User Experienced Software Aging:
Test Environment, Testing and Improvement Suggestions

Sayed Tenkanen

University of Tampere
School of Information Sciences
Interactive Technology
M.Sc. thesis
Supervisor: Professor Roope Raisamo
December 2014

University of Tampere
School of Information Sciences
Interactive Technology
Sayed Tenkanen: User Experienced Software Aging: Test Environment, Testing and Improvement Suggestions
M.Sc. thesis, 48 pages
December 2014

Software aging is empirically observed in software systems in a variety of manifestations ranging from slower performance to various failures as reported by users. Unlike hardware aging, where in its lifetime hardware goes through wear and tear resulting in an increased rate of failure after certain stable use conditions, software aging is a result of software bugs. Such bugs are always present in the software but may not make themselves known unless a set of preconditions are met. When activated, software bugs may result in slower performance and contribute to user dissatisfaction. However, the impact of software bugs on PCs and mobile phones is different as their uses are different. A PC is often turned off or rebooted on an average of every seven days, but a mobile device may continue to be used without a reboot for much longer. The prolonged operation period of mobile devices thus opens up opportunities for software bugs to be activated more often compared to PCs. Therefore, software aging in mobile devices, a considerable challenge to the ultimate user experience, is the focus of this thesis. The study was done in three consecutive phases: firstly, a test environment was set up; secondly, mobile device was tested as a human user would use under ordinary-use circumstances and finally, suggestions were made on future testing implementations. To this end, a LG Nexus 4 was setup in an automated test environment that simulates a regular user's use conditions and executes a set of human user use cases, and gathers data on consumption of power as well as reaction and response times in the various interactions. The results showed that an operating system agnostic test environment can be constructed with a limited number of equipment that is capable of simulating a regular user's use cases as a user would interact with a mobile device to measure user experienced software aging.

Keywords and Phrases: software aging, smart device, smart phone, user interface, user experience, reaction time and response time

Acknowledgements

I would like to express my appreciation to my supervisor, Professor Roope Raisamo for his valuable time and contribution to this thesis. Thanks to Professor Erkki Mäkinen for his comments on the final report.

My gratitude goes to Tommi Toropainen for the opportunity to work on this topic and for his help to gather valuable data. Special thanks to Mika Katara for his efforts and time to guide me through the writing process and for the many encouragements. I am also honored to take this opportunity to acknowledge all those who have had an impact on this work in so many ways. Thank you for sharing your opinions, suggestions and for your moral support.

I would like to sincerely thank my wife, Henniina for her continued support throughout this work process.

Finally, all honor and thankful praises go to God for his innumerable favors and blessings in all of the efforts made toward this thesis.

Table of Contents

List of Figures

1. Introduction

The increasing use and popularity of mobile devices has brought to light many issues that are critical to a regular user's experience in using the device. It has been observed in mobile devices that as the runtime period of the system or process increases, its failure rate also increases. These failures are reported in a wide variety of ways, including incorrect service (e.g., erroneous outcomes), no service (e.g., halt and/or crash of the system), or partial failure (e.g., gradual increase in response time). These phenomena empirically observed in smart systems are defined as 'aging' in both hardware and software.

However, software aging is different from that of the hardware. Hardware undergoes wear and tear after certain stable use conditions in its life time, resulting in increased rate of failure, i.e., aging of the hardware [Tobias, 1995]. Software aging is a result of software bugs. Grottke et al. [2008] have shown that many aging-related failures of software systems are indeed the consequence of software faults, and Figure 1 explains their pathology for aging related failures.

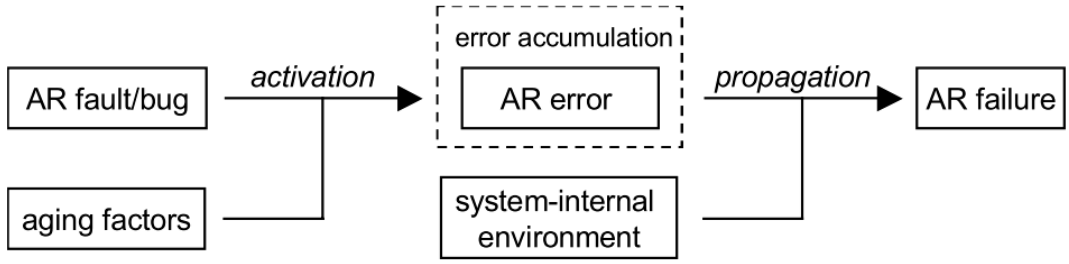

Figure 1: General "chain of threats" specific to these aging related failures

Understanding how a software system evolves will give further details about how software bugs are produced. A software system is a collection of various modules that work together to produce an end result [Sommerville, 2007]. Therefore, the production process of a software system, beginning with the design through its various phases of coding, testing and release, is also multilayered, which requires various tools and an

integrated development environment (IDE). For example, to develop an Android application the development environment can be set up with Android SDK and ADT plugin for Eclipse (assuming the IDE chosen is Eclipse), and the latest SDK tools with the chosen platform or platforms can be used to test the application [Android Developer, 2014].

The more complicated a software is, the more contributions it requires in various areas to provide the many features or services it is designed to provide. Software thus may be prone to late system deployment, may be delivered poorly, and may require system rework due to performance issues. On top of such complicated development process, the software development entities are constantly under pressure to deliver different versions of software and features faster than their competitors [Meeker and Wu, 2013]. All of these factors impact the development process, making it difficult to maintain the quality of code while keeping the costs manageable. Such issues in the end have a great impact on the performance and cost of software. However, high maintenance costs owing to poor performance, or increases in maintenance cost due to ad hoc performance fixes are equally inadmissible for software to function successfully.

