between the \( \text{release}_i \) and \( \text{release}_{i-1} \). The relation we used to calculate activity rate is the Equation 3.4.
Chapter 3. Research Methodology

ActivityRate\textsubscript{i} = \frac{changedLOC\textsubscript{i}}{(Release\textsubscript{i} - Release\textsubscript{i-1})} \quad (3.4)

Where, changedLOC\textsubscript{i} is the total number of additions of LOC in the project code base for a particular Release\textsubscript{i}. Release\textsubscript{i} - Release\textsubscript{i-1} is the time in days in-between releases. For Release\textsubscript{1}, the time in days between the first release and the first commit is noted. Lehman et al. [36] suggested to calculate the possible activity rate by using total number of changes per release. Also, Xie et al. [36] calculated the average number of changes per day.

V-Conservation of familiarity

For the law of conservation of familiarity, this study uses change rate in LOC of each mobile apps. The change rate in LOC is the ratio of changed LOC of a particular release to the total size of code base in LOC, and is calculated using the equation 3.5.

\[
\text{changeRate}_{i} = \left(\frac{\text{changedLOC}_{i}}{\text{LOC}_{i}}\right) \times 100 \quad (3.5)
\]

where, changedLOC\textsubscript{i} is the added LOC on a particular release \textit{i}, and LOC\textsubscript{i} is the size of code base of a particular release \textit{i} in lines of code.

The study use quadratic model to correlate change rate against time. \textit{y} represents the change rate (See Equation 3.5) in quadratic equation of changing rate and \textit{x} represents time in days(since first release). The coefficient \textit{a} in quadratic equation 3.2 represents the change rate growth pattern. If the coefficient \textit{a} in the changing rate equation is positive than the changing rate increases over time and if its negative than the changing rate is invariant or declines over time. Since, the Pearson’s correlation coefficient of changing rate were less, this study further analyze the law by graphical analysis.

3.5 Validity Threats

According to Parry et al. [31] and Koch [20] in empirical studies, threats to validity for retrospective artefact’s analysis from the version control system can be categorized into construct, internal and external validity. Threats to construct validity are described in Subsection 3.5.1. Internal and external validity threats are discussed at Subsections 3.5.2 and 3.5.3 respectively.

3.5.1 Construct validity

Some of early studies used functional content to measure the size, growth and changes into the software system. Therefore, using LOC as a size metrics might be the threat for this study. However, Herraiz et al. [15] reported that using LOC also resulted the same pattern as those measured using number of modules.
Furthermore, Cordy et al. [33] suggested that for small size software systems, use of uncommented non-blank lines of code may or only be the appropriate metric for measuring size and growth pattern of such software systems. Also, their study found that considering total line of code and total uncommented non-blank line of code have similar nature of growth pattern. Moreover, Herraiz has reported that a small subset of basic size metrics such as LOC are enough to characterize a software system [13]. Hence, the metrics used in our study can accurately measure the intended characteristics of mobile applications.

3.5.2 Internal validity

The ability to capture precise size and time of the application releases are concern to internal validity. The size of LOC and calculated time in days (since first release) are measured technically rather than accidentally. Moreover, the releases were also manually inspected for confirmation of the result. In need of further repetition of the data collection, the same data can be extracted through the tool used. Therefore, the internal validity in this study seems to be less problematic.

3.5.3 External validity

The issue here is to generalize the result of this study. The applications under this study are 9 open source mobile applications for Andriod platform. As the selected mobile applications have at least 11 release versions, the correlation analysis performed will be accurate enough for the evolution studies [13]. Moreover, the criteria we used for selecting mobile applications (See Subsection 3.4.1) are particularly devised to minimize the threats from external validity. On the other hand, this study is the first attempt to study mobile applications evolution. Therefore, it is hard to say that the result of this study can be generalized for commercial mobile apps, apps for other mobile platforms such as iOS, Windows Phone, etc. Therefore, a number of studies of similar kinds are needed to generalize the results to broader domain.
Chapter 4

Sample Mobile Applications

Based on the selection criteria presented in Subsection 3.4.1, we selected 9 mobile apps. A brief description of the selected apps:

Relaunch: It is a simple file manager for android mobile. It also consists of features like task killer, application manager, task manager etc. The tool supports in both application mode and in launcher mode. The source code of the application can be found at https://github.com/yiselieren/Relaunch. The programming language used in this application is Java (100%). The first commit of the application was made on Wed Aug 3, 2011 in git repository. It is small scale software mobile application of 10252 LOC.

SMSSync: SMSSync is an android application that allows user to turn the android phone into SMS gateway. The source code of the application can be found at https://github.com/ushahidi/SMSSync. The application is licensed under (GNU Lesser General Public License, v.3) LGPL. The programming language used in this application are Java (90.3%), JavaScript (5.4%), PHP (4.2%), shell (0.1%). The first commit of the application was made on Tue May 11, 2010 in git repository. It is small scale software mobile application of 30907 LOC.

Vanilla Player: Vanilla Player is an android application the plays the song of user’s choice. Its categorization is music and audio. The source code of the application can be found in https://github.com/adrian-bl/vanilla. The programming language used in this application are java (99.5%), others (0.5%). The first commit of the application was made on Mon Feb 15, 2010 in git repository. It is small scale software mobile application of 12351 LOC.

Anki-android: It’s a android application that helps android mobile users to remember things easy. The source code of the application can be found at https://github.com/ankidroid/Anki-Android. Its category is education. The first commit of the application was made on Wed Jun 3, 2009 in git repository. The application is licensed under GPLv3. It is small scale software mobile application of 78377 LOC.
AntennaPod: The application is a podcast manager that does streaming and downloading for Android mobile. Its category is media and video. Its licensed under MIT license. The source code of the application can be found at https://github.com/danieloeh/AntennaPod. The programming language used in this application is Java (100%). The first commit of the application was made on Fri Dec 23, 2011 in git repository. It is small scale software mobile application of 25839 LOC.

MobileOrg: The Android application that supports storing, viewing and editing the org-mode files. Its category is productivity. The source code of MobileOrg can be found at https://github.com/matburt/mobileorg-android. The programming language used in this application is Java (99.9%) and shell (0.1%). The first commit of the application was made on Fri Dec 4, 2009 in git repository. It is small scale software mobile application of 16537 LOC.

Terminal Emulator: It is an Android application that helps to operate Linux command line shell. Its category is tools. It is licensed under Apache2. The source code of the application can be found at https://github.com/jackpal/Android-Terminal-Emulator. The programming language used in this application is Java (94.3%), C++ (3.4%), C (1.9%) and others (0.4%). The first commit of the application was made on Sat Sept 4, 2010 in git repository. It is small scale software mobile application of 13685 LOC.

Github-android: The Android application that helps Android mobile user to stay connected with the projects they involved from anywhere. Its category is productivity. It’s licensed under Apache version 2.0. The source code of the application can be found at https://github.com/github/android. The programming language used in this application is Java (65.7%) and JavaScript (34.3%). The first commit of the application was made on Wed Oct 12, 2011 in git repository. It is small scale software mobile application of 25561 LOC.

Android-autostarts: The Android application that helps to know what application does in background. Which application runs and what other applications event trigger during mobile startup. Its category is tools. It’s licensed under GPLv3. The source code of the application can be found in https://github.com/miracle2k/android-autostarts. The programming language used in this application is Java (100%). The first commit of the application was made on Mon Jun 22, 2009 in git repository. It is small scale software mobile application of 2811 LOC.

