Models for Mobile Application Maintenance Based on Update History

Xiaozhou Li, Zheying Zhang and Jyrki Nummenmaa
School of Information Sciences, University of Tampere, Kalevantie 4, FI-33014, Tampere, Finland
{*xiaozhou.li, zheying.zhang, jyrki.nummenmaa*}*@uta.fi*

Keywords: Software Maintenance, Software Release, Mobile Application, Maintenance Model.

Abstract: Good software development and particularly maintenance practices form an important factor for success in software business. If one wants to constantly produce new successful releases of the applications, a proper efficient software maintenance process is the key. In this work, we study data from mobile application maintenance to understand and conceptualize how mobile application maintenance takes place. Based on the data on release history, we deduce different mobile application maintenance models from the perspectives of maintenance scheduling and maintenance requirements.

1 INTRODUCTION

Mobile devices have evolved into multi-functional mobile personal processors, with a wide variety of functionalities and software. Rapid development is often connected to fierce competition, which, conversely, is also the main driving force of development. So far, together with the development of the mobile device market, competition between mobile platforms has brouhgt the evolution of mobile devices and their software to a new level. The number of mobile applications has increased tremendously with the applications rapidly expanded into a wide range of domains.

Different from traditional software-intensive systems, mobile applications are often written specially to utilize the unique features a particular mobile device offers. Most applications were developed creatively satisfying unrevealed needs of users instead of complying with a specific client's expectations. It is not uncommon to change requests from users' feedback after the applications are brought to market. In addition, as the mobile platforms are evolving very quickly, an application needs some adaption or modification when new versions of operating systems are released. All these make the maintenance and evolution of a mobile application a critical process.

Good software development and particularly maintenance practices form an important factor for success in software business. Software maintenance in general is both time and money consuming (Boehm, 1984). If one wants to constantly produce new successful releases of the applications, a proper efficient software maintenance process is all-important. However, current research related to mobile application maintenance is still limited.

In this paper, we study data from mobile application maintenance to understand and conceptualize how actual maintenance occurs. The major hypothesis is update history of existing mobile applications could show processes that mobile applications share when being maintained, and methods for mobile application maintenance could be deduced from analysis of the collected update history data. Thus, based on the observed patterns, we propose three maintenance models. With the help of conceptualization of these initially implicit models, we expect that not only do we better understand the mobile application maintenance, but the models can also be explicitly used in conducting future mobile application maintenance. The update history data, including update time span and content, are collected from Apple app store.

The remainder of the paper is organized as follows. Section 2 gives some background on software maintenance terms and process description. Section 3 introduces our data collection method and analysis process. Section 4 presents the research findings of three maintenance methods for mobile applications. Section 5 contains discussion and conclusions.

2 BACKGROUND

2.1 Software Maintenance

The software maintenance phase covers the whole procedure ranging from the delivery of the software to its retirement. Martin and McGlure define software

maintenance as "Changes that have to be made to computer programs after they have been delivered to customer or user" (Martin and McClure, 1983). Software maintenance is also defined as "the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment"(IEEE, 1998). Software maintenance forms an increasingly important part in the development and evolution of software engineering theory and practice. However, it is also expensive and time-consuming. The modification of software product could take up a percentage ranging from 40% to 70% of the total cost of the software life cycle (Hunt et al., 2008).

There are four types of software maintenance practices, i.e. Corrective maintenance, Adaptive maintenance, Perfective maintenance, and Preventive maintenance. Corrective maintenance refers to acts in order to correct errors detected after software release. Adaptive maintenance is to adapt software to newly changed system environment (Swanson, 1976). Perfective Maintenance is to enhance existing features and improve the general performance of software (Singh and Goel,).

The fourth maintenance type, Preventive maintenance, defined as modification in order to prevent a software system from future crash and enhance the maintainability and credibility (Gerëtìsbakh, 1977) is of great importance as well, however, it will not be further discussed in this paper. This is because we assume that preventive practices, such as refactoring (Fowler, 1999), can not be seen as visible changes from users' view but should be carried out throughout the whole maintenance process. In addition, we define in this paper *maintenance requirements* as the requirements that after software release specify how the software should be updated. Maintenance requirements encompass the three types of maintenance and also requirements for additional features.

