The Evolution of Mobile Apps: An Exploratory Study

Jack Zhang, Shikhar Sagar, and Emad Shihab
Rochester Institute of Technology
Department of Software Engineering
Rochester, New York, USA, 14623
{jxz8072, fxs1203, emad.shihab}@rit.edu

ABSTRACT

As mobile apps continue to grow in popularity, it is important to study their evolution. Lehman's laws of software evolution have been proposed and used to study the evolution of traditional, large software systems (also known as desktop apps). However, do Lehman's laws of software evolution hold for mobile apps?, especially since developing mobile apps presents different challenges compared to the development of desktop apps.

In this paper, we examine the applicability of three of Lehman's laws on mobile apps. In particular, we focused on three laws: the law of continuing change, increasing complexity, and declining quality. We extracted a number of metrics and performed a case study on two applications: VLC and ownCloud. Our findings show that the law of continuing change and declining quality seem to apply for mobile apps, however, we find different outcomes for the law of increasing complexity. Then, we compare the mobile app version to the desktop version and find that the two versions follow the same trends for the law of continuing change. On the contrary, the desktop and mobile version have different trends for the law of increasing complexity and the law of declining quality.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures*

General Terms

Software evolution

Keywords

Software evolution, mobile software engineering, mobile applications

1. INTRODUCTION

The popularity of mobile apps is exponentially growing. These mobile apps are inherently different than traditional desktop apps. While previous research has focused on the differences between mobile and desktop apps from a programming point of view, to the best of our knowledge, there has not been much research that examines the difference between mobile and desktop apps from an evolutionary point of view.

In 1974, Lehman and Belady proposed a set of laws pertaining to software evolution [1]. Lehman's laws were stated based on empirical observations of commercial software. Much time has passed since then, but many of Lehman's laws still apply to this day despite the evolution of technology in general [2].

With the introduction of smartphones, a new market for software applications has popped up. These mobile applications are different from the traditional desktop applications [3]. The development of these applications is also different. Could the evolution of these applications be different as well? Do we have to worry about the increasing complexity of mobile applications? Is it necessary for mobile applications to continually change in order to survive? Are mobile applications the special exceptions to these Lehman's laws?

Much research has been done with Lehman's laws in terms of evolution of large desktop applications. Lehman's laws represent the dynamics of software development for the lifetime of a piece of software. However, Lehman's laws have not been observed in mobile applications. In this paper, we investigate the applicability of Lehman's laws of software evolution on mobile apps. In particular, we focus on following 3 laws:

1. **Continuing Change:** Software has to continually change if the software is to remain useful
2. **Increasing Complexity:** Unless complexity checks are in place, software will get more complex over releases
3. **Declining Quality:** As the software evolves, the quality in software drops

To conduct our study, we use four applications namely - the VLC desktop application, VLC for Android, the ownCloud desktop application, and ownCloud for Android. First, we extracted software evolution metrics to examine whether Lehman's laws apply to mobile apps. Second, we compare the evolution metrics for Android versions of the applications with their Desktop counterparts in order to determine the similarities or differences between the mobile and desktop apps. In particular, we focus on answering the following research questions:

RQ1 Do Lehman's law of continuing change, increasing complexity, and declining quality apply to mobile apps?

RQ2 Do Lehman's laws apply to mobile apps and their desktop version in the same way?

DeMobile '13, August 19, 2013, Saint Petersburg, Russia

In Section II we review related work. In Section III, we discuss our case study setup. In Section IV, we present our results. In Section V, we give an account of our experiences and limitations of the study. In Section VI, we summarize our work.

2. RELATED WORK

A plethora of previous work focused on the evolution of desktop applications (e.g., [4–6]). However, to the best of our knowledge, there have not been any studies that examine the applicability of Lehman's laws on mobile applications.

One of the initial studies on Evolution in Open Source Software was performed by Michael Godfrey *et al.* [6]. First they defined the difference in the development process for a commercial system vs. Open Source Software (OSS). Then, they measured the Linux kernel at the system and subsystem level. They collected Lines of Code (LOC) for stable and development releases with respect to time, and found the increase in growth rate of the Linux kernel, both at system and subsystem level. In our case study, we also use LOC as one of our primary metrics to measure different aspects of the software systems.

Yi Wang *et al.* [7] provided a set of metrics to evaluate the development process of OSS. The study considered specific properties of the open source community and the essential role played by the members in the evolution of OSS. They conducted a lightweight case study on the Ubuntu project. Using their metrics they found the number of modules, number of developers, number of bugs in a specific moment, number of fixed bugs, and the ratio of fixed bugs to existing bugs metric useful in assessing the evolution of OSS. They also analyzed modules which they considered as the atomic unit in their analysis. But, their metrics model was limited to modules and was based on Open Source Communities. In this paper, we considered the number of bugs, number of fixed bugs, and the ratio of fixed to exisiting bugs metric used in their study to observe the laws of software evolution in mobile applications.

Businge *et al.* [8] study software evolution using Lehman's laws on plug-ins in Eclipse. They used metrics such as the number of changes in lines, classes, and dependency metrics to examine the applicability of Lehman's laws. Our work complements the prior work done in the past by examining whether or not mobile apps follow Lehman's Laws. The main focus of our paper is to compare the desktop and mobile versions of the same OSS in order to find similarities/differences in the evolution patterns in these applications.