Descriptive statistics on the sample mobile apps are collected and further analyzed. We collected total number of developers (or contributors) involved,
total number of commits by each developer, total release versions (major/minor), lines of code, first release and last release date (in YY/MM/DD format). The collected data is listed in Table 4.1.

<table>
<thead>
<tr>
<th>Apps</th>
<th>Devs</th>
<th>Core</th>
<th>Version</th>
<th>LOC</th>
<th>First</th>
<th>Last</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android-autostarts</td>
<td>2</td>
<td>1</td>
<td>24</td>
<td>2811</td>
<td>09/07/26</td>
<td>12/08/14</td>
</tr>
<tr>
<td>Vanilla Player</td>
<td>7</td>
<td>1</td>
<td>19</td>
<td>12351</td>
<td>10/02/28</td>
<td>12/03/14</td>
</tr>
<tr>
<td>Github-android</td>
<td>48</td>
<td>1</td>
<td>13</td>
<td>25561</td>
<td>12/07/09</td>
<td>12/12/05</td>
</tr>
<tr>
<td>AntennaPod</td>
<td>12</td>
<td>1</td>
<td>34</td>
<td>25839</td>
<td>12/07/20</td>
<td>13/04/12</td>
</tr>
<tr>
<td>SMSSync</td>
<td>9</td>
<td>1</td>
<td>14</td>
<td>30907</td>
<td>10/11/10</td>
<td>13/03/20</td>
</tr>
<tr>
<td>Relaunch</td>
<td>3</td>
<td>2</td>
<td>20</td>
<td>10252</td>
<td>11/09/17</td>
<td>12/04/02</td>
</tr>
<tr>
<td>Terminal Emulator</td>
<td>48</td>
<td>&gt;2</td>
<td>21</td>
<td>13685</td>
<td>11/08/29</td>
<td>13/01/26</td>
</tr>
<tr>
<td>MobileOrg</td>
<td>22</td>
<td>2</td>
<td>17</td>
<td>16537</td>
<td>10/02/25</td>
<td>13/01/08</td>
</tr>
<tr>
<td>Anki-android</td>
<td>50</td>
<td>&gt;2</td>
<td>16</td>
<td>78377</td>
<td>09/11/10</td>
<td>13/02/07</td>
</tr>
</tbody>
</table>

*Table 4.1: Overview of the sample mobile applications*

The table 4.1 shows name of the selected apps, total number of developers, number of core developers involved (based on observations from Figures 4.1 and 4.2), total released versions (major/minor), size in LOC, first released date and the last released dates of each applications respectively. If the commits per developer of a single developer is greater than 85% of the total commits in a project code base, then such projects are considered to be developed by a single core developer, else the projects are considered developed by multiple core developers. Contributions by each developer in individual projects are depicted in Figure 4.1 and Figure 4.2. The figures show that among 9 sample mobile apps, five apps have a single core developer (more than 85% commits by single developer) and the remaining four apps have more than one core developer. Rest of the study uses this categorization, and suggests that majority of mobile apps are developed by a single or a few developers.
Chapter 4. Sample Mobile Applications

Figure 4.1: Commits per developer of projects with single core developer
Figure 4.2: Commits per author of projects with multiple core developers
Chapter 5

Lehman Laws in Mobile Applications

To answer research question RQ1. *Does the evolution of mobile applications confirm Lehman laws?*, we individually tested six laws with collected historical data from the project repository. The data collection methods and metrics used for testing each law is described in Sub-section 3.4.2 and 3.4.3. Each section in this chapter presents the result of testing each individual law on the collected metrics from the sample mobile applications. I-Continuing change is at Section 5.1, II-Increasing complexity is at Section 5.3, III-Self regulation is at Section 5.4, IV-Conservation of organizational stability is at Section 5.5, V-Conservation of familiarity 5.6, and VI-Continuing growth is at Section 5.2. The analysis and discussion on the obtained results are presented in Chapter 7.

Summary result on the tested Lehman’s laws for each application is presented in Table 5.1.

<table>
<thead>
<tr>
<th>Lehman laws</th>
<th>Results</th>
<th>Remarks</th>
<th>Sections</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Continuing change</td>
<td>Support</td>
<td>All sample apps are changing continuously</td>
<td>5.1</td>
</tr>
<tr>
<td>II-Increasing complexity</td>
<td>Inconclusive</td>
<td>Applied complexity indicator may not be reliable</td>
<td>5.3</td>
</tr>
<tr>
<td>III-Self regulation</td>
<td>Partial support</td>
<td>Three sample apps do not support</td>
<td>5.4</td>
</tr>
<tr>
<td>IV-Conservation of organizational stability</td>
<td>Invalid</td>
<td>Not a single app has invariant activity rate</td>
<td>5.5</td>
</tr>
<tr>
<td>V-Conservation of familiarity</td>
<td>Inconclusive</td>
<td>Statistical analysis may not be reliable</td>
<td>5.6</td>
</tr>
<tr>
<td>VI-Continuing growth</td>
<td>Partial support</td>
<td>Three sample apps do not support</td>
<td>5.2</td>
</tr>
</tbody>
</table>

*Table 5.1:* Summary results of testing Lehman laws on sample mobile applications

24
5.1 I-Continuing Change

This law states that, if a system is not adapted to its usage environment as it evolves, the system starts becoming less satisfactory [27]. To test this law, changes in LOC of each releases of individual projects are noted. Thus noted data is used to plot graphs of each applications with changed LOC over time in days (since the first release). The graphs showing changed LOC over time in days (since first release) are presented in Figures 5.2 and 5.1.

![Graphs of multiple core developers mobile applications](image)

**Figure 5.1:** Changes in lines of code in each release of multiple core developers

Figure 5.2 presents the graphs of single core developer applications. From the graph, it can be seen that all single core developer mobile applications undergo continuous change since their first releases. Figure 5.1 presents the graphs of multiple core developers mobile applications. From the graph, it can be seen that all multiple core developers mobile applications undergo continuous change since their first releases. All our samples shows support for I-Continuing change, irrespective of developers involvement.
This law might not need statistical reason to prove it. Because the graphs shown in Figure 5.2 and 5.1 clearly shows that the mobile applications are continuously changing since its first release. Moreover, the percentage change in LOC used to construct graphs for conservation of familiarity (See Section 5.6), clearly indicates that the applications are under continuous change since their first release. From Figures 5.9 and 5.10, the changed LOC in the each graph can be seen varying from one to other release version. Which means the mobile apps undergo
continuous change from one version to another. Therefore, it can be concluded that all the mobile applications under this study are evolving continuously.

## 5.2 VI-Continuing Growth

It states that as system evolves the functional content must be increased to fulfill the users’ need [27, 26]. The growth can be regarded as increase in size (LOC) or increase in number of functions of the system. In this study, we have used increase in LOC as the growth indicator of sample mobile applications.

From the Figures 5.3 and 5.4, it can be clearly seen that the size (LOC) of all mobile applications have increased since their respective first releases. The corresponding growth patterns of sample mobile apps are listed in the Table 5.2. Moreover, the Table 5.2 indicates that all the mobile applications growth pattern are well fitted to the quadratic Equation 3.2 with average Pearson’s correlation coefficient ($R^2$) of 0.947. As can be seen from the graphs, some of the release versions of Github-android, SMSSync and ReLaunch applications have decreased LOC count in some release with respect to prior releases. These observations have raised questions in validity of VI-Continuous growth in our sample mobile applications. Hence, it can be concluded that there is a partial support for VI-Continuing growth in our sample applications.