Many software bugs are reproducible and can be easily found in order to be fixed during the testing and debugging phase. Those bugs that are hard to find or fix remain in the software during the operational phase and may never be fixed. They may not make themselves known as failures even long after the software is taken into use and many operations are retried, and even if the system is rebooted. Their manifestation is thus non-deterministic and dependent on the software reaching very rare states [Gray, 1985].These instructions do not cover research methods or matters pertaining to the content of the thesis.

Bugs and faults are dealt with through software system upgrades. Furthermore, software updates help software systems meet the constantly changing needs of the user [Miedes and Muñoz-Escoí, 2012]. However, the update can also have an effect on a software system, rendering it faultier or bug prone and thus a victim of software aging. One of the recent examples is Microsoft pulling the Windows 8.1 update offline even after releasing it for worldwide distribution [ZDNet, 2013] and then posting it back online after fixing the issues.

Therefore software updates are prone to developer's mistakes or other issues that fail to achieve the intended goal of preventing software aging notwithstanding the fact that

it is also very costly to update a huge quantity of Windows run machines around the world in terms of money and time spent. When considering an existing software system update, the entities involved may fail to consider the potential impacts it will bring to the system and finally to the user due to various constraints such as lack of resources, or fighting for an edge over the competitors by introducing new features and the like. All of these issues will make the changes take a longer time than expected or budgeted and yet introduce new bugs that will hamper performance and reliability.

As time passes, software also grows bigger in size in both the lines of code and in the disk space it requires on the system due to the changes required and added. Comparing the previous versions of Android to the latest, there has been a steady growth in size as new features have been added [Google Developers, 2014]. This growth in size can be explained by the fact that each new feature requires new code or a certain change in the code. This trend of increasing size with new changes and features will make troubleshooting and maintenance more difficult compared to previous versions of the software. In a study of certain commercial software product, Parnas [1990] mentioned that the list of known unrepaired bugs exceeded 2,000. Furthermore, as the size of the program grows, it places more demands on the computer memory and other resources resulting in slower and poorer performance overall.

Despite such complications with the end results of software, there has been increasing use of and dependency on software for automation since the first automated telephone switchboard was introduced. Significant investments have been made in the IT sectors in its various stages of developments and implementations. According to Gartner [2013], the yearly IT spending worldwide was expected to reach 3.7 trillion dollars in the year 2013 in the areas of devices, data center systems, enterprise software, IT services, telecom services and such. Furthermore, a research by Meeker et al. [2013] on key internet trends of 2013 indicates that there has been 8% year by year growth in 2012 to a total of 2.4 billion global internet users and it goes on to prediction that mobile device users will overtake fixed Internet access by 2014. This increase in dependency demands the best possible implementation of software and other related technologies. Therefore, the failure in estimating the deliverables of software technologies and proper implementations are catastrophic. Sessions [2009] claims that various IT failures including the failure to implementation appropriate software technologies costs the global economy a staggering 500 billion dollars per month.

Although considerable research has been devoted to software development rather less attention has been paid to the progressive performance degradation of software, i.e., software aging as experienced by a mobile device user. Software aging on a mobile device is a critical impediment to the users due to its pervasive nature. The question still remains as to how can a mobile device be tested for user experience that is not limited to only human interaction given the fact that human interaction is not easy to scale for a host of mobile devices in the future? How can then human interaction with mobile device be simulated, e.g., unlock the mobile phone to launch an application? How can the performance degradation be measured over time since the last rebooting/starting the device?

This thesis consists of seven chapters. Chapter 2 introduces the topic of software aging in depth. Chapter 3 lays the foundation for this study through various consideration in connection with previous researches in software aging. Chapter 4 records the methodology of the various experiments conducted. The results are presented in Chapter 5. Chapter 6 is a detailed view of the results against the expectations in the light of the previous studies with suggestions for future test implementations and improvements. Chapter 7 presents conclusion of this work.

2. Software Aging

In the interactive technologies, the term ‘user experience’ encompasses a wide range of areas that are associated with a user toward the technology in question. Forlizzi and Battarbee [2004] have shown that this association is generally over a wide variety of aspects that range from traditional usability to beauty, hedonic, affective or experiential aspects of technology use. There is always room for better understanding of what user experience really means in the various cases of different interactive technologies. Moreover, new interactive technologies introduce new areas of understanding and its impact need to be surveyed.

Notwithstanding the apparent misunderstanding the importance of studying user experience could not be emphasized more. Hassenzahl and Tractinsky [2006] claimed that, even though user experience is an unusual phenomenon it has been embraced by the human computer interaction (HCI) community, practitioners and researchers alike.

However, user experience has been critiqued for its lack of clarity and its in-applicability to define an experience of a user with regard to a device as a general experience for all the users of that device and in different cultures. In other words, the user experience of a device for a user is not equally applicable to all the users of the device in all the cultures. Hassenzahl and Tractinsky [2006] further explained that the user experience studies have gained momentum in recent years as an alternative to the to the dominant task and work related ‘usability’ paradigm and it is also in line with a much older principle which states that at the heart of the user experience is what the person experiences at that specific moment.

Furthermore, Hassenzahl and Tractinsky [2006] have expressed their view on user experience as an intersection of the three prominent perspectives where each of these perspectives opens up paths for the understanding of user interactions with technology. The three perspectives are: the understanding of beyond the instrumental purpose of the technology, the emotion and affect involved in interactions, and the immediate experience pertaining to the experiential aspect of the interaction (see Figure 2).