3. CASE STUDY SETUP

3.1 Data Sources

To perform our study, we selected 4 projects: VLC for Android, the VLC desktop version, ownCloud client for Android, and the ownCloud desktop client. VLC is a popular media player that is well known for supporting a wide variety of media formats. OwnCloud is a cloud-storage medium similar to Dropbox. The ownCloud projects we are examining are actually client software used to access cloud storage. We extracted the source code and commit history for these four projects from their Git repositories.

We picked these four projects for the following three reasons:

1. We looked for applications that are different in terms of their popularity. VLC is a well-known program used by many users. The development community is also very active, numbering in a total of 511 developers altogether working on it. ownCloud is a less-known application with a small development community, numbering in a total of 31 developers working on it.

2. Another important factor is the amount of data available for a project. These projects had a development history of at least 2 years. In addition, most tools that calculate software metrics (e.g., the Understand metrics tool [9]) only support certain languages. This becomes problematic when it comes to examining the Android applications, in particular because Android applications commonly use a mix of languages like Java, Javascript, XML, and PHP. The source code for these four projects has been examined, and we have determined the languages the projects used could be covered by the Understand metrics tool (the languages being Java, C, C++, XML, and JavaScript).

3. Third, we selected projects where both the mobile and the desktop applications were developed by the same company. During data selection, we have seen a variety of applications where the the mobile application was a port of the desktop application developed by a third party. Even though these other projects are OSS, we felt that the process would be too different and cause our data to become unreliable. By enforcing the criteria that mobile and desktop applications must be developed by the same company, we found only a few applications that fit the criteria (and had a long enough commit history).

3.2 Data and Metric Extraction

To examine each of Lehman's Laws, we defined a set of metrics to measure the different aspects of software evolution. To calculate the metrics, we leveraged the repository data for each of the four applications. In particular, we obtained the commit history of the different projects from their respective Git repositories. To get the commit history, we wrote a script that first gets all the unique commit IDs from the project's Git repository. Then, for each commit ID, we obtained the full log of that commit and stored it as a text file. Next, we filter out the commits so that only commits within a specified date are considered (the exact dates considered for each project are stated later). Then, we wrote scripts that calculate the churn, the number of commits, number of bug-fixing commits (i.e., commits that contain words associated with a bug fix), and the number of feature commits (i.e., commits that are not associated with bug fixing) and the number of hunks (i.e., different parts of a file touched) in a commit. To calculate the number of hunks, we wrote a script that uses the built-in diff tool supplied by Git to compare two different commits. The output of the diff tool was a text file showing the hunks of code that are different. The script would detect the number of lines that contained the symbols "@@", where that particular symbol indicated the beginning of the hunk. By counting the number of lines that contained the "@@" symbol, we were able to calculate the number of hunks between two release dates.

3.3 Evolution Metrics Associated to Lehman's Laws

Lehman's Law for Continuing Change: A program that is used in a real-world environment must necessarily change, or else become progressively less useful in that environment. This law states that the existing software needs to keep changing over time, or else the software will become obsolete. The continuous change in the software accounts for feature additions, bugs fixing and platform changes. To stay in the fast growing business world, it is critical to provide changes and keep the software bug-free. We used three metrics to measure this law: code churn, feature commits and number of hunks.

Table 1: Metric Data based on Dates

	Data Point	A1	A2	A3	A4	A5	A6	A7	A8
VLC Android	Lines of Code	160	5,773	18,085	21,256	199,909	415,882	438,442	467,476
	Number of Files	35	57	114	138	818	1,630	1,736	1,816
VLC Desktop	Lines of Code	719,413	735,205	767,287	778,393	778,393	788,621	790,864	796,357
	Number of Files	56,718	56,793	57,069	57,225	57,224	57,243	56,654	56,663
ownCloud Android	Lines of Code	11,345	11,995	16,451	41,484	55,346	56,895	N/A	N/A
	Number of Files	125	145	181	313	393	450		
ownCloud Desktop	Lines of Code	3,641	10,984	15,050	20,275	23,064	25,017	N/A	N/A
	Number of Files	60	117	178	258	287	329		

Table 2: Metric Data based on Date Ranges

	Data Point	A1-A2	A2-A3	A3-A4	A4-A5	A5-A6	A6-A7	A7-A8
VLC Android	Code Churn	4,189	26,379	22,197	137,286	114,839	29,524	31,121
	Total Commits	92	98	115	266	464	557	330
	Feature Commits	15	15	30	44	65	87	45
	Bug-Fix Commits	77	83	85	222	399	470	285
	Number of Hunks	38	99	130	810	1424	1,003	660
VLC Desktop	Code Churn	98,286	466,876	3,417,944	638,691	165,702	169,574	101,667
	Total Commits	1,096	1,714	2,023	1,944	1,371	1,515	1,256
	Feature Commits	258	401	477	542	376	379	366
	Bug-Fix Commits	838	1,313	1,546	1,402	995	1136	890
	Number of Hunks	2,697	4,820	14,629	1,9821	3,801	5,443	3,823
ownCloud Android	Code Churn	9,967	2,5811	63,900	108,600	42,208	N/A	N/A
	Total Commits	22	63	265	178	130		
	Feature Commits	5	10	73	42	36		
	Bug-Fix Commits	17	53	192	136	94		
	Number of Hunks	55	82	223	433	551		
ownCloud Desktop	Code Churn	30,227	28,045	198,180	355,950	108,309	N/A	N/A
	Total Commits	119	193	215	390	273		
	Feature Commits	36	45	41	51	35		
	Bug-Fix Commits	83	148	174	339	238		
	Number of Hunks	44	82	217	294	555		

1. *Code Churn:* Code Churn is the number of lines added, modified or deleted in a piece of code. Churn provides the degree to which a given source code file has changed over time. Code churn can be measured by processing revisions in a version control system and counting the total number of lines changed [10]. We used code churn to look at the changes made in the software. Presence of code churn refers to the fact that the software system is continuously changing.