<table>
<thead>
<tr>
<th>Apps</th>
<th>Quadratic Coeff.(a)</th>
<th>N</th>
<th>$R^2$</th>
<th>Growth Trend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android-autostarts</td>
<td>0.008</td>
<td>24</td>
<td>0.948</td>
<td>Linear</td>
</tr>
<tr>
<td>Vanilla Player</td>
<td>0.042</td>
<td>19</td>
<td>0.851</td>
<td>Linear</td>
</tr>
<tr>
<td>Github-android</td>
<td>−3.165</td>
<td>13</td>
<td>0.979</td>
<td>Sub-linear</td>
</tr>
<tr>
<td>AntennaPod</td>
<td>−0.162</td>
<td>34</td>
<td>0.958</td>
<td>Sub-linear</td>
</tr>
<tr>
<td>SMSSync</td>
<td>0.122</td>
<td>14</td>
<td>0.935</td>
<td>Super-linear</td>
</tr>
<tr>
<td>ReLaunch</td>
<td>−0.079</td>
<td>20</td>
<td>0.941</td>
<td>Sub-linear</td>
</tr>
<tr>
<td>Terminal Emulator</td>
<td>−0.008</td>
<td>21</td>
<td>0.950</td>
<td>Linear</td>
</tr>
<tr>
<td>MobileOrg</td>
<td>0.019</td>
<td>17</td>
<td>0.988</td>
<td>Linear</td>
</tr>
<tr>
<td>Anki-android</td>
<td>0.002</td>
<td>16</td>
<td>0.980</td>
<td>Linear</td>
</tr>
</tbody>
</table>

*Table 5.2: Quadratic correlation of LOC and time in days (since the first release) of sample mobile applications*

From the Figure 5.3 and Table 5.2, the growth pattern of single core developer was found to be a mix of super-linear, linear and sub-linear trends. While the growth pattern of multiple core developers was found to be linear and sub-linear (See Figure 5.4). This indicates that the applications developed by a single core developer can have super-linear growth while applications developed by multiple core developers may not have super-linear growth pattern. Hence, it can be
concluded that applications developed by small team differs from applications developed by large team in terms of growth pattern in mobile applications.
5.3 II-Increasing Complexity

According to this law, the complexity of the system increases as the system evolves unless the proper measure is taken to control the complexity [27]. Lehman supported the law by stating, as system evolves addition of functionality or necessity of changing operational domains for user satisfaction leads to increase in complexity and system starts to deteriorate [26].

We have used LOC growth pattern as an indicator of software complexity (See sub-section 3.4.3). From Table 5.2, out of 9 sample mobile apps: 3 (Github-android, AntennaPod, ReLaunch) have sub-linear, 5 (Android-autostarts, Vanilla Player, Terminal Emulator, MobileOrg, Anki-android) have linear, and remaining one (SMSSync) has super-linear growth pattern. We further analyzed each releases of every applications to check whether any complexity control major was taken.

The quadratic equation of the SMSSync application is \( y = 0.122x^2 - 47.975x + \)
5998.849, with Pearson correlation coefficient ($R^2$) of 0.935. The quadratic coefficient value $a$ in this case is positive, indicating super-linear growth pattern. For SMSSync application, we found that work was done to reduce the complexity. Code clean up was done in version 2 and version 14. It was also seen that bug fixing was done in version 3, 4 and 9. Similar activities were observed in other applications as well. Even though applications undergo complexity control, some of the applications have shown declining growth. For example, despite complexity control mechanisms performed in ReLaunch, Github-android and AntennaPod, the growth rate started to decline.

Above observation suggests the use of LOC growth pattern may not be an accurate indicator of complexity in software code base for open source mobile applications, consequently making this law inconclusive in our sample mobile applications.

5.4 III-Self Regulation

According to Lehman, the law of self regulation is defined as “The program evolution process is self regulating with close to normal distribution of measures of product and process attributes. [27]”. Lehman [25, 26] provides evidence for the law by stating that the ripple effects are seen in empirical growth on a steady growth trend. The ripples are the interaction between the dependent forces of desired growth and bounded resources.

Lehman [26] used number of modules as dependent variable and release sequence number as independent variable, however Johari et al. [18] have used LOC as dependent variable and time as independent variable to evaluate the Lehman’s law of self regulation. According to Lehman, incremental changes fluctuates around the mean(m) and if the change is near to or more than $m+2s$ than it is almost always followed by near to or sometimes less than zero (or negative) increment, indicating an alteration between growth and stabilization [26].

We calculated incremental changes in LOC of consecutive release on all mobile apps to study this law. Increment change in LOC is the difference in LOC of two consecutive releases. It is calculated using the Equation 3.3 in each releases of all sample mobile applications. Thus calculated increment change in LOC of all sample mobile apps are listed in Tables 5.3 and 5.4. In Table 5.3, first column shows the two consecutive versions from which incremental changes in LOC is calculated. Remaining columns lists the incremental changes in LOC of Android-autostarts, Vanilla Player, Github-android, AntennaPod, and SMSSync respectively. Similarly, Table 5.4 lists the incremental changes in LOC of two consecutive releases of ReLaunch, Terminal Emulator, MobileOrg, and Anki-android respectively. Thus calculated incremental changes in LOC is plotted in Figures 5.5 and 5.6. Also, we calculated mean (m) and standard deviation (s) of incremental changes in LOC of respective samples, to observe the deviation pattern of incre-
mental changes with respect to the mean value. Thus calculated mean(m) and m+2s lines are shown in Figures 5.5 and 5.6.

Observations from the Figures 5.5 and 5.6 indicate:

- Most of incremental changes seems to fluctuate around the mean value, suggesting support for III-Self regulation.
- Incremental changes of all mobile apps have crossed m+2s line at least once

**Figure 5.5:** Incremental change in LOC of projects with a single core developer
in their evolutionary history. But, the incremental changes of greater than or equal to $m+2s$ is almost always followed by near to or sometimes also less than zero, indicating an alteration between rapid growth and stabilization.

- Positive increments are more frequent than negative increments, indicating growth in overall code base of mobile applications from versions to versions, but they are not continually growing.