2.2 Mobile Application Maintenance

Under the background of current competition-intensive mobile application market, the daily increasing rate of new applications is tremendously high while numbers of applications aiming for the same user group and sharing the same functionalities. Thus the products that constantly providing continuous high quality and exciting new features from time to time will take the leading position and be recognized by more potential users. Concerning the improvement of software quality, methodologies are supposed to contribute to either the product itself or the process (Sommerville, 2007). Maintenance for mobile application is aiming to maintain the high quality and stable performance. The improvement of mobile maintenance process is therefore of great importance accordingly. To better organize the process of mobile application maintenance coping with the competition and unpredictability of the market, a maintenance schedule, which is defined as the continuous planning timetable for mobile application maintenance process, should be used as one of the tools.

Besides coping with the maintenance requirements of previously mentioned types, occasionally adding new features to existing software is of great importance to mobile applications. We define a new feature as a new type of service or new experience provided by the application, such as avatars, game themes, rewards, tasks, and so on. Despite of the fact that adding new features is somehow considered as part of perfective maintenance (Hatton, 2007), we emphasize the importance of adding new features in mobile application maintenance phase and consider it a unique maintenance type besides the existing four. Concerning the Kano's model of customer satisfaction (Berger et al., 1993), attractive requirements, which provide users unexpected new features, boosts user satisfaction most sharply. Additionally, providing one-dimensional features, results in proportional user satisfaction increase (Sauerwein et al., 1996). The importance of adding new features in mobile application maintenance phase is indisputable, and the maintenance process lasts till the software closedown.

Resulting from the variety of user feedbacks and unpredictability of market trends, the maintenance should be organized iteratively (Choudhari and Suman, 2010). Therefore, a well-organized maintenance schedule including a reasonable new feature updating frequency is important, and that schedule should be well planned for mobile application maintenance.

Considering maintenance process in general as one stage, that stage shall contain components or properties including objectives, inputs and outputs, and methods (Sommerville, 2007). A mobile application maintenance process can be simply depicted as what is shown in Figure 1. Before the actual maintenance process starts, certain input conditions must be clarified, taking into account possible influencing factors. For example, mobile application business models (Vannieuwenborg et al., 2012), objectives and market trends influence the choice of maintenance requirements. The situation of maintenance team and customers, on the other hand, affects the organization of maintenance schedule. Feedback from end users provides simultaneously maintenance requirements and schedule managing guidelines.

Taking all input information into account, the maintenance process is an iterative process (Bennett and Rajlich, 2000) of working on certain amount of maintenance requirements (the objectives), according to the maintenance schedule with a certain maintenance method (the method of managing maintenance requirements and schedules). Apparently, the outcome of maintenance work comprises application releases with updated content, recorded in update history. The recorded release dates suggest a maintenance schedule while update content shows which maintenance requirements are fulfilled. Thus, we are investigating the possibilities to deduce possible maintenance models based on the update history.

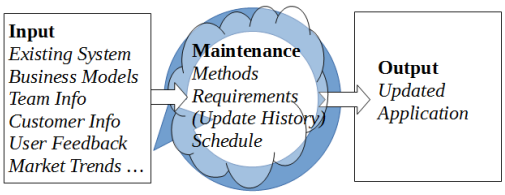

Figure 1: General Process Model of Mobile Application Maintenance.

3 COLLECTION AND ANALYSIS OF UPDATE HISTORY DATA

As a basis of our analysis, we will consider the realized maintenance schedules of mobile applications. We primarily try to combine the schedule information with the information on new features in the releases, in order to find maintenance patterns. Appropriately, the application update history of iOS(iPhone Operating System) applications provides information concerning the update content and update time of each application, which is more accessible from Apple app store than that from Google Play Market[1]. Thus, we collected the update history data of 100 mobile applications covering different categories on iOS platform. As shown in Figure 2, from the screenshot of the update history, it is easy to identify information concerning release date and update content.

The application market regions that we chose to investigate are the USA and China as the application store from the USA region contains by far the most available applications, and China has the most rapidly increasing number of applications on the Apple app store. However, how cultural factors influence the process of mobile application maintenance is not our current focus but belongs to the future work. Moreover, the fact that the two countries differ from each other in terms of customs and culture in many ways means that any findings or universal patterns found are not just a phenomena of a single country.

[1]https://play.google.com/store.