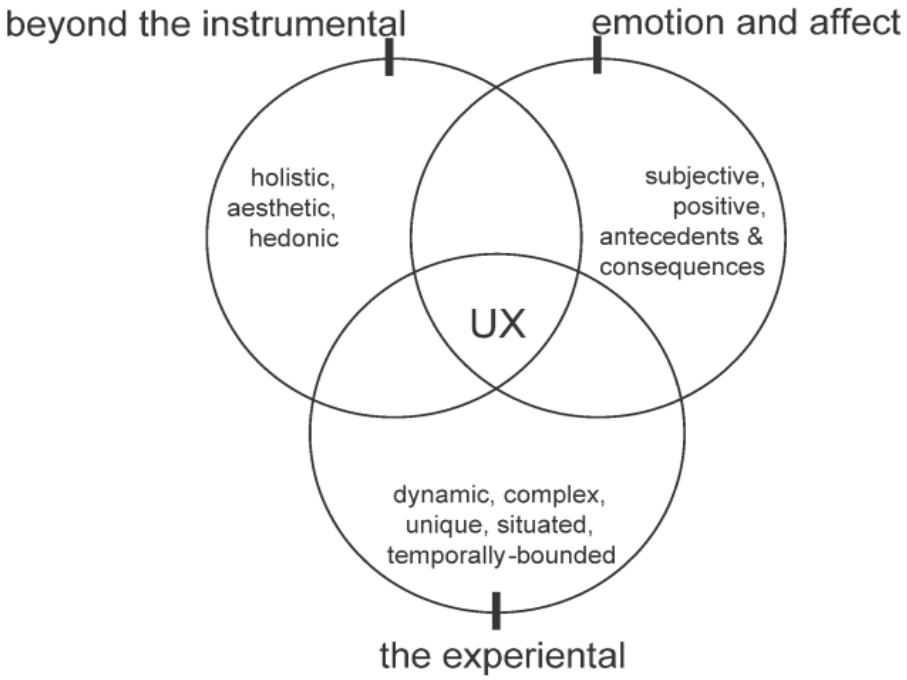

Figure 2: Facets of UX [Hassenzahl and Tractinsky, 2006]

However, none of these perspectives separately captures the complete user experience spectrum. The instrumental purpose of the technology only covers the areas that acknowledge the use as a subjective, situated, complex and dynamic encounter. This, however, underestimates the user's internal state of mind that encompasses the particular user's predispositions, expectations, needs, motivation, mood, and so on toward the technology interaction. Furthermore, the characteristics of the technology also come under organizational and social scrutiny in terms of how complex, purposeful or useful and functional in its environment within which the interactions with the technology may occur. These aspects of technology interactions impose a wide range of design and experience challenges which can be utilized to avail new opportunities and reach desired goals.

There has been strong interest in user experience validation in recent times around various interactive technologies. Notwithstanding the complexity of the user interaction with technologies, many interactive products find their way into our daily lives because of their advantages over the disadvantages resulting in growing and changing base of users shifting the parameters of demand for interactive products.

Using a meta-analysis of 45 end-user satisfaction studies published between 1986 and 1998 Mahmood et al. [2000] have shown that the level of end-user satisfaction in interactive technology has widely been accepted as an indicator of its success. The study focused on relationships between end-user satisfaction and a set of nine variables: perceived usefulness, ease of use, user expectations, user experience, user skills, user involvement in system development, organizational support, perceived attitude of top management, and user attitude toward the interactive systems in widely divergent settings. Figure 3 depicts the relationship among the end-user satisfaction and the nine variables. Their most significant finding concerned the relationships of user involvement in systems development, perceived usefulness, user experience, organizational support and user attitude toward the interactive technology.