Despite the presence of ripples (rapid growth and stabilization cycle), three of our sample apps do not have continuous growth. These three applications are Github-android, SMSSync and ReLaunch. This observation indicates only partial support on our sample mobile applications.
<table>
<thead>
<tr>
<th>Versions</th>
<th>Autostarts</th>
<th>Vanilla</th>
<th>Github</th>
<th>AntennaPod</th>
<th>SMSSync</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2-V1</td>
<td>101</td>
<td>-11</td>
<td>2</td>
<td>21</td>
<td>383</td>
</tr>
<tr>
<td>V3-V2</td>
<td>708</td>
<td>42</td>
<td>196</td>
<td>159</td>
<td>46</td>
</tr>
<tr>
<td>V4-V3</td>
<td>27</td>
<td>834</td>
<td>731</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>V5-V4</td>
<td>7</td>
<td>3303</td>
<td>232</td>
<td>-25</td>
<td>1510</td>
</tr>
<tr>
<td>V6-V5</td>
<td>38</td>
<td>774</td>
<td>6413</td>
<td>2</td>
<td>905</td>
</tr>
<tr>
<td>V7-V6</td>
<td>34</td>
<td>511</td>
<td>17720</td>
<td>1096</td>
<td>494</td>
</tr>
<tr>
<td>V8-V7</td>
<td>233</td>
<td>33</td>
<td>2320</td>
<td>831</td>
<td>4556</td>
</tr>
<tr>
<td>V9-V8</td>
<td>143</td>
<td>395</td>
<td>9</td>
<td>673</td>
<td>1667</td>
</tr>
<tr>
<td>V10-V9</td>
<td>95</td>
<td>3100</td>
<td>-7</td>
<td>148</td>
<td>696</td>
</tr>
<tr>
<td>V11-V10</td>
<td>3229</td>
<td>1870</td>
<td>703</td>
<td>177</td>
<td>1277</td>
</tr>
<tr>
<td>V12-V11</td>
<td>655</td>
<td>725</td>
<td>-4182</td>
<td>151</td>
<td>59165</td>
</tr>
<tr>
<td>V13-V12</td>
<td>121</td>
<td>780</td>
<td>0</td>
<td>2166</td>
<td>4259</td>
</tr>
<tr>
<td>V14-V13</td>
<td>158</td>
<td>708</td>
<td>504</td>
<td>-2543</td>
<td></td>
</tr>
<tr>
<td>V15-V14</td>
<td>29</td>
<td>721</td>
<td>601</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V16-V15</td>
<td>4</td>
<td>2523</td>
<td>19</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>V17-V16</td>
<td>6</td>
<td>999</td>
<td>1336</td>
<td>684</td>
<td></td>
</tr>
<tr>
<td>V18-V17</td>
<td>3</td>
<td>5881</td>
<td>712</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V19-V18</td>
<td>7</td>
<td>1361</td>
<td>475</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>V20-V19</td>
<td>3059</td>
<td>9699</td>
<td>6</td>
<td>1361</td>
<td></td>
</tr>
<tr>
<td>V21-V20</td>
<td>2796</td>
<td>4797</td>
<td>390</td>
<td>328</td>
<td></td>
</tr>
<tr>
<td>V22-V21</td>
<td>1531</td>
<td>108</td>
<td>436</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V23-V22</td>
<td>-269</td>
<td>431</td>
<td>1028</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V24-V23</td>
<td>-2</td>
<td>556</td>
<td>2729</td>
<td>467</td>
<td></td>
</tr>
<tr>
<td>V25-V24</td>
<td>436</td>
<td>2819</td>
<td>556</td>
<td>467</td>
<td></td>
</tr>
<tr>
<td>V26-V25</td>
<td>1028</td>
<td>2729</td>
<td>2819</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V27-V26</td>
<td>328</td>
<td>436</td>
<td>556</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V28-V27</td>
<td>436</td>
<td>1361</td>
<td>2819</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V29-V28</td>
<td>556</td>
<td>2729</td>
<td>436</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V30-V29</td>
<td>2819</td>
<td>1361</td>
<td>556</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V31-V30</td>
<td>436</td>
<td>2729</td>
<td>2819</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V32-V31</td>
<td>467</td>
<td>1361</td>
<td>556</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V33-V32</td>
<td>436</td>
<td>2729</td>
<td>2819</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>V34-V33</td>
<td>467</td>
<td>1361</td>
<td>556</td>
<td>-2</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.3: Increment change of LOC in each releases of single core developer mobile applications
Chapter 5. Lehman Laws in Mobile Applications

Table 5.4: Increment change of LOC in each releases of multiple core developers mobile applications

<table>
<thead>
<tr>
<th>Versions</th>
<th>ReLaunch</th>
<th>Terminal</th>
<th>MobileOrg</th>
<th>Anki-android</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2-V1</td>
<td>477</td>
<td>1927</td>
<td>228</td>
<td>0</td>
</tr>
<tr>
<td>V3-V2</td>
<td>75</td>
<td>7</td>
<td>174</td>
<td>6603</td>
</tr>
<tr>
<td>V4-V3</td>
<td>1324</td>
<td>103</td>
<td>201</td>
<td>1091</td>
</tr>
<tr>
<td>V5-V4</td>
<td>-34</td>
<td>640</td>
<td>932</td>
<td>-189</td>
</tr>
<tr>
<td>V6-V5</td>
<td>717</td>
<td>0</td>
<td>785</td>
<td>5</td>
</tr>
<tr>
<td>V7-V6</td>
<td>1697</td>
<td>0</td>
<td>121</td>
<td>2</td>
</tr>
<tr>
<td>V8-V7</td>
<td>126</td>
<td>23</td>
<td>1330</td>
<td>4432</td>
</tr>
<tr>
<td>V9-V8</td>
<td>900</td>
<td>1028</td>
<td>138</td>
<td>-4</td>
</tr>
<tr>
<td>V10-V9</td>
<td>106</td>
<td>157</td>
<td>3760</td>
<td>35165</td>
</tr>
<tr>
<td>V11-V10</td>
<td>2762</td>
<td>694</td>
<td>3827</td>
<td>18</td>
</tr>
<tr>
<td>V12-V11</td>
<td>6959</td>
<td>8848</td>
<td>1329</td>
<td>10104</td>
</tr>
<tr>
<td>V13-V12</td>
<td>-36</td>
<td>49</td>
<td>1877</td>
<td>8406</td>
</tr>
<tr>
<td>V14-V13</td>
<td>399</td>
<td>975</td>
<td>4703</td>
<td>36238</td>
</tr>
<tr>
<td>V15-V14</td>
<td>4314</td>
<td>475</td>
<td>38</td>
<td>21666</td>
</tr>
<tr>
<td>V16-V15</td>
<td>-205</td>
<td>4</td>
<td>2902</td>
<td>326</td>
</tr>
<tr>
<td>V17-V16</td>
<td>-1149</td>
<td>275</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>V18-V17</td>
<td>392</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V19-V18</td>
<td>578</td>
<td>406</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V20-V19</td>
<td>116</td>
<td>2028</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V21-V20</td>
<td>2439</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V22-V21</td>
<td>689</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.5 IV-Conservation of Organizational Stability (Invariant Work Rate)

Conservation of organizational stability states that “the average effective global activity rate on an evolving system is invariant over the product life time” [25]. In essence it deals with the activity rate of developers working for the development of a software system (See Table 2.1). This law predicts that the average activity rate of developers remains invariant over the software system life-cycle. As suggested by Lehman et al. [25], activity rate can be calculated based on the number of changes per release. We calculated the average number of changes per day for a release_i using the total number of changes in release_i divided by the days in between the release_i and release_{i-1}. The relation we used is Equation 3.4. Thus calculated activity rates of each sample mobile apps are plotted in boxplots 5.7 and 5.8.

As depicted in box plot of single core developer projects in figure 5.7, the
activity rate is not constant. The activity rate variation in Github-android and AntennaPod is wide, whereas the activity rate variation for the projects Android-autostarts, Vanilla Player, and SMSSync is relatively narrow but far from being invariant. Based on this observation, we conclude that conservation of organizational stability (or invariant work rate) does not hold for all single core developer mobile applications.

Figure 5.8 depicts the activity rate of multiple core developers porjects. In figure 5.8, ReLaunch shows wide variation in activity rate then that of projects Terminal Emulator, MobileOrg and Anki-android. Even though MobileOrg and Terminal Emulator shows a relatively narrow variation in activity rate, they are not invariant. Moreover they have outliers as shown in the box plot 5.8. This
Figure 5.8: Activity rate of mobile applications with multiple core developers

observation suggests that the law of conservation of organizational stability is not valid for multiple core developer projects as well.