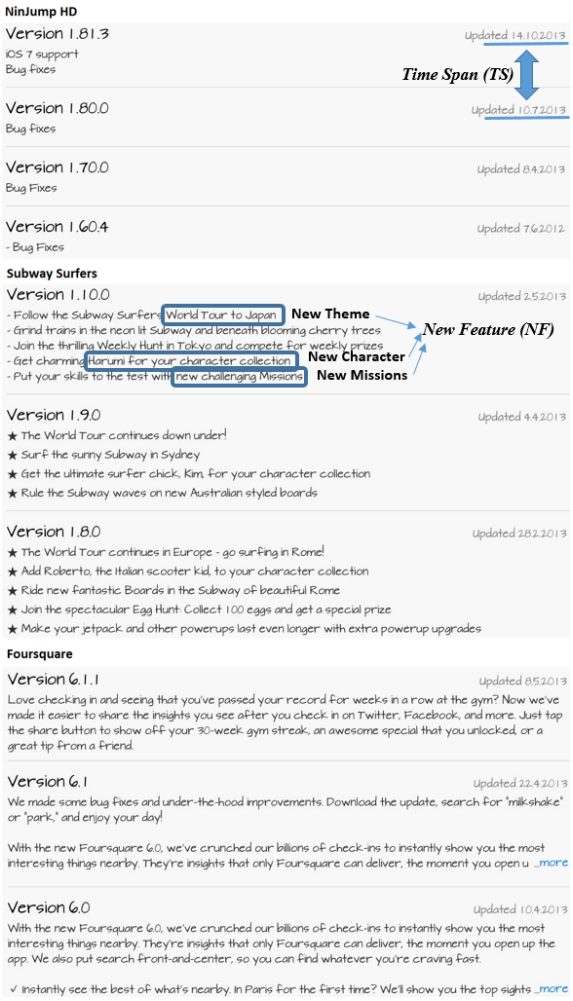

Figure 2: Update History Screenshots of NinJump HD, Subway Surfers, and Foursqure from iTunes.

Taking into account the validity of data, we selected applications that conform to the following restriction: the application should have at least five updates or has been delivered in Apple app store for at least two years if it has no less than five updates. When there were more than 15 updates after the first update in 2012, we considered only the updates since 2012. Based on the update history, the value of time span (TS) of each update was calculated as the difference of two consecutive release date. The time span standard deviation (TSSD) reflects the organizational level of the maintenance schedule.

By analyzing the update content we can find whether new features (NF) were implemented in a certain update. The aim of our data analysis was to find the correlation between the maintenance meth-

ods and the two parameters concerning the frequency of adding new features and the consistency of update time span. Thus, we used the value 1 for updates with new features implemented and 0 for updates without new features. The *Average New Feature Update Frequency* (ANFUF) is the arithmetic mean of assigned new feature update parameter NF (*NF = 0 or 1*) of selected updates. The collected data of NF and TS from the three sample applications is presented in Table 1. The data representing the part of update history in Figure 2 is marked in bold italic font.

Table 1: Update Data Examples.

NinJuam HD		Subway Surfers		Foursquare	
TS	NF	TS	NF	TS	NF
15	0	38	1	10	0
35	0	10	0	11	0
169	1	35	1	13	1
5	0	28	1	2	0
48	0	25	1	12	0
12	0	10	0	13	1
53	0	13	1	11	1
106	***0***	21	1	10	0
215	***0***	1	0	9	1
67	***0***	9	0	17	0
67	***0***	10	***1***	8	0
...	...	***25***	***1***	11	1
		20	***1***	15	0
		7	0	10	0
		13	1	19	***0***
		20	1	***8***	***0***
		...	...	***12***	***1***
				...	...

The formula for arithmetic mean is

$$ANFUF = \frac{1}{n}\sum_{i=1}^{n} NF, \; where \; NF = \; 0 \; or \; 1$$

The other variable is the *Time Span Standard Deviation* (TSSD), which is simply the standard deviation of all update time spans (x) as below.

$$TSSD = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}, \; where \; \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

When calculating time span, a business day calculation web application (Timeanddate, 2014) is used in order to exclude weekends and public holidays. By setting two variables for each mobile application, all selected applications could be located in chart with TSSD as x-coordinate and ANFUF as y-coordinate, which is presented as Figure 1.

We can make some initial observations directly from Figure 3. First of all, a majority of application samples locate in the division where TSSD is lower

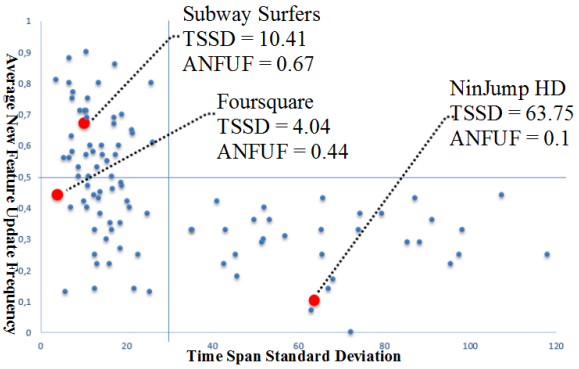

Figure 3: Statistics concerning the Relation of TSSD and ANFUF.