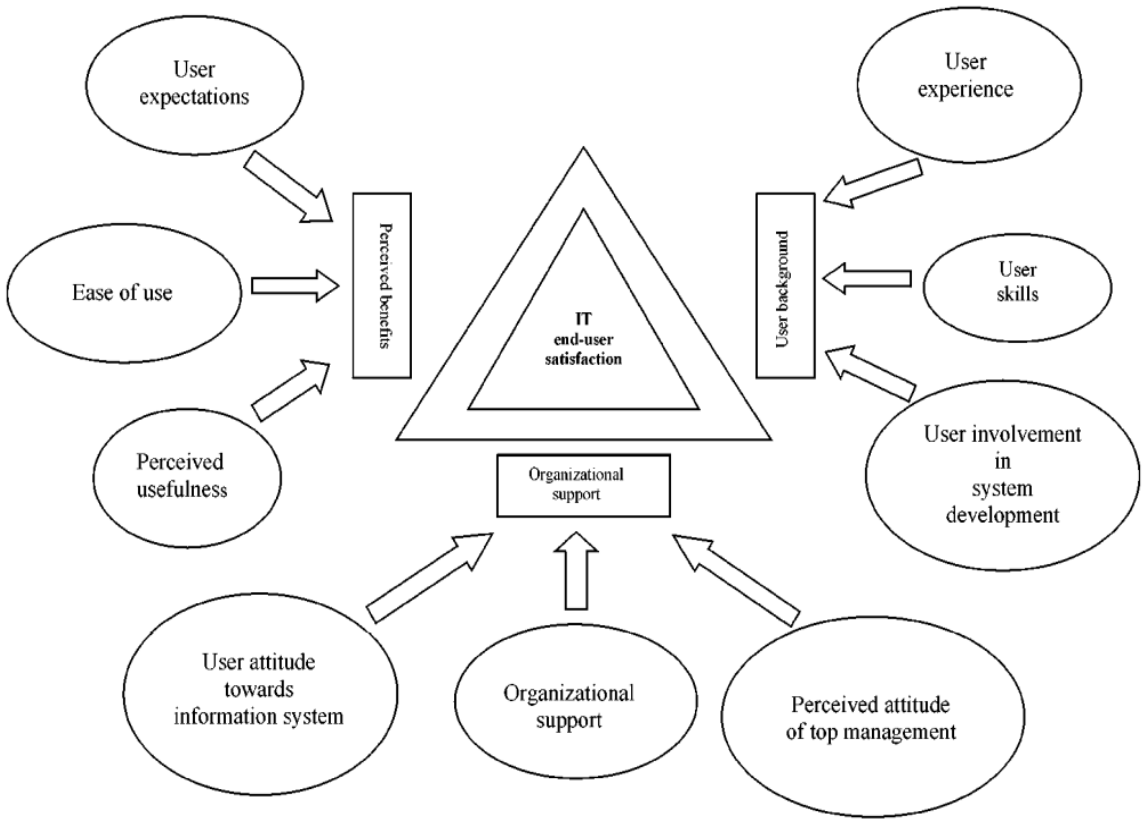

Figure 3: Research model of factors affecting end-user satisfaction

The empirical analysis presented by Cotroneo et al. [2010] shows that software aging effects are related to the static features of the software. The research was

conducted by adopting a set of common software metrics concerning program structure, e.g., size and cyclomatic complexity, along with some features specifically developed for the study and the matrices were computed from ten complex software applications affected by aging. A similar approach was followed in the course of the current study of user experienced software aging in a mobile device by observing user interaction with the interactive technology over time through the perceived experience of the user.

2.1. Software Aging Test Methods: Understanding and Scope

Software aging test method in this study strives to understand the usability of the mobile device over time as the aging can be noticed and measured through the use of the various features build into the device [Cotroneo et al. 2010]. The Guidance of Usability by the International Standards Organization in the ISO 9241-11 (1998) defined usability as, "[t]he extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use." Nielsen [2012] defined usability by five quality components:

- Learnability: How easy is it for a user to complete a basic task at their first use of a system?
- Efficiency: How quickly can a user familiar with the system perform tasks?
- Memorability: How easy is it for a returned user to reestablish proficiency regarding the system?
- Errors: How many errors does a user make using the system? How severe are the mistakes, and how difficult or easy is it to recover from the mistakes?
- Satisfaction: How satisfactory is it to use the product?

Usability defined in the five components above encompasses a wider range of aspects covering the ease of use, the effectiveness of the product, the subjective user preferences that bring user satisfaction and such. However, an individual satisfaction toward a product does not guarantee the best usability or even the applicability of the product for all [Mahmood et al., 2000]. On the other hand, a product that is laden with usability problems may very well be difficult to operate and thus defeating the purpose

of the mere existence of the product, e.g., a user interface with usability issues may lower user productivity and discourage further user engagement. Usability can be improved by understanding the actual need of the user in the right context where the product will ultimately be used following a set of usability evaluation methods categorized by Nielsen [1995]. They are as follows:

- Automatically – in which usability measures are computed by running a user interface specification through special software.
- Empirically – in which usability is assessed by testing the interface with real users or experts.
- Formally – in which usability measures are calculated by exact models and formulas.
- Informally – in which usability measures are obtained based on rules of thumb and the general skill and experience of the evaluators.

However, despite the wide dissemination and growing popularity of various aspects of user experience it still remains to be clearly defined and well understood as a whole. Law et al. [2009] suggested that the interests of wide range of parties including academia and industry in the issues of user experience led to the fact that the researchers and practitioners in the field have become more aware of the limitations of the traditional usability framework that fundamentally focuses on user cognition and user performance in the various aspects of human-technology interactions. Furthermore, their research indicates that user experience highlights the non-utilitarian aspects of such interactions shedding light on user affect, sensation, and meaning and value in everyday life.

This thesis presents a study on software performance of a mobile device as experienced by a user. Thus, for the sake of this study, the user experience understanding and scope was defined in area of user interactions with the mobile device.

This empirical method was chosen as the best fit in this study as it fulfills the requirement of the regular interaction by end user to test the case of software aging. Furthermore, the increased popularity in the empirical method also strengthens the choice of the evaluation method.

One of the most natural interactions between a user and the mobile device is to launch various applications that are already included in a mobile device right out of the packaging when it is sold. As an example, a user will launch the browser application on a mobile device to browse a website. Furthermore, the process of launching an application on a device without physical buttons poses a set of its own challenges. The apparently simple process of launching an application can be divided into following set of steps:

1. User taps the icon to launch the application
2. A haptic feedback is immediately issued to notify the user that the tap is registered.
3. Audio and visual feedbacks are further provided to the user.
4. The user releases the screen area of tapping by lifting his or her finger.
5. Soon afterward the first change in the screen is observed. The time taken to show the first change, reaction time can be calculated at this stage.
6. Finally, it reaches a state when the application is completely launched by going through the states from the standby screen where the user tapped the icon, e.g., Chrome browser icon, to the state where the application is completely launched (see Chapter 3 for details). Response time, the time taken to complete all the stages of launch until the application is available for use can be measured.

Therefore, by evaluating the steps mentioned above in a user interaction between the user and device, the software aging can be measured. This study thus aims to answer how these different phases can be measured. How to understand from a user's perspective the slowness in his or her device performance? How the reaction time, the time to see the first change after the user taps an icon to launch an application, changes in course of time starting for the last reboot? How does the response time, the time it takes for the launch to complete from the tap of the user to launch that particular application, change over time? In addition to that, how can the battery consumption be observed as to how the power is consumed? Does the current consumption increase or decrease over time? Or does it stay at the same level? How does the performance

change after reboot? Can a certain testing environment be used for all mobile devices irrespective of its operating system?