One peculiar behavior we observed in this study is that majority of the mobile projects have released more than one versions in a single day. More specifically, out of 9 sample projects 5 projects (55.56%) have released more than one versions in a single day, making the activity rate infinite for such releases. This phenomena has made the further statistical analysis impossible without reasonable assumptions about the infinite activity rate.
5.6 V-Conservation of Familiarity

Conservation of familiarity states “During the active life of an evolving program, the content of successive releases is statistically invariant” [25]. To study this law, we calculated change rate of LOC using equation 3.5. The details of metrics and equations used to calculate change rate is described in sub-section 3.4.3. With the change rate data of each release of each sample mobile apps, we plotted graphs as shown in figures 5.9 and 5.10. The figures show change rate in LOC along y-axis, and time in days since the first release along x-axis.

We performed quadratic correlation analysis on the change rate of each applications using equation 3.2, where $y$ denotes change rate (in %) and $x$ denotes time in days since the first release. The number of sample points ($N$), resulted quadratic coefficient value ($a$), Pearson’s correlation coefficient ($R^2$) and p-value of change rate are shown in table 5.5.

<table>
<thead>
<tr>
<th>Applications</th>
<th>N</th>
<th>Quadratic Coeff.($a$)</th>
<th>$R^2$</th>
<th>P-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android-autostarts</td>
<td>24</td>
<td>0</td>
<td>0.0099</td>
<td>0.9004</td>
</tr>
<tr>
<td>Vanilla Player</td>
<td>19</td>
<td>0.0001</td>
<td>0.1558</td>
<td>0.258</td>
</tr>
<tr>
<td>Github-android</td>
<td>13</td>
<td>0.0012</td>
<td>0.0864</td>
<td>0.6364</td>
</tr>
<tr>
<td>AntennaPod</td>
<td>34</td>
<td>0.0007</td>
<td>0.0698</td>
<td>0.3257</td>
</tr>
<tr>
<td>SMSSync</td>
<td>14</td>
<td>0</td>
<td>0.02063</td>
<td>0.8917</td>
</tr>
<tr>
<td>ReLaunch</td>
<td>20</td>
<td>0.0014</td>
<td>0.1743</td>
<td>0.1963</td>
</tr>
<tr>
<td>Terminal Emulator</td>
<td>21</td>
<td>0.0001</td>
<td>0.0900</td>
<td>0.4081</td>
</tr>
<tr>
<td>MobileOrg</td>
<td>17</td>
<td>0</td>
<td>0.0623</td>
<td>0.6387</td>
</tr>
<tr>
<td>Anki-android</td>
<td>16</td>
<td>0</td>
<td>0.0099</td>
<td>0.9369</td>
</tr>
</tbody>
</table>

*Table 5.5:* Quadratic correlation on change rate of LOC of all sample mobile applications

Figure 5.9 shows the change rate of single core developer mobile applications and figure 5.10 shows the change rate of multiple core developers. According to graph all the mobile applications have shown continuous change since their first release. The change rate have shown rather random behavior than any statistical trend. Moreover, from the table 5.5, the correlation coefficient values($R^2$) of change rate are very few for all applications. Therefore, the statistical study for this law is unreliable. Based on these observations, the law of conservation of familiarity in this study is inconclusive.
Figure 5.9: Change rate of LOC of mobile applications with single core developer
Figure 5.10: Change rate of LOC of mobile applications with multiple core developers
Chapter 6
Characteristics of Mobile Applications

To answer research question RQ2 *How does the size of development team influence mobile applications evolution?*, we divided the question into three subquestions (See Section 3.2). The sub-research questions were formulated based on the testable characteristics of mobile applications described in Chapter 1. Each sub-research questions listed below are answered in Sections 6.1, 6.2 and 6.3 respectively. Analysis and discussion on the obtained results for each sub-research questions is in Chapter 7.

**RQ2** How does the size of development team influence mobile applications evolution?

**RQ2.1** Do the time between releases correspond with the team size?

**RQ2.2** Do the growth pattern of developers corresponds with the growth pattern of application size?

**RQ2.3** Are major releases created by more developers and minor releases by few developers?
6.1 Time between releases

The research question \textit{RQ2.1 Do the time between releases correspond with the team size?} requires time between releases in days of each application. The required data is directly extracted from the repository itself as described in Subsection 3.4.2.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{time_between_releases}
\caption{Time between releases of mobile applications with single core developer}
\end{figure}

Figure 6.1 shows the box plot of time between releases of single core developer mobile applications. Figure 6.2 shows the box plot of time between releases of multiple core developer mobile applications.

Summary of statistical analysis on the time between releases data of each application is shown in Table 6.1. First column in the table shows the list of mobile
Characteristics of Mobile Applications

Figure 6.2: Time between releases of mobile applications with multiple core developers

applications under this study. Second and third column shows the minimum and maximum values, fourth column lists the median value of each mobile applications time between releases. Similarly, fifth and sixth column represents median and standard deviation values of each mobile applications time between releases.

From the table 6.1, it can be seen that five mobile applications (that is, 55.56% of the total sample apps) has a median of less than 15 days between the releases. These applications are Android-autostarts, Vanilla Player, Github-android, AntennaPod, ReLaunch. This indicates that majority of mobile applications has short release cycle. Moreover in their extremities, five mobile applications have released two versions in a single day, making time between releases a zero. These
### Applications

<table>
<thead>
<tr>
<th>Applications</th>
<th>Min.</th>
<th>Max.</th>
<th>Median</th>
<th>Mean</th>
<th>S.D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android-autostarts</td>
<td>2</td>
<td>234</td>
<td>13</td>
<td>47.96</td>
<td>66.30</td>
</tr>
<tr>
<td>Vanilla Player</td>
<td>0</td>
<td>260</td>
<td>12</td>
<td>39.95</td>
<td>66.19</td>
</tr>
<tr>
<td>Github-android</td>
<td>0</td>
<td>271</td>
<td>6</td>
<td>32.31</td>
<td>74.19</td>
</tr>
<tr>
<td>AntennaPod</td>
<td>0</td>
<td>200</td>
<td>6</td>
<td>14</td>
<td>34.14</td>
</tr>
<tr>
<td>SMSSync</td>
<td>7</td>
<td>307</td>
<td>29.5</td>
<td>74.71</td>
<td>94.19</td>
</tr>
<tr>
<td>ReLaunch</td>
<td>0</td>
<td>44</td>
<td>6.5</td>
<td>9</td>
<td>10.11</td>
</tr>
<tr>
<td>Terminal Emulator</td>
<td>0</td>
<td>359</td>
<td>27.5</td>
<td>45.27</td>
<td>76.99</td>
</tr>
<tr>
<td>MobileOrg</td>
<td>2</td>
<td>205</td>
<td>39</td>
<td>66.53</td>
<td>63.43</td>
</tr>
<tr>
<td>Anki-android</td>
<td>1</td>
<td>299</td>
<td>46.50</td>
<td>84</td>
<td>89.58</td>
</tr>
</tbody>
</table>

*Table 6.1: Calculated values of mean, median and standard deviation from time between releases of each mobile applications*

Mobile applications are *Vanilla Player, Github-android, AntennaPod, ReLaunch, Terminal Emulator*. This observation makes the result *inconclusive*.

#### 6.2 Growth of application with respect to growth of developers

The research question *RQ2.2 Do the growth pattern of developers corresponds with the growth pattern of application size?* deals with the correlation between the growth applications size in LOC to the growth of developers (cumulative authors involved in the project). We calculated the cumulative growth in developers involved in each release of each mobile applications.

Figures 6.3 and 6.4 shows the growth of cumulative number of authors contributing in the particular project over time in days of sample mobile applications. X-axis in the graph shows time in days while Y-axis shows growth in cumulative number of authors of that project. The resulted growth of LOC of each applications were due to the number of developers contributing in those applications. Therefore, we mapped the total number of developers contributing in that application and total growth of LOC increased by those developers. The research question RQ2.2 is formulated to study whether these two aspects have similar pattern or not. That is, if the growth rate of team have similar pattern compared to growth rate of the applications than the stated research question’s answer is positive.