2. *Feature Commits:* Feature commits is measured as the number of commits made by each developer to the source code that add features. To calcualte this metric, we used extract the total number of commits. Then we determined the bug-fix commits using keywords *'fix', 'workaround', 'crash', 'bug', 'error', 'exception', 'handle', and 'handling'*. Finally, we filtered the bug-fix commits from the total number of commits to get the feature commits. The feature commits metric shows the additions being put into the software system, i.e., the software system is continuously changing by adding new features over time.

3. *Number of Hunks:* The Git repository has a feature which uses a diff command to provide a chunk of changed code. The diff command extracts the number of areas, where the piece of code was changed between two commits. This is called a hunk. We extracted the total number of hunks between two dates.

Lehman's Law of Increasing Complexity: states that unless work is done to maintain or reduce complexity, the software will become more complex as the software evolves. The majority of the prior work use different code complexity metrics to measure complexity, however, the majority of the complexity metrics are highly correlated with lines of code and other complexity metrics. Therefore, we came up with different complexity metrics that are not correlated with each other to help us examine the law of increasing complexity.

1. *Lines of Code:* Prior work showed that lines of code is highly correlated with code complexity metrics. Therefore, we extracted the total Lines of Code (LOC) and Source Lines of Code (SLOC).

2. *Commits per File:* The total number of commits shows the changes made in the software system. By calculating the average commits per file, we are able to normalize the number of commits, since we expect larger files to have more commits. The higher the average number of commits per file, the more changes are being made at the file level. This metric serves as an indicator of software complexity.

Lehman's Law for Declining Quality: states that the quality of the system appears to decline unless it is rigorously adapted, as required, to take into account changes in the operational environment [11]. The law refers to the state of the software system, where quality is a function of several aspects. It is difficult to empirically measure the quality of a software system, as it can be relative. Therefore, to measure declining quality, we use the number of bug fixing commits.

1. *Bug Fix Commits:* A piece of software that is low in quality will be prone to more bugs and defects. Although counter-intuitive, for this law to hold true, we should see on average more Bug Fix Commits. If there's more bug fixes, then there