2.2. Software Aging Tests Administration

The very nature of this study required to have user interactions that can be repeated for several times, e.g., launching an application from the home screen of a mobile phone. In addition to the repetitive nature of the interactions, these interactions needed to be carried out over a relatively large period of time at given intervals with the requirement that a set of similar steps will be followed. Thus, in the light of the aforementioned test conditions, to ensure the best repeatability test automation was taken into the central focus of the tests administration. Test automation allowed for executing the tests using a robotic test environment that simulated human end user interactions with flexibility to design the tests with 100% repetition accuracy. Some superficial factors were also considered, e.g., computers in the current times are less expensive to deploy compared to a human work hour cost. Furthermore, a computer knows no tiredness or boredom that may be a hindrance to a human tester for the repeated nature of this study.

However, the fact that the computer does not have a mind of its own as opposed to the human tester, extra care must be taken to deal with all the possible issues. These issues are needed to be fed into the computer with the best possible instructions. Thus, the process of articulating possible error scenarios as well as the solution along with the regular set of instructions require significant amount of time and other resources [Fewster and Graham, 1999].

Fewster and Graham [1999] described the goodness of a test case by four attributes: the quality of the test, the extent of the exemplary nature of the test, the cost incurred in the processes of the test and how much effort is needed in maintenance of the test. However, as may be evident from the diagram, Figure 4, these four attributes are at odds against each other. A test that is capable of testing more than one case is likely to be more costly. It may also require more time and other resources to analyze and report. The complexity of the test may require rather extensive maintenance upon changes in the software or the settings.

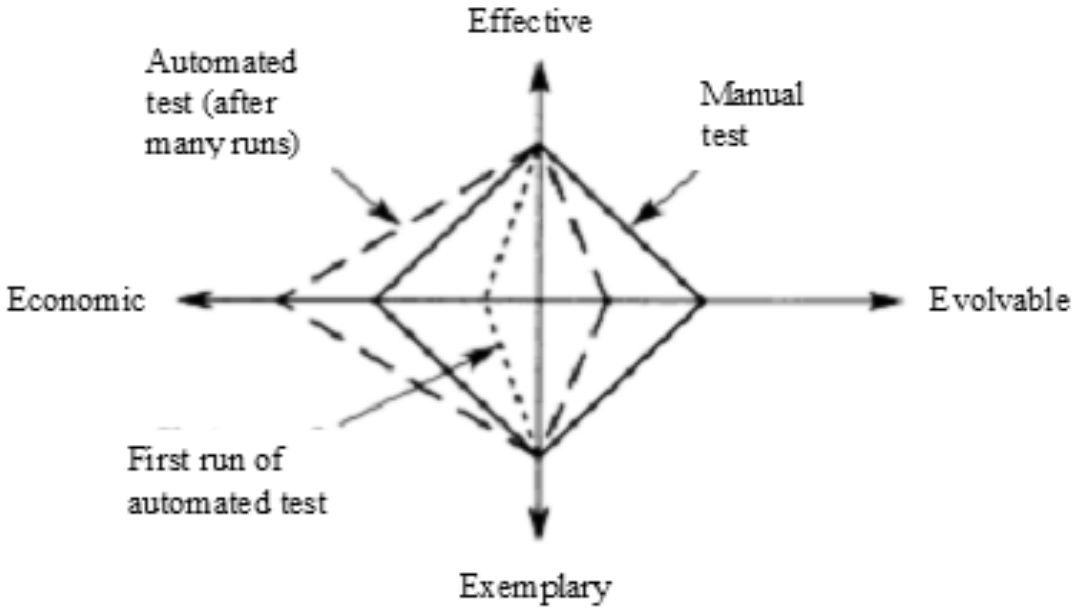

Figure 4: The goodness factor by the four attributes of test case in test automation

In order to reap the benefits of an automated test, the test has to be cautiously chosen and implemented. However, Fewster and Graham's [1999] work claims that the automated quality, i.e., testing by automation is independent of test quality, as the effectiveness and the extent of how exemplary a test can be is not affected by the way the test is administered, e.g., performed manually or automated. Automated test is generally more economic as the cost of execution is low compared to the manual execution by a human subject.

The tests executed in this study were clearly defined. Thus, the definitive nature of the tests allowed for a detailed set of instructions including error handling scenarios.

2.3. Software Aging Beyond Stock Android

This study was set to be made on a Nexus 4 device running the original version of Android or the stock Android available for Nexus 4 devices [Google Developers, 2014]. However, the scope of Android platform use in mobile devices and its various implementations go beyond the current scope of testing.

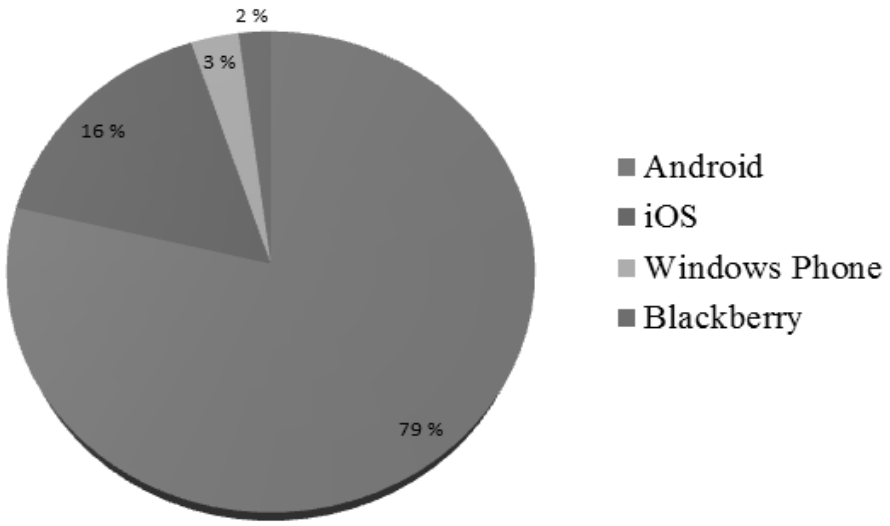

Figure 5: 2013 Full year operating system market shares

In 2013, Android mobile operating system took the top place for holding the most operating system market shares (a total of 1,492 million units were sold), iOS and Windows Phone come as second and third respectively [Ahonen, 2014]. Figure 5 puts the distribution in perspective where Android dominates with 79% of the market.