Table 6.2 provides number of releases, quadratic coefficient (a), correlation coefficient ($R^2$), p-value and the growth pattern indicated by the value of quadratic coefficient (a). For identification of authors growth pattern, we have used the quadratic equation 3.2, where $y$ denotes cumulative number of authors of each application and $x$ denotes time in days of the application since its first commit in git.
Figure 6.3: Cumulative authors growth trend of projects with a single core developer repository. For each value of $a$ in the quadratic equation 3.2: Range: Sublinear $< a <$ Superlinear

where,

$a = -0.05$ to $+0.05$ (assumption)

$a =$ Linear

From Table 6.2, it can be seen that all the applications posses linear pattern.
Figure 6.4: Cumulative authors growth trend of projects with multiple core developers

<table>
<thead>
<tr>
<th>Applications</th>
<th>N</th>
<th>Quadratic Coeff. ((a))</th>
<th>(R^2)</th>
<th>p-value</th>
<th>pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android-autostarts</td>
<td>24</td>
<td>0</td>
<td>0.5598</td>
<td>0.0001</td>
<td>Linear</td>
</tr>
<tr>
<td>Vanilla Player</td>
<td>19</td>
<td>0</td>
<td>0.9231</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>Github-android</td>
<td>13</td>
<td>0</td>
<td>0.9482</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>AntennaPod</td>
<td>34</td>
<td>0</td>
<td>0.7765</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>SMSSync</td>
<td>14</td>
<td>0</td>
<td>0.9754</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>ReLaunch</td>
<td>20</td>
<td>0</td>
<td>0.792</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>Terminal Emulator</td>
<td>21</td>
<td>0</td>
<td>0.9872</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>MobileOrg</td>
<td>17</td>
<td>0</td>
<td>0.9734</td>
<td>0</td>
<td>Linear</td>
</tr>
<tr>
<td>Anki-android</td>
<td>16</td>
<td>0</td>
<td>0.9866</td>
<td>0</td>
<td>Linear</td>
</tr>
</tbody>
</table>

Table 6.2: Quadratic correlation of cumulative authors of each mobile applications

As a result, if we compare the growth pattern of LOC and cumulative number of developers of all mobile applications than we can confirm that five applica-
tions shows similar patterns. These applications are Android-autostarts, Vanilla Player, Terminal Emulator, MobileOrg and Anki-android. This indicates that the majority of applications shows correspondence between the growth pattern of developers with the growth pattern of mobile applications code base. But other four applications does not show any correspondence, therefore the result is inconclusive.

6.3 Major/ minor releases and developer involvement

To answer research question RQ2.3 Are major releases created by more developers and minor releases by few developers?, we have to cull the major and minor releases of the applications. The major version releases are released when the functions of the applications become incompatible with the change of API\(^1\). As a result, lots of work need to be done to make the changes in the application. Therefore, assuming that the major version releases are created by big teams and minor or patch version releases are created by small teams, we further analyzed the number of authors contributing in respective releases.

We found that most of mobile applications does not contain major releases. The applications which contain major releases are ReLaunch, SMSSync, Anki-android, Github-android and Android-autostarts. In ReLaunch, major version release was not created by big team. In Github-android and Android-autostarts applications, the major release were the initial release. Therefore, assuming that the total number of contributors at that point might be the big team, this study supports that the major version of both the applications were created by big teams. Which means other than ReLaunch application, the applications containing major releases were created by big teams. Since, the applications containing major releases are less in number, it cannot be confirmed that mobile applications containing major releases are created by big teams or vice-versa. Therefore, the result is inconclusive in this case.

\(^1\)See http://www.vogella.com/articles/Git/article.html#versioncontrolssystems
Chapter 7

Analysis and Discussion

There are a number of studies on open source as well as commercial software system evolution, making it a well established field in software engineering. Despite the popularity of mobile applications in recent years, published studies on mobile applications evolution are non-existent. Hence this study attempts to fill an important gap in the existing knowledge base within mobile application engineering.

The results presented in Chapter 5 and 6 are further analyzed and discussed section wise in this chapter. Some of the important findings and learned lesson of this study are:

- All sample mobile applications supports Lehman’s law of I-Continuous Change
- Majority (6) of the sample mobile applications supports the law of VI-Continuous Growth while 3 of them do not
- Among sample mobile applications, one has super-linear, five has linear, and remaining three has sub-linear size growth pattern
- Due to discontinuous growth pattern of three sample apps (Github-android, SMSSync and ReLaunch), III-Self regulation is partially supported, despite the presence of ripples (growth and stabilization cycle) in incremental changes of all sample apps
- Due to weak correlation, statistical analysis on change rate is unreliable, hence V-Conservation of familiarity is inconclusive
- Law of II-Increasing Complexity is inconclusive, because using LOC growth pattern as complexity indicator may not be a good choice in open source mobile applications
- IV-Conservation of organizational stability is not supported, because not a single applications has constant activity rate
- Correspondence between the growth pattern of developers and the growth pattern of application size was inconclusive
Mobile applications code base can range from 2KLOC to 70KLOC

- The contributions on mobile applications development projects are significantly higher for a single or few developers

- Time between releases is less in small team size mobile applications in comparison to large team size applications

- Only few mobile apps have major releases, hence we studied all releases (major/minor) equally in testing Lehman laws

### 7.1 Continuing Change

In [33], authors measured growth with respect to binary size, number of files, LOC and uncommented non blank-LOC in two small scale open source software system. While measuring growth with respect to size, LOC and uncommented non blank-LOC they found their study to be continuously changing. In [18], authors used LOC and function metrics (number of methods, number of class and number of public methods) to find the growth of two medium sized open source software and found their study to be continuously growing. Xie et al. [36], used uncommented non blank-LOC and number of modules to measure growth of 7 open source software system written in C language. In both the cases, they found the law to be supportive. [16] analyzed growth of Linux using different approaches. They measured number of system calls, number of configuration options, LOC and number of functions and found that the software systems were continuously changing. In [7] studies total number of dependencies, unique dependencies, total SLOC, SLOC in plug-ins considering only the .java files having dependencies on Eclipse, total NOC(.java) and NOC(.java) with dependencies on Eclipse to investigated growth in 21 Eclipse third-party plug-ins. They found the law to be supportive in all cases.

Despite the variations in metrics used to characterize software projects, all of the studies were supportive towards the law of Continuing Change. Similarly, this study also found that the law to be supportive in open source mobile applications. Moreover, it can be stated that no matter whether the applications developed by single or multiple core developers: the mobile applications undergo continuous change from its first release. A reason for such behavior can be, none of the contributors would like to abandon the application which have incurred lots of cost in terms of time and effort unless, it is no longer feasible to maintain it.

Though all the mobile applications seems to have continuous change in the respective applications, the changes made in some versions of some applications seems to be risky. For example, changes in some versions of applications such as: Anki-android, MobileOrg and Android-autostarts were found risky. Following are
the steps that should be followed while planning changes such that the risks can be reduced [26].

1. The changes that have done for the releases must be updated in the documentation.

2. The changes accumulated in the new release should address the change that is desired in the application.

3. All the reasons under consideration that made changes / explicit assumption adopt to change design / implementation in applications, should be documented so that future release plan can be made easily. The other reasons might be issues, the kind of implementation done in design / algorithms etc.