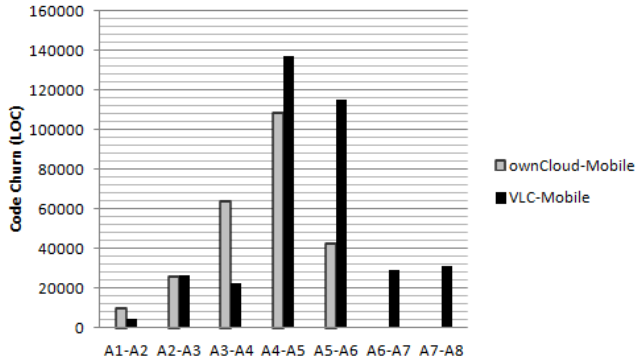

Figure 1: Code Churn for ownCloud and VLC

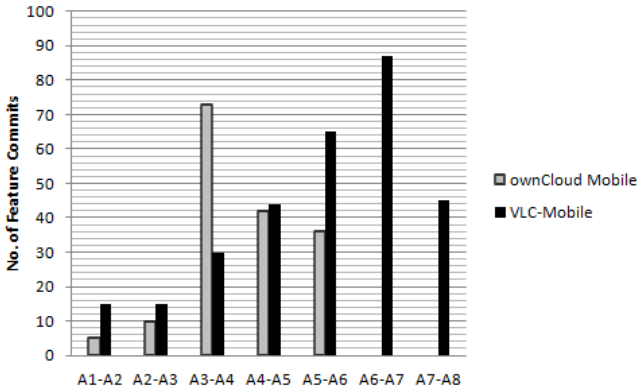

Figure 2: Feature Commits for ownCloud and VLC

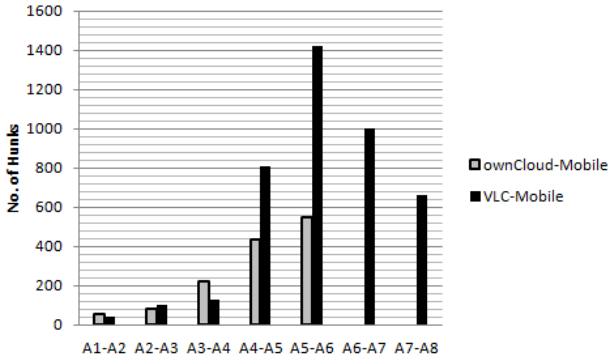

Figure 3: No. of Hunks for ownCloud and VLC

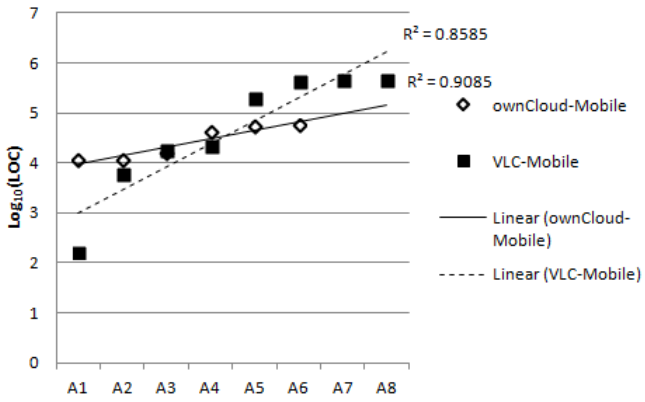

Figure 4: LOC for ownCloud and VLC

are more bugs identified in the system. To detect bug fix commits, we used a script to search through all the commits to a project and keep the files that have the following bug fix keywords: fix, workaround, crash, bug, error, exception, handle, and handling.

The results of our data collection are shown in Tables 1 and 2. Table 1 presents the metrics data that can be captured by a single date, whereas Table 2 presents the metrics data that can only be captured by date ranges. In order to improve readability, we replaced dates with data points in the tables. Data point A1 presents the initial date of 8/19/2011 for both ownCloud applications, and the date 11/15/2010 for both VLC applications. The subsequent data points were calculated by adding 4 months to the previous data point. So, A2 is the date four months after A1, and A3 is the date four month after A2 and so on. In Table 2, we observe the data that could only be captured in the interval between two dates. No data exists for ownCloud on data point A7 and A8 because VLC is older than ownCloud.

We have deem the use of data points suitable for two reasons. First, the time between data points are equivalent (which is a period of 4 months). Second, the dates captured for both Android and Desktop versions of the applications are within 10 days of each other. For both Android and Desktop in VLC, the dates captured are exactly the same. For both Android and Desktop in ownCloud, the dates were on average about 7 days apart. The beginning date was determined by the initial commit date of the mobile applications. The subsequent dates were determined by the adding four months to the previous date. The last data point for all applications represents the most recent commit date (within 4 months) found in the source code repository.

4. RESULTS

After collecting the data for both research questions, we analyzed each one of the metrics and answered the research questions. For each law, we plotted graphs between the metrics and the time interval. For the scatterplot graphs, we also fit a linear line and calculated R^2 values. The linear line represents the general trend of a relationship, while the R^2 values indicate how close the values of the scatterplot follow the trend. The closer R^2 is to one, the better the fit of the scatterplot points to the line.

Do Lehman's laws apply to mobile apps and their desktop version in the same way

RQ1: Do Lehman's law of continuing change, increasing complexity, and declining quality apply to mobile apps?

To examine the **law of continuing change**, we looked at the code churn, feature commits and number of hunks. From Figures 1 , 2 and 3, we observed the mobile apps are changing constantly over time. The code is added, deleted or modified regularly, which is reflected by the code churn value. The fact that code churn exists mean the system is changing over time. There is also a constant increase in the feature commits, which implies that new features were regularly added in the software. The number of hunks are also increasing over time for both the apps. All the three metrics provide evidence that the mobile apps do follow Lehman's law of continuing change.

For the **law of increasing complexity**, we used LOC and the average number of Commits/File metrics. From the Figures 4 and