This large distribution of Android and owing to its very nature of being an open-source platform, various Original Equipment Manufacturers (OEM) have customized the Android operating system to run on their products. In 2013, Samsung led the competition in the Android market with over 65% share in 2013 [Jones, 2013] with customized version of Android running on their devices. It introduced its own platform security tool, applications and other user interface changes. Similarly some of the other top Android manufacturers such as HTC, Motorola, LG and Sony also modified the basic version of Android and added their own tools and applications. TouchWiz UI in Samsung, HTC Sense UI in HTC are some of the differentiations these vendors have introduced though out their mobile device line up.

These changes may impact how software aging is perceived by an end user as these changes impact how processor, memory and battery is used. Besides these changes and their impacts on a mobile phone performance, any mobile phone user can go to an application store, e.g., Google Play (http://play.google.com), Amazon Appstore for

Android (http://www.amazon.com/mobile-apps/b?node=2350149011) and such. These application stores offer applications developed by third parties who may choose to strictly adhere to the guidelines set by Google for application development or not. As Android applications operate within a shared resource environment, the performance of an application will be impacted by how efficiently it interacts with those resources in the broader system scope. Some applications also operate in a multithreaded environment and compete with other threaded processes for resources. Improper handling of resources can cause performance problems and will make it difficult to diagnose a particular failure or issue further. Android Developers site (http://developer.android.com/tools/ debugging/ index.html) offers ways to measure the performance of applications by third parties. The Systrace tool provides a very useful way to collect and review code execution data for an application in the Android operating system.

Similarly, Windows Phone Application Analysis tool makes way for the application developer to test the application. The process of analysis is divided into two parts, App Monitoring and Profiling. These require the application to conform to standards of the following items: application launch time, application responsiveness and maximum memory usage by the application [Dev Center – Windows, 2014]. iOS also offers its share of ways to tackle software performance issue, i.e., software aging [iOS Developer Library, 2013].

Therefore, this study also aimed at incorporating various devices that can be tested irrespective of their sizes and the operating systems. The next chapter gives an over view of some of the related areas of research. The methodology of the study is explained in Chapter 4. The discussion of the result is found in Chapter 5 and the summary is presented in Chapter 6.

3. Focus on Software Aging in the Past

The phenomenon of aging affects both hardware and software alike. There has been, thus, an increased focus on the process of software aging after the idea was popularized in the latter years of the 1990s [Huang et al., 1995].

Starting with hardware, Nightingale et al. [2011] have done extensive research on a set of a million consumer PCs to understand the effect of hardware introduced operating system failures on consumer machines. Using post-hoc analysis of machine status reports and OS crash logs they showed that the rate of failure is non-trivial, with hardware crashes at a rate of 1 in 190 over an eight month observation period. However, the recurrence of a failure is common; for instance, the first hardware crash increases the possibility of further crashes by up to two orders of magnitude with the rate of recurrence being as frequent. Their research also concludes that the CPU speed matters, but overclocking significantly degrades the reliability whereas underclocking significantly improves the same.

The research conducted by Gray [1990] over the period of 1985 through 1990 shows that one of the significant causes of software failure has gone from hardware and its maintenance issues to the failures in the software the system uses. In practice, the change has been significant as the following numbers suggest: the need for hardware and its maintenance fell from 50% to 10%, but, on the other hand, the maintenance on software increased from 33% to 60%. Furthermore, the trend is likely to continue as there has been a dramatic rise in the lines of code implemented in software and other factors such as dependency on vendor products, customized applications and limited time to test the actual software before deployment. Thus, a highly successful system can only exist with high quality software that has few defects found in the software and it needs to be able to exploit highly effective recovery mechanisms to understand and tackle any software errors which may appear in the course of use and can be maintained in non-disruptive fashion, e.g., new updates that can be delivered without interrupting the user's activity.

Pienaar and Hundt [2013] have shown that even though JavaScript and its semantics have been standardized over the course of time, its incompatible use by various JavaScript engines and their embedding browsers have fatal impacts on mobile devices by introducing memory leaks. Despite the fact that JavaScript is garbage collected, the

intended outcome may not be reached in many cases due to incompatible use. For example, if references made to otherwise dead objects exist, during execution the system will consider those objects reachable and the memory thus allocated cannot be reclaimed. This inadvertently gives rise to memory leak. As a consequence, such conditions led to some massively sized Gmail processes which made it difficult for mobile devices, such as Chrome books or tablets, typically with less physical memory (compared to a desktop computers) to function normally. In such situations the application had to be terminated on a regular basis, interfering with the use by a regular user.

The competitiveness in the software development industry imposes tremendous pressure to improve the end result of the software right from the beginning of the development process. But to stay competitive the organizations need to pay more attention to delivering the finished product at a lower cost compared to the competitors. In such market driven software development processes one of the ways to gain certain edge over the competitors is to analyze the test process that will help avert rework. As the rework accounts for more than 50% of the actual development time [Duka and Hribar, 2010]. The most common reason for such high costs in rework is caused by fixing the same bug in the various phases of the software development and maintenance processes. A fault is much cheaper to fix in an early stage compared to the later stages.