4. During any release, previous assumptions made in previous releases must be reviewed to prevent from inconsistencies with the previous set of rules.

5. Besides adopting applications to environment with feature addition, releases focusing in defect removal, structural clean ups and performance enhancement will be needed in order to maintain the applications.

6. While applying changes in any release, model regarding the effect of change should be developed. Project contributers must be aware of the effect of change, whether its safe / risky / very risky to release with the changes accumulated.

7. The noticeable additions or change to requirements must be tracked in each release or in certain interval of time to know which part of the applications were being frequently modified. This will help the project contributers to decide the weak spot of the application. Then, developers can focus on restructuring the application. Developers should be careful while performing restructuring of the application so that none of the application’s functionality will left behind.

### 7.2 Increasing Complexity

As quadratic correlation model fits better than the linear correlation while study software growth pattern [15], we have decided to use quadratic model as well. We have used LOC growth pattern as an indicator of complexity in software projects.

Robles et al. [32] have used linear correlation to investigate the law of increasing complexity. Roy et al. [33] have investigated 2 small scale open source software system. Their study used different approach for each system. One system was measured using total number of global function, variables, macros against release
date and another against release sequence number. In both the studies the law was supportive. In [16], author used McCabe cyclomatic complexity to measure the complexity and found that the control measure was taken to reduce the complexity in Linux kernel. Xie et al. [36] have used average number of function calls per function, McCabe’s cyclomatic complexity and common coupling to measure complexity and found the law was supportive. In [18] authors computed coupling between the classes, response for a class and weighted method per class to measure complexity and found both the open source software support the law. S. Koch [20] has found that in open source software systems, large team size system shows super-linear growth pattern compared to small team size. Also, project having large number of LOC exhibit super-linear pattern compared to projects having less LOC. However, this study found that the open source mobile applications developed by multiple core developers does not have super-linear growth pattern and not necessarily possess largest number of LOC compared to single core developer applications.

In all the sample mobile applications evidence of complexity control mechanisms were present. Though the applications undergo complexity control, some of the application still have declining growth. For example, despite of complexity control mechanism performed in ReLaunch, Github-Android and AntennaPod mobile applications, still the application’s growth start to decline. The reason might be because of complexity presence in other aspects of application such as: complexity in requirements specifications, architecture, design and implementation etc. The reason may be more resources were allocated to control complexity that reduced resources for the application enhancement and growth of the application. Also, might be due to lack of conscious effort to reduce complexity that resulted in declining growth [26].

To prove or disprove the law, study should be done in detail. If the evolving system exhibit increase in complexity than it should follow sub-linear pattern. If the system does not exhibit increase in complexity then it might be because of proper measure taken to reduce the complexity. So, in both the cases the law stands true. On the other hand, open source projects do not have stable organizational structure (See results of Section 5.5). This phenomenon makes it harder to directly validate the law using only the growth pattern in open source projects.

Despite the instability of organizational structure of open source projects a number of researchers have studied growth pattern to indicate complexity of software projects. Some of these studies are [11, 20]. Though their results are not directly comparable to the Lehman laws, they have attempted to compare the prediction of Lehman’s law of increasing complexity. In this light, we conclude that the law of increasing complexity is inconclusive in our sample open source mobile applications.

Complexity control includes removing dead / repetitive code, restructuring, updating documentation etc [26]. According to Lehman et al. [26], effort for
complexity control is largely anti-regressive. The effort for complexity control must be balanced. More effort for complexity control, such as code refactoring etc. can reduce the effort towards application enhancement and growth of the application and result in declining growth and vice-versa. Therefore, complexity control should be planned in each release or in sequences of releases.

7.3 Self Regulation

This study uses incremental change in LOC as a metric to analyze the law of self regulating. In this study, we found partial support for the law of self regulating.

Xie et al. [36], analyzed incremental module growth of 7 open source projects. Their study confirmed the law to be supportive for all the projects. Businge et al. [7], analyzed the incremental growth of the dependencies for each plug-ins for Eclipse IDE. Their study also confirmed the law to be supportive. Israli et al. [16], analyzed incremental change in number of files and found their study indirectly support for the existence of self regulation. Johari et al. [18], observed the growth curve of two open source software and found the steady increase in size. Their study also confirms the law of self regulation to be supportive.

For law of self regulation, Lehman stated [26], if the incremental changes in LOC are less than or equal to mean \((m)\) than the growth potential is safe and lead to success. But if the increment change in number of LOC are greater than mean and less than \(m + 2s\) than the release is risky. The next release might be followed by code clean up. Even though the next release is not planned to do some code clean up, its better if for next few releases code clean up is done to stabilize the application releases. Similarly, if the increment change in number of LOC near to \(m + 2s\) or greater will be very risky release leading to instability of the application for one or more release. The releases must be followed by major code clean up focusing on defect fixing or performing some anti-regressive work. The value \(m + 2s\) is indicated as a threshold, exceeding which might lead to instability of the application and can delay the release date.

This study further analyzes the applications to know whether those ripples are caused by the same reasons as suggested by Lehman. In ReLaunch, the 11th release was in between \(m\) (mean) and \(m + 2s\) (mean plus two times the standard deviation), therefore in 12th release developers performed code clean up and then after it can be seen that it start evolving normally. Similarly in 15th release, developers performed some code clean ups than after the next releases were seen evolving normally in gaps of release dates. Similar behavior were found in other applications as well except in Android-autostarts application. In Android-autostarts application, the last 4 releases were found different than other applications. In its 20th release developers added some feature and fixed some of the bug. While in 21st release, ”find in market” option is replaced by ”find in appstore”. It seems that they have released the version for Apple AppStore.
Similarly, in 23rd release, they have released the version only for vodafone apps/elect. The behavior of last 4 releases were different than other applications. The releases seems to be very risky [26], because when they released the version for other platforms, lots of code were being changed that lead to high increase in LOC.

On the other hand, Lehman's law states, change in operational domain for user satisfaction leads to increase in complexity and system starts to deteriorate. The successive versions of Android-autostarts were released in long gaps as assumed by Lehman [26]. Therefore, from the analysis, it can be seen that the Android-autostarts application is in very risky mode. For which, contributors of Autostart application should be aware of and should promptly start focusing on restructuring the application.

In Anki-android and MobileOrg, number of later releases were seen above mean value, which according to Lehman is risky [26]. From qualitative analysis, it was found that in both the applications, code clean up was not performed. However, some bug fixes could be seen. Despite the bug fixes, it seems that application releases were continuously in risky mode. That is, still the applications are not stable. Developers should start working on control mechanism to reduce the complexity, in order to make the applications evolve for long term.

### 7.4 Conservation of Organizational Stability (Invariant Work Rate)

In [36], authors used two metrics to check the law of conservation of organizational stability. The average number of changes per day was calculated by using number of total changes in particular release divided by time between two successive releases. The other metric used was total number of functions added and changed divided by total number of functions. Their study found, using both metrics, not supportive to the law. Our study also used similar metric, which again resulted against the law. In [32], authors found that their study of large open source software systems does not support the law. However, their study was different from the Lehman's study. In [16], authors focused on development version which is frequently released with continuous activity by developers. Their study found the law to be supportive.