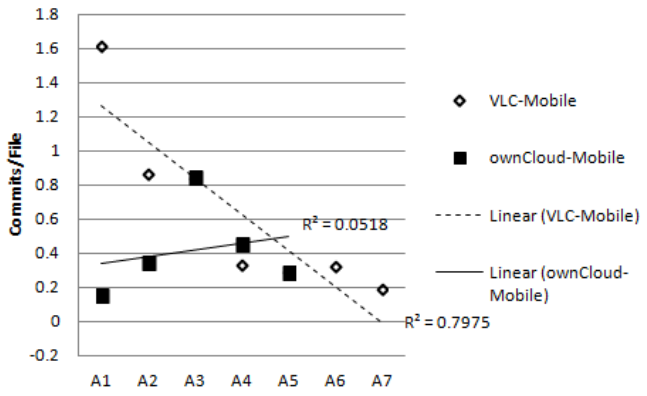

Figure 5: Commits per File for ownCloud and VLC

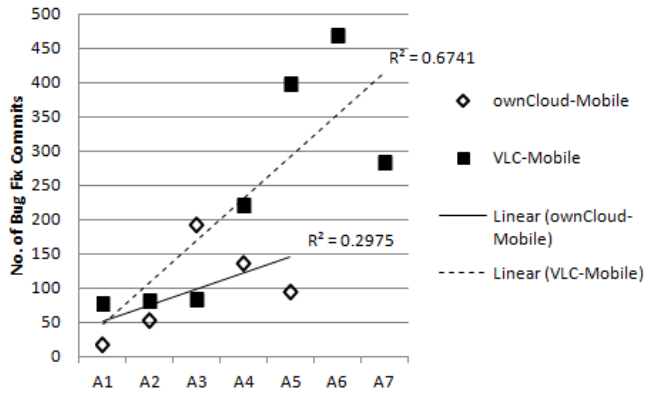

Figure 6: Bug Fix Commits for ownCloud and VLC

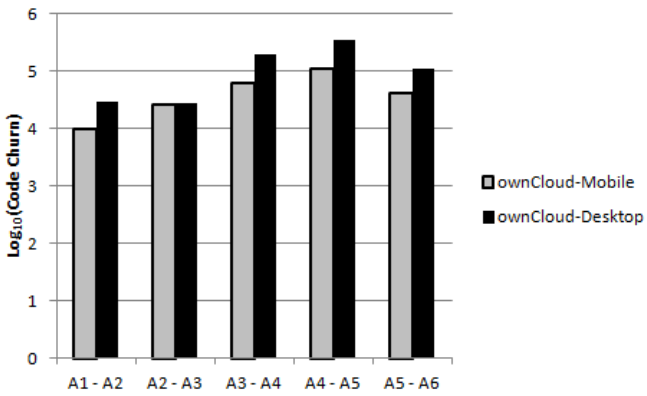

Figure 7: Code Churn for ownCloud over 24 months

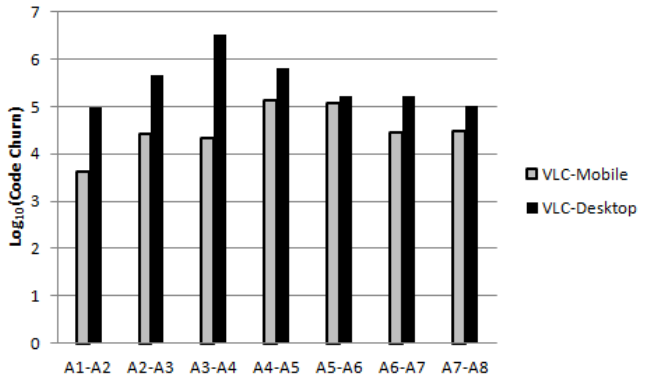

Figure 8: Code Churn for VLC over 32 months

5, we observe an increase in LOC for both the applications. Although the rate at which the VLC mobile app increases is more than the ownCloud app, the complexity is increasing over time in both the mobile applications. But when we closely observed the commits/file 5, we noted a high correlation between the two metrics. According to figure, we found the average commits/file is decreasing over time for the VLC mobile app, whereas in the ownCloud app, there was an increase in complexity initially and a decrease later on. Since the result of the commits/file metrics is different for the different apps, we can not analyze the complexity of the system. However, the increase in LOC implies that the complexity in the system is increasing over time.

For the **law of declining quality**, the number of bug-fix commits is used. In Figure 6, the number of bug-fixing commits for both mobile apps is increasing. The number of bug-fixing commits shows a constant rate of bug fix commits, there is a steep increase in both the apps, followed by a sudden decrease. From this metric, it is challenging to analyze the quality of the app. Based on this analysis, we observe that perhaps the law of declining quality does hold for mobile apps. That said, we believe that using more metrics and performing this analysis on more apps is needed to have stronger confidence in out conclusions.

To answer this research question, we compare the evolution metrics for both the desktop and mobile apps. First, we examined the **law of continuing change**. For code churn, we take the logarithm of the actual code churn values so we can compare the trends of the desktop and mobile apps side-by-side (the logarithm was taken since the mobile values were much lower). From Figures 7 and 8, we observe that the code churn between the mobile and desktop versions show the same trend, where the number of code churn rises and falls. We also observe the same kind of trends for the number of hunks as illustrated in Figures 9 and 10.

But perhaps the more interesting trends to observe are the number of feature commits. From Figures 11 and 12 we see different trends. The mobile version of these graphs show a peak for the number of feature commits, while the desktop versions show a stable trend for the number of feature commits introduced at each date. From these graphs we can conclude that all applications induce change over time, following the law of continuing changes. In terms of the hunks and churn, the software system even follows the same trends. At the commit level, however, mobile versions exhibit more rapid changes when compared to their desktop counterparts.

When examining the **law of increasing complexity** for mobile and desktop applications, we observe some differing relationships. After observing Figures 13 and 14, we can see that the LOC for both graphs are showing increasing trends. The rates of increase are different, but they are increasing nonetheless.

RQ2: Do Lehman's laws apply to mobile apps and their desktop version in the same way?

The relationships observed for both mobile and desktop applications when observing the average number of commits per file (shown in Figures 15 and 16) are different. For one, the values are too sporadic to tell us anything. For the law to hold true, we should see an increasing trend, the idea being the more commits are made per file, the more complex the system is getting due to