than 30 and ANFUF is higher than 0.5. It shows that the maintenance schedule of these applications are better organized with at least one new feature implemented within two updates. On the other hand, applications that locate in the down-right division have less organized maintenance schedule with fewer new features than other applications. They also have less than half of the total update amounts compared to the other applications. The other applications locate in the left-down division where TSSD and ANFUF are both low, which indicates these applications keep maintaining though without as many new features. In addition, no applications have both high TSSD and ANFUF as it seems not highly possible that maintenance schedule is poorly organized when many new features are updated simultaneously.

4 MAINTENANCE MODELS

As shown in Figure 3, we divide the whole area into four divisions by the lines of *ANFUF = 0.5* and *TSSD = 30*. Based on the analysis of the distribution of application samples and the details in update history, we deduce the following three mobile application maintenance models.

4.1 Emergency-oriented Maintenance

For mobile applications located in the down-right division of the figure, the maintenance is to deal with various types of emergent issues within maintenance phase, including bugs, system crashes, environment adaption, as soon as possible without considering the implementation of new features. The essential aim of their maintenance phase is to maximally maintain the stability of applications in all possible environments. We name this maintenance model Emergency-Oriented Maintenance. The process of Emergency-

Oriented Maintenance is to a large extent random and spontaneous. The whole maintenance process is more likely the extension of the debugging of previous development phase. Together with continuous testing, debugging and refactoring, as well as incoming user reviews, there are more potential bugs emerging. Thus, fixing bugs is of the highest priority compared with other maintenance requirements. When the maintenance starts, maintainers cover at least critical corrective maintenance requirements first, including some perfective maintenance requirements if it is possible. Additionally, whenever adaptive maintenance occurs, it should be promoted to the highest level. Correspondingly, the release date depends on the accomplished amount of bugs fixed and the degree of urgency.

One of the typical examples of conforming to Emergency-Oriented Maintenance is NinJump HD[2], an action adventure game. The update history of this game is presented in the first column of Table 1. Concerning the update time span of this application, it is obvious that the time spans range from one week to ten months, which is highly unorganized. On the other hand, there are no new features implemented in the whole maintenance process except for once in the second update. Nevertheless, the stability of the game is proven to be good according to the user feedback, with most criticism concerning the game idea, not reporting critical bugs.

4.2 Feature-oriented Maintenance

The applications in the left-up division contain a high new feature updating frequency, which shows focus on new features. Therefore, we name the maintenance model in this division Feature-Oriented Maintenance. Compared with Emergency-Oriented Maintenance, Feature-Oriented Maintenance is more organized concerning update time span. Despite of the fact that occasionally it takes more than one month to release an update version, the variance of the time spans is mostly lower than 30. The main aim of applying this maintenance method is to provide constant new features, in order to keep users attracted. Meanwhile, within each release, corrective and adaptive maintenance requirements are included as well, taking both stability and innovation into account. However, in the case of critical bugs, an emergency release will be delivered despite that it might to some extent deteriorate the well-planned maintenance schedule.

Subway Surfers[3] is a typical example of Feature-Oriented Maintenance. The update history of this game and an excerpt the screenshot is briefly presented in Table 1 and Figure 2. We could easily observe that except for bug fixing minor updates, each major update of this application contains one brand new theme as well as relevant other new items. Concerning the update span, each update takes approximately one month, which is much more organized than NinJump HD. In addition, continuously updating with new themes and features results in accordingly positive feedbacks and high popularity.

4.3 Constant Maintenance

The applications located in the left-down division have low time span variance without a high new feature implementation rate. The maintenance schedule is like in Feature-Oriented Maintenance, but with focus on the process organization than new feature implementation. We call this maintenance model Constant Maintenance. Constant Maintenance model is a regular but disciplined maintenance model that does not consider updating new features as the first priority. The empahsis is on constantly organized maintenance process. Compared to the other two maintenance models, this maintenance model is similar to the idea of applying agile principles (Beck et al., 2001) in maintenance phase. The whole maintenance process is constantly divided into maintenance iterations with durations of normally 10 to 20 days. Each of the maintenance iterations covers corrective maintenance requirements of highest priority as much as possible and occasionally perfective maintenance requirements and new features. Adaptive maintenance requirements are of high priority whenever appearing.