Our result is consistent with the argument that open source projects do not have stable organizational structure because contributors flexibility in working behavior. On the other hand, from the commits per author graph in figures 4.1 and 4.2, it is clear that a few and relatively constant number of developers contribute significantly to the open source projects. This phenomenon suggests a relatively stable organizational structure can be a possibility in open source projects.
7.5 Conservation of Familiarity

Businge et al. [7], considered both change rate and growth rate. Their study statistically concluded the law to be not confirmed from the data they considered. Xie et al. [36] suggest that the changing rate and growing rate are neither constant nor decreasing. However, the result was not verified statistically rather it was based on the visual inspection of graphs. Barry et al. [2], considered only growth rate. Their study found the result was decreasing and also statistically significant. Similarly, this study also considered changed, added and deleted LOC. This study also resulted in accordance to [7]. Lawrence [21] used net module growth as a metric and concluded that the result is statistically random. Roy et al. [33] considered number of files as a metrics and found both changing rate and growing rate tend to decline over time. Isreli et al. [16] found that their study supports the law of conservation of familiarity.

The reasons for not supporting the law of Conservation of Familiarity in open source mobile applications can be numerous. It might be because of the huge demand in change in the applications. One of the reason might also be due to changes needed to be made the view point of developers. However, Lehman suggested that applications start to show declining growth in long run. Lehman did not suggested any reasons about the declining growth of the systems in long run. Lehman suggested the law by viewing the systems in long run and found maximum number of systems growth to be declining. According to Lehman, in the long run of software evolution, complexity in the system increases, which force the system growth to decline. Also, according to Lehman, rather than progress work / active planning , there is always improvement / waiting queue to be sorted out [26]. However, in this study, the exact reason behind not supporting Lehman’s law is still unknown.

As Paulson et al. [30] stated, earlier the defect found and fixed in the application, the changing rate will start to decrease. Similarly, in this study, only few applications were found decreasing in changing rate. Therefore, this study finds that the law of Conservation of Familiarity is inconclusive in our sample mobile applications.

7.6 Continuing Growth

In [33], while measuring growth with respect to size, LOC and uncommented non blank-LOC they found continuous growth. Whereas, while measuring with respect to number of files they found that in one system growth was continuous and in another system number of files remained constant in later releases. Therefore, they suggested that in small scale system measuring growth with uncommented non blank-LOC may or only be the best metrics. All the open source Android mobile applications in this study have less than 100KLOC. Therefore, assuming
Chapter 7. Analysis and Discussion

the applications having less than 100KLOC as small scale system, this study also used size metric as LOC and found the law to be partially supportive. Three applications (Github-android, SMSSync and ReLaunch) have decreased LOC in some releases with respect to prior releases.

Israeli [16] analyzed growth of Linux using different approaches. They measured number of system calls, number of configuration options, LOC and number of functions. Among which they found continuous growth with linear growth trend except while measuring growth with number of configuration options in which they found super-linear growth trend. Xie et al. [36] used uncommented non-blank-LOC and number of modules to measure growth of 7 open source software system written in C language. In both the cases, they found the law was supportive. [7] study total number of dependencies, unique dependencies, total SLOC, SLOC in a plug-in considering only the .java files having dependencies on eclipse, total NOC(.java) and NOC(.java) with dependencies on eclipse to investigated growth in 21 Eclipse third-party plug-ins. They found the law to be supportive in all cases. In [18], authors used LOC and function metrics (number of methods, number of class and number of public methods) to find the growth of two medium sized software and found the growth was mix of linear and sub-linear in both the cases. In [32], authors used linear model to find growth of large open source software system. They found the law to be supportive.

7.7 Characteristics of Mobile Applications

To suggest the factors that are affecting the open source mobile applications in observed agreements and deviations to the Lehman’s laws, we developed a number of research questions as listed in RQ2 (See Section 3.2) and the details results are presented in Chapter 6. Some of the interesting characteristics we found are:

- The mobile applications are developed by a single developer or a few developers, despite the huge numbers of contributors presence. This indicates the possibility of having head or lead programmer in open source mobile applications, and consequently the presence of relatively stable organizational structure in terms of number of active developers contributing towards a project.

- Time between release of the sample mobile applications is not constant. But the time between release of the small team size projects were relatively lower than that of the large team size projects. This indicates that the releases of small team size projects are more frequent than the releases of large team size projects.

- The correspondence of the contributors growth pattern and the application size growth pattern was inconclusive as only 3 projects shows correspon-
dence. This indicates that there is equal possibility of having large teams with smaller applications as well as small teams with bigger applications.

- Majority of our sample mobile applications do not have major releases. Only 5 applications has one or a few major releases. This indicates that the mobile applications are released more frequently with simple bug fixes and maintenance. This is the reason for considering all the releases equal, which makes at least 11 data points for effective correlation analysis.
This thesis performed evolution studies on 9 different Android open source mobile applications hosted at www.f-droid.org. Git commands were used to extract the historical data from git repositories. For this study, LOC count is used as a size metric (dependent variable) and time in days (since first release) as independent variable. Out of eight Lehman’s laws of software evolution, 6 were tested for validation.

After the results, analysis and discussion of mobile applications evolution study, table 8.1 concludes our results in the context of published software evolution studies in commercial and open source projects.

<table>
<thead>
<tr>
<th>Lehman laws</th>
<th>Commercial</th>
<th>Open-source</th>
<th>Mobile apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>I-Continuing change</td>
<td>Supportive</td>
<td>Supportive</td>
<td>Supportive</td>
</tr>
<tr>
<td>II-Increasing complexity</td>
<td>Supportive</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>III-Self regulation</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
<td>Partial support</td>
</tr>
<tr>
<td>IV-Conservation of organizational stability</td>
<td>Supportive</td>
<td>Inconclusive</td>
<td>Not supportive</td>
</tr>
<tr>
<td>V-Conservation of familiarity</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
<td>Inconclusive</td>
</tr>
<tr>
<td>VI-Continuing growth</td>
<td>Supportive</td>
<td>Supportive</td>
<td>Partial support</td>
</tr>
</tbody>
</table>

Table 8.1: Comparision of mobile apps evolution with the published open source and commercial software evolution

The contributions on our sample mobile applications was significantly greater of a single or a few developers, despite a huge number of contributers presence. Time between releases of small team size mobile applications were found less compared to large team size mobile applications. The correspondence between the growth pattern of developers and the growth pattern of application size was inconclusive (only 5 projects shows correspondence). Only few applications have major releases in their evolution lifetime.
Future work of this study will be to perform extensive validation of Lehman laws on large number of mobile applications from both commercial as well as open source domain. Such study could use a number of alternative metrics such as number of files, number of modules, time in days, release sequence numbers, etc. Also, other characteristics of mobile applications can be considered, such as ported or non-ported and single or multi-platform mobile applications to check whether these characteristics may have any influence on the Lehman laws of software evolution. Such studies will help researchers and practitioners in the mobile software engineering field to understand how exactly mobile applications having different characteristics evolve over time and consequently proper majors can be suggested if any maintenance to be done. In this way, long term evolution of mobile applications can be discussed and implemented in future mobile application releases.

This study sample consists of open source mobile applications developed for Android platform only, one obvious extension of this study would be to conduct similar studies on commercial mobile apps as well as open source apps on other competitive mobile platforms such as iOS, Tizen, Symbian, etc.

As open source mobile applications do not have stable organizational structure, use of LOC growth pattern may not accurately indicate the predictions of law of Increasing Complexity. On the other hand the commits per author charts in figures 4.1 and 4.2 shows a few number developers contribute significantly more than other developers. This phenomenon suggests the possibility of having lead or head programmer organization structure in open source mobile application projects. In this light, a more rigorous study using different complexity metrics in open source mobile applications can be a valuable future work.

Since this is the only study done explicitly to find any agreement / deviation from Lehman’s law in mobile applications known so far, more of similar kind of studies is needed in future to establish and generalize the results of this study.


References