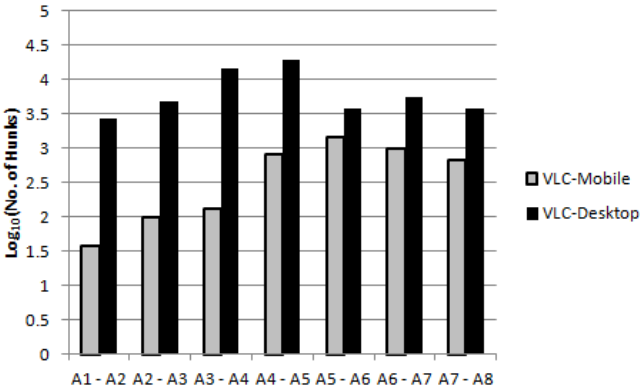

Figure 9: Number of Hunks for VLC over 32 months

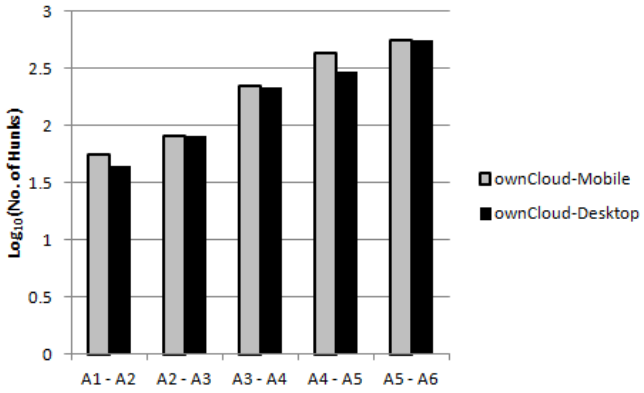

Figure 10: Number of Hunks for ownCloud over 24 months

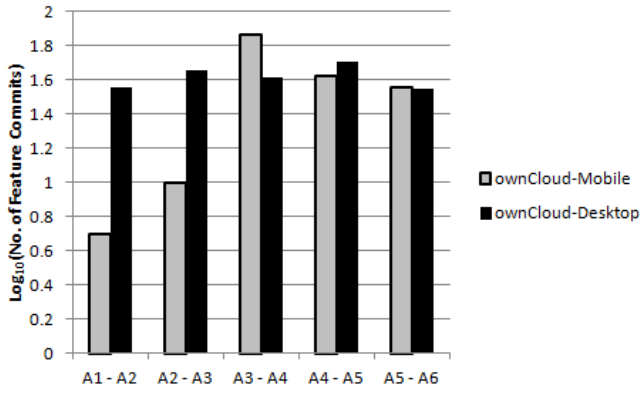

Figure 11: Number of Feature Commits for ownCloud over 24 months

repeated changes to the system. Instead, we see various spikes. We believe the fluctuations are caused by the number of files in the system. The changes in the number of files are not constant trends, so the randomness of these metrics mixed with the randomness of the commits themselves will cause great fluctuations in the graph. Therefore, we cannot conclude whether the mobile and desktop applications follow the law of increasing complexity.

Instead, we can conclude the development for mobile applications and the development of desktop applications are different

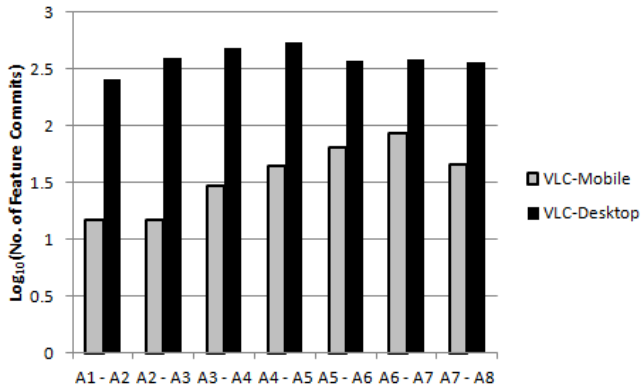

Figure 12: Number of Feature Commits for VLC over 32 months

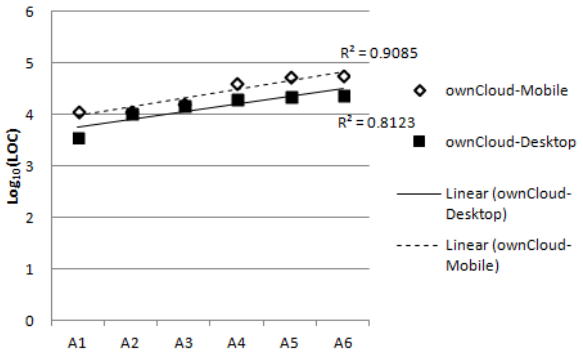

Figure 13: Lines of Code for ownCloud over 24 months

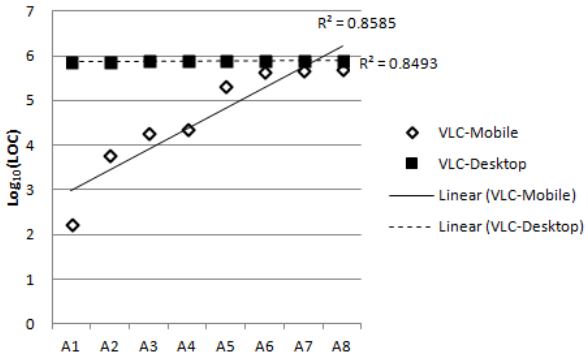

Figure 14: Lines of Code for VLC over 32 months

when measured from a complexity perspective. In LOC, the mobile applications show rapidly increasing trends of LOC while the desktop versions show a trend towards a constant trend of LOC. Furthermore, despite the unreliability of Figures 15, and 16, we can still see the trends for mobile applications differing from the desktop applications.

Also, for the **law of declining quality** the mobile applications and desktop applications show different trends. The bug fix commits in Figures 17 and 18 show fluctuations of increasing and decreasing number of bug fixes. We believe this is attributed to the fact that we considered bug fixing at a commit level. We did not consider a situation where a single bug-fix commit fixed many bugs.