Constant Maintenance model requires no specific new features in each update as routines. New features could be added when it is needed and planned in a certain update. On the other hand, the time span of updates is required to be short. This model will result in a maximally organized maintenance process. However, the effort and cost spent on planning will be higher as well.

Foursquare[4] is a typical application in this category, with it's update history presented in the third column of Table 1. The update history shows that each update span lasts around 15 days. Each update implements all possible types of maintenance requirements as well as new features with certain defined prioritizing preference.

[2]http://www.backflipstudios.com/games/ninjump/.

[3]http://kiloo.com/games/subway-surfers.

[4]https://foursquare.com/.

5 CONCLUSIONS

This paper introduces three mobile application maintenance models based on the analysis of mobile application update history. We collected data to identify the new feature update frequency and standard deviation of update time span. By studying the distributions of these two variables for sample mobile applications, we observed the patterns of how existing mobile application maintenance works. The summarized three mobile application maintenance models form the basis of future research to study explicit and thorough definitions, instructions and guidelines, which could be further used in maintenance practice. The three proposed models provide promising ways to organize better maintenance schedules and maintenance requirements.

Our future research concerning mobile application maintenance will focus on constructing maintenance process model for mobile applications based on the three maintenance methods obtained from previous findings. Traditional software maintenance process models have been widely studied and utilized (Ren et al., 2011). Considering the defects of heavyweight maintenance practices and specific requirements of current mobile applications, an agile lightweight maintenance process model could be a more efficient and economical solution. Related research has been done concerning the result of introducing XP practices to software maintenance process. The positive results indicate it could be successful introducing agile practices into software maintenance process (Svensson and Host, 2005). Furthermore, concerning the construction of mobile application maintenance models, there are also more detailed research areas to be taken into account, including methodologies of eliciting maintenance requirements from user feedbacks, managing maintenance requirements, applying agile principles in mobile application maintenance practices, and so on. Further research should also include bigger samples with applications from other platforms.

REFERENCES

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunnimgham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Principles behind the agile manifesto. *Retrieved*, 11:2008.

Bennett, K. H. and Rajlich, V. T. (2000). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 73–87. ACM.

Berger, C., Blauth, R., Boger, D., Bolster, C., Burchill, G., DuMouchel, W., Pouliot, F., Richter, R., Rubinoff, A., Shen, D., et al. (1993). Kano's methods for understanding customer-defined quality. *Center for Quality Management Journal*, 2(4):3–36.

Boehm, B. (1984). Software engineering economics. *Software Engineering, IEEE Transactions on*, SE-10(1):4–21.

Choudhari, J. and Suman, U. (2010). Iterative maintenance life cycle using extreme programming. In *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, pages 401–403.

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Gerëtìsbakh, I. (1977). *Models of preventive maintenance*, volume 23. North-Holland Pub. Co.(Amsterdam and New York).

Hatton, L. (2007). How accurately do engineers predict software maintenance tasks? *Computer*, 40(2):64–69.

Hunt, B., Turner, B., and McRitchie, K. (2008). Software maintenance implications on cost and schedule. In *Aerospace Conference, 2008 IEEE*, pages 1–6.

IEEE (1998). Ieee standard for software maintenance. *IEEE Std. 1219-1998*, pages i –3.

Martin, J. and McClure, C. L. (1983). *Software Maintenance: The Problems and Its Solutions*. Prentice Hall Professional Technical Reference.

Ren, Y., Liu, Z., Xing, T., and Chen, X. (2011). Software maintenance process model and contrastive analysis. In *Information Management, Innovation Management and Industrial Engineering (ICIII), 2011 International Conference on*, volume 3, pages 169–172.

Sauerwein, E., Bailom, F., Matzler, K., and Hinterhuber, H. H. (1996). The kano model: How to delight your customers. In *International Working Seminar on Production Economics*, volume 1, pages 313–327.

Singh, Y. and Goel, B. A step towards software preventive maintenance. *SIGSOFT Softw. Eng. Notes*, 32.

Sommerville, I. (2007). *Software Engineering*. International computer science series. Addison-Wesley.

Svensson, H. and Host, M. (2005). Introducing an agile process in a software maintenance and evolution organization. In *Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on*, pages 256–264.

Swanson, E. B. (1976). The dimensions of maintenance. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 492–497, Los Alamitos, CA, USA. IEEE Computer Society Press.

Timeanddate (2014). Calculate the number of days between two dates @ONLINE.

Vannieuwenborg, F., Mainil, L., Verbrugge, S., Pickavet, M., and Colle, D. (2012). Business models for the mobile application market from a developer's viewpoint. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*.

Appendix of the mobile applications update history data available upon request.