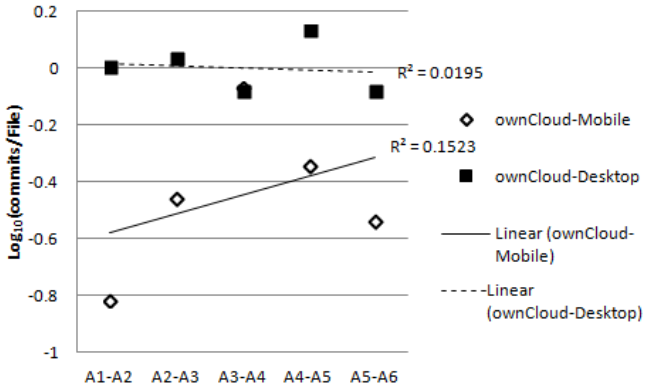

Figure 15: Commits per File for ownCloud over 24 months

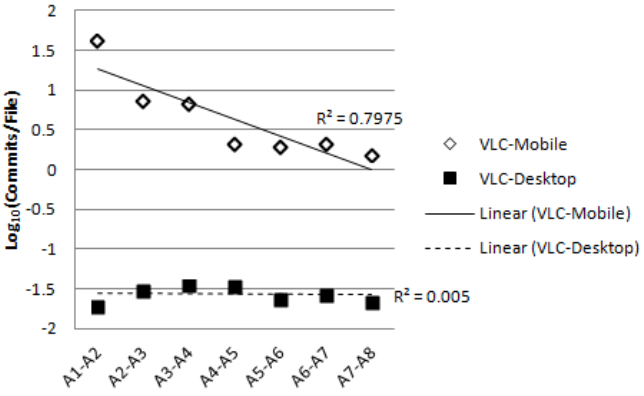

Figure 16: Commits per File for VLC over 32 months

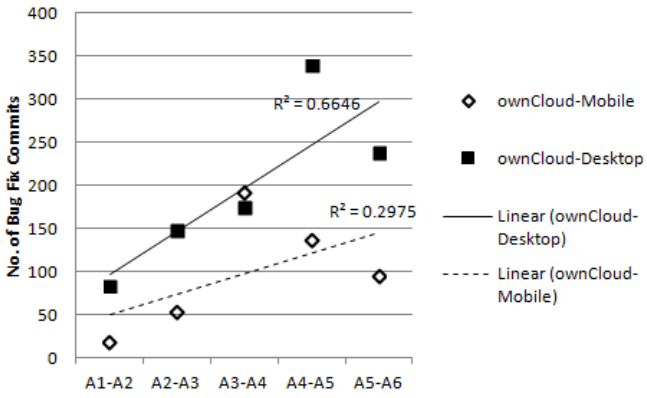

Figure 17: Bug Fix Commits for ownCloud over 24 months

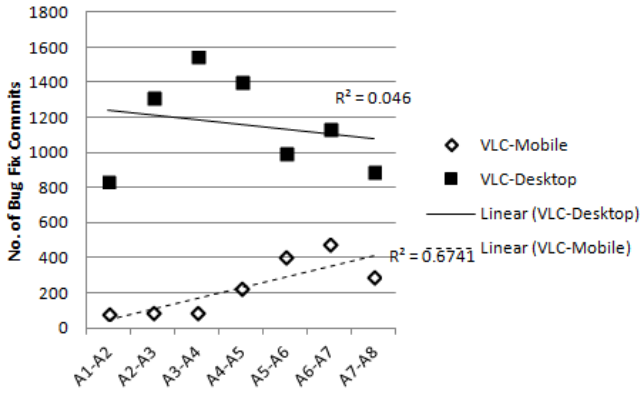

Figure 18: Bug Fix Commits for VLC over 32 months

As with the metrics used for the law of increasing complexity, we cannot conclude as to whether the applications follow the law of decreasing quality. Regardless, whether these metrics are a good indicator of declining quality does not change the fact that there is a difference in the evolution of software quality between mobile application and desktop applications.

5. EXPERIENCES AND LIMITATIONS

When we initially started our research, we planned to use a number of complexity metrics to measure the effects of Lehman's Laws (which is what the majority of prior work did). These metrics include Coupling Between Objects, Cyclomatic Complexity, Halstead Difficulty, and Halstead Effort. After doing some correlations, we found that these metrics were highly correlated with one another in our case study projects. That means the metrics we were going to apply to all laws would have measured the same thing and we really only needed one complexity metric. In the end, we decided to use LOC to measure complexity, and discarded all other metrics that were correlated with LOC.

We also considered a number of process metrics, but a problem we often ran into were the mobile projects are still too young. For example, we have considered using the Backlog Management Index (BMI) [12] to measure software quality, which takes into account the number of closed bugs and total number of bugs identified in a release. The problem we ran into in this metric relied a lot on a good quality issue tracker. In most mobile projects we observed, an issue tracker is not created until half-year to a year after the initial commit was made. And even then, people do not really post bugs until much later. It is hard enough to find a mobile project with a long history, let alone a mobile project with a mature issue tracker.

One of the more interesting challenges we faced when collecting data was tool replacement. We have ran into many situations where project owners would switch source code repositories (e.g., from SVN to Git) and issue trackers in the middle of development. Because we had difficulties in accessing old repositories, we were only able to get the commit history of the most recent years despite the fact that the project has been around for much longer. Fortunately, for ownCloud and VLC, we collected data for at least 6 data points (at 4 month intervals), which allowed us to see the relationships between applications as indicated by the graphs.

Selecting the time interval and the date placement was a big issue in this research. The fact is mobile applications are recent. We could not find a sufficient amount of data right in the middle of a mobile project, so we had to include the beginning stages of mobile app development to get a sufficient amount of data. We initially wanted to collect data on release dates only, but the release schedules for mobile applications and desktop were different, one offering releases every two weeks while the other offered releases on arbitrary dates. This is why we decided to settle on a time interval of four months. For the desktop applications, it would seem logical to capture the first couple of months of development, and then map those time frames to the time frames of the mobile application development. The problem was both the VLC and ownCloud desktop applications recently migrated their source control to Git. While it

is technically possible to gather the legacy code from that time, it would have been too difficult and time-consuming to obtain. Surprisingly, despite the data collection for desktop applications starting in the middle of software development, we still see both similar and different trends in software development.

The software metrics we have chosen seem to be giving us mixed results. While some of the software metrics (i.e. Lines of Code) gives us predictable trends, other software metrics (i.e. number of bug fix commits) gave us trends that we did not expect. A possible explanation for these unexpected trends is the metrics we have chosen were wrong. In the future, we will look into the validity of the metrics we have chosen for this research as well as come up with a more appropriate set of metrics to measure the evolution of mobile applciations.

6. CONCLUSION

The results of this research gives insights to the applicability of Lehman's laws in comparison to mobile applications. For the **law of continuing change**, we found remarkably similar trends between mobile applications and desktop applications. For the **law of increasing complexity**, we see through the LOC metric that both mobile and desktop applications show an increase in complexity, even if the rate in which the system is getting more complex is different. Unfortunately, we are not able to conclude whether this law holds true due to the inaccuracies of our other complexity metrics. For the **law of declining quality**, we found that in terms of the number of bug fixing commits, the mobile apps also continue to decline in quality over time (i.e., the number is increasing over time). That said, we did observe a difference in trends between the desktop and mobile versions of the app. In our future work, we plan to focus our energies into selecting more appropriate metrics to re-observe these laws.

Our greatest findings in this research are the relationships in the evolution of mobile software applications and their desktop counterparts. If we are strictly to look at the evolution of the lines of code, and the lines of code that has changed, then we will see the same trends in development. If we were to look at evolution using process level metrics, we found that the development of the program exhibited different trends. For the next steps, we would like to see why these differences exist despite the similar evolution in structure to gain a deeper understanding of software evolution in general.

7. REFERENCES

[1] M. Lehman, "Programs, life cycles, and laws of software evolution," *Proceedings of the IEEE*, vol. 68, no. 9, pp. 1060–1076, 1980.

[2] G. Xie, J. Chen, and I. Neamtiu, "Towards a better understanding of software evolution: An empirical study on open source software," in *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009, pp. 51–60.

[3] R. Minelli and M. Lanza, "Software analytics for mobile applications–insights and lessons learned," *2011 15th European Conference on Software Maintenance and Reengineering*, vol. 0, pp. 144–153, 2013.

[4] K. Johari and A. Kaur, "Effect of software evolution on software metrics: an open source case study," *SIGSOFT Softw. Eng. Notes*, vol. 36, no. 5, pp. 1–8, Sep. 2011. [Online]. Available: http://doi.acm.org/10.1145/2020976.2020987

[5] G. Singh and H. Singh, "Effect of software evolution on metrics and applicability of lehman's laws of software evolution," *SIGSOFT Softw. Eng. Notes*, vol. 38, no. 1, pp. 1–7, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2413038.2413046

[6] M. W. Godfrey and Q. Tu, "Evolution in open source software: A case study," in *Software Maintenance, 2000. Proceedings. International Conference on*. IEEE, 2000, pp. 131–142.

[7] Y. Wang, D. Guo, and H. Shi, "Measuring the evolution of open source software systems with their communities," *SIGSOFT Softw. Eng. Notes*, vol. 32, no. 6, Nov. 2007. [Online]. Available: http://doi.acm.org/10.1145/1317471.1317479

[8] J. Businge, A. Serebrenik, and M. van den Brand, "An empirical study of the evolution of eclipse third-party plug-ins," in *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*. ACM, 2010, pp. 63–72.

[9] SciTools, "Understand: Source code analysis and metrics," http://www.scitools.com/, June, 2013.

[10] A. Meneely and O. Williams, "Interactive churn metrics: socio-technical variants of code churn," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 6, pp. 1–6, 2012.

[11] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, Aug. 1994. [Online]. Available: http://dx.doi.org/10.1109/2.303623

[12] L. Grammel, H. Schackmann, and H. Lichter, "Bugzillametrics: an adaptable tool for evaluating metric specifications on change requests," in *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting*, ser. IWPSE '07. New York, NY, USA: ACM, 2007, pp. 35–38. [Online]. Available: http://doi.acm.org/10.1145/1294904.1294909

[13] S. Apel, O. Lessenich, and C. Lengauer, "Structured merge with auto-tuning: balancing precision and performance," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 120–129. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351694

[14] R. Sindhgatta, N. C. Narendra, and B. Sengupta, "Software evolution in agile development: a case study," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, ser. SPLASH '10. New York, NY, USA: ACM, 2010, pp. 105–114. [Online]. Available: http://doi.acm.org/10.1145/1869542.1869560

[15] Owncloud source code repository. [Online]. Available: https://github.com/owncloud

[16] Vlc source code repository. [Online]. Available: http://wiki.videolan.org/Git