The situation where a certain fault is carried over from one phase to the next is called fault slippage. Duka and Hribar [2010] suggested a measure called faults-slip-through to determine the faults that are more cost-effective to find in an earlier stage of the development. Despite the fact that there are numerous ways to make software development more effective and efficient, the fact—the software development processes spend their 50% of the total time in rework—still remains.

However, software testing is also a significant part of the development process which can be better utilized to find more faults early when the faults are cheaper to identify and fix. Damm and Lundber [2006] demonstrated improvements in decreasing fault rates and Return on Investment (ROI) using an early fault detection concept.

Software aging is a constant challenge for the ultimate user experience. It has been observed in mobile devices that as the runtime period of the system or a process increases, its failure rate also increases. Users have reported such failures in a wide variety of ways. Some of the most common issues include incorrect service (e.g.,

erroneous outcomes), no service (e.g., halt and/or crash of the system), or partial failure (e.g., gradual increase in response time) [Huang et al., 1995]. Furthermore, these faults result in data inconsistency in the mobile device. The inconsistency in data can be further explored in investigating the memory bloating and leaking, failure to release file-locks that lead to data corruption, storage space fragmentation and accumulation of round-off errors. These issues ultimately cause the mobile device to perform poorly and as a result the device user experiences a slow degradation of the services.

Even though the existence of software aging has been widely reported there is still room for more study, especially from the end user's perspective as to how the user sees the device to age.

4. Methodology

The purpose of this thesis was to study how software aging affects mobile device performance, in other words, how user experienced software aging can be measured on mobile phone. This chapter explains the research methodology of this study, the sample selection, the procedure used in designing the tests and data collection.

This study tries to understand the question of how a mobile device ages over time since the last boot through setting up a device agnostic testing environment, simulating a common mobile phone user use conditions and executing a set of use cases, e.g., powering on the device by pressing the power button, unlocking the device for use, launching and closing applications and measuring the various phases of an application launch. Nexus 4 was chosen as the test device running Android operating system 4.3. The device was placed on a table to be made available to a robot with 3 degrees of freedom namely x, y and z axes to simulate the behavior of a human interacting with the device, e.g., tapping, swiping and such in its given scope. The testing environment was further equipped with high speed camera to capture the changes on the screen following an action of tapping an icon to launch the application. The battery consumption was also measured by a current consumption analyzer on each of these interactions. A set of applications were used to simulate a regular user's use conditions and interactions with a mobile phone. Each application was launched from the home screen of the mobile device.

Launching an application on a mobile device with no physical buttons is very different from that of a mobile device with buttons—the lack of the physical button-press feedback is overcome by providing multi modal feedback. The virtual button on the touch screen is touched with a stylus or a finger. Following the touch a certain time is required to get the feedback which is known as touch feedback latency. The release of the stylus or the finger happens sooner or later as it depends on the user and the release feedback takes an amount of time which is called touch release latency. Visual feedback (change in color of the item interacted with), audio feedback (audible click sound) and tactile feedback (short vibration that can be felt) are also provided in the process. Kaaresoja and Brewster [2010] showed that the process and phases of a virtual button press can be illustrated as in Figure 6.

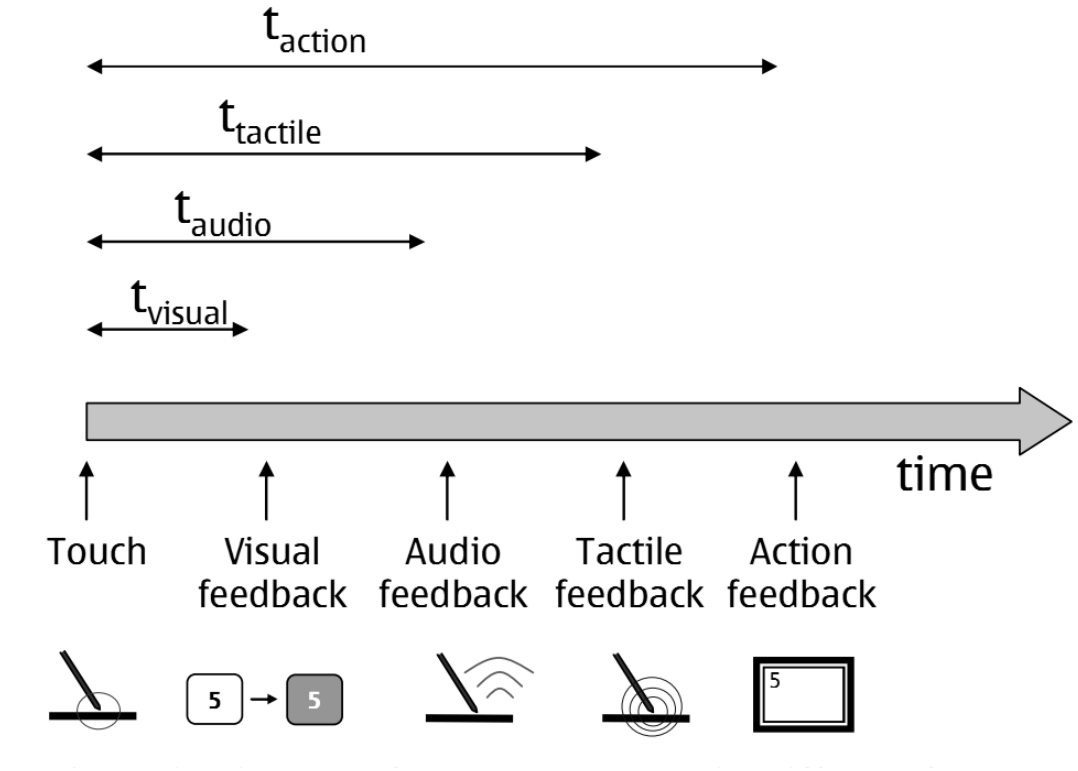

Figure 6: A touch event and its different feedback types

In an effort to measure the changes in each application launch during the test a similar approach was followed to categorize the different phases of the launch. Each launch was considered to be divided into seven different smaller events:

1. User taps the icon of the application to launch
2. Haptic feedback
3. Audio feedback (if audio is enabled)
4. Visual feedback
5. User releases after the tap
6. Reaction time
7. Response time

Figure 7 explains how the whole process of launching an application can be divided into seven different events over time by plotting the events against time in milliseconds along the X-axis.

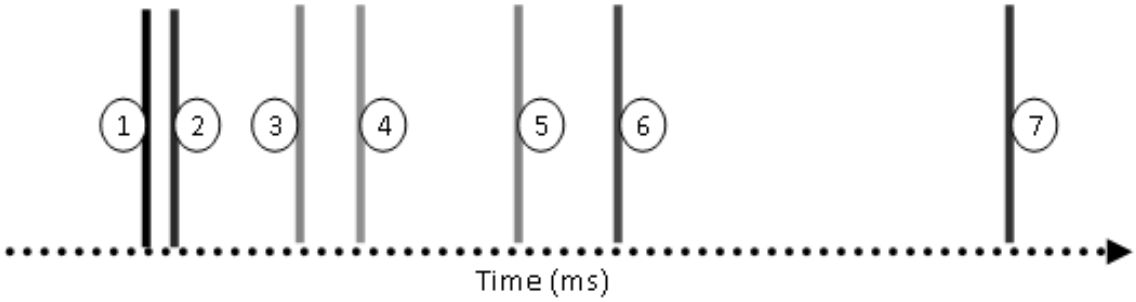

Figure 7: Typical application launch events presented against time in milliseconds

However, due to the unavailability of the equipment to measure the haptic feedback and audio feedback during the test execution, they were ignored from the test considerations.

Figures 8 and 9 depict the various stages of an application launch demonstrating the events described above, e.g., Chrome launch from a user's perspective. In this case, it is expected that the user will launch the application from the device home screen after the device is unlocked on standby or by going to the home screen upon pressing home button.

Mobile device in home screen waiting for user interaction

User taps Chrome icon. Following the tap a series of events occur: user first feels the haptic feedback, simultaneously an audio feedback is also given if the device is set up to provide audio feedback and the user also sees a visual feedback.

Figure 8: Various stages of an application launch from a user's perspective

User releases after the tap action. Application launch process continues: first the reaction time is calculated. In the picture above the transition is well underway as the chrome app can already be partially visible. Human eye is incapable of observing very minute change in the home screen until a big change is visible like it is shown in the picture above.

Finally, when the launch process is completed, the response time is calculated.

Figure 9: Various stages of an application launch from a user's perspective

4.1. Test Automation and Test Equipment

In this study the human interaction with the device was simulated through a controlled environment in a laboratory with a robot programmed to launch a set of applications exactly like a human would do on a day to day basis. The primary reason for automating the testing process was to make it possible to repeat a set of procedures, theoretically, infinite number of times without any changes whatsoever.

The device set up in a testing laboratory further enabled exact control over the temperature of the environment, level of lighting and provided consistent power supply. The controlled set up in the laboratory also made it possible to use high speed camera to capture the changes on the screen of the device without any interference or imposing any restrictions on regular movements to interact with the device. The set up was further equipped to measure the consumption of battery through every test execution.

This automation was designed and executed to simulate the regular human user interaction with the device without compromising for human error and was equipped to measure with high speed camera and power consumption. The nature of such set up also allows for any device form factor to be tested irrespective of its operating system or size, i.e., a device or operating system agnostic testing environment.

A set of 13 different applications were chosen to be launched on a regular basis. The applications we divided into two broad categories: regular use applications and a benchmarking application.

The regular use applications included: Calculator, Camera, Clock, Gallery, Gmail, Hangouts, Maps, Messaging, Phone, Play Music, People and YouTube. 3DMark for Android from Futuremark was chosen as the benchmarking application due to the vendor's cross platform scope and popularity. One of the three tests, Ice Storm Extreme was chosen as a control to measure via the score at the end of every test executions. The Nexus 4 was chosen as the test mobile device running the stock version of Android 4.3 with all the applications preinstalled except for 3DMark benchmarking application. 3DMark was installed from the Android application store, Google play.

The process of automation was handled from three major perspectives to work on the device under test. They are test design and development, test execution system setup and the robotic environment that simulated an end user interaction with the device. Python programming language was used in conjunction with various other libraries that enabled features of image recognition, robotic API command execution instructions to administer robot actions in various tests of launching the above mentioned 13 applications as the combination was the best suit to make use of the freely available libraries required to get the most benefits of the robotics and data gathering process. In the test development process the data to be gathered was also outlined and documented. The test script to execute the test was written keeping in mind that there can be changes in the test data. Thus it was kept as modular as possible to minimize the maintenance requirements. The test execution scheduler in the setup allowed for all of the test or some parts of it to be executed in any number of iterations at any possible intervals without aborting an ongoing test process.

Figure 10 represents a detailed view of the test automation design that worked as a framework for the current study and also in the future because of the operating system agnostic nature of the combination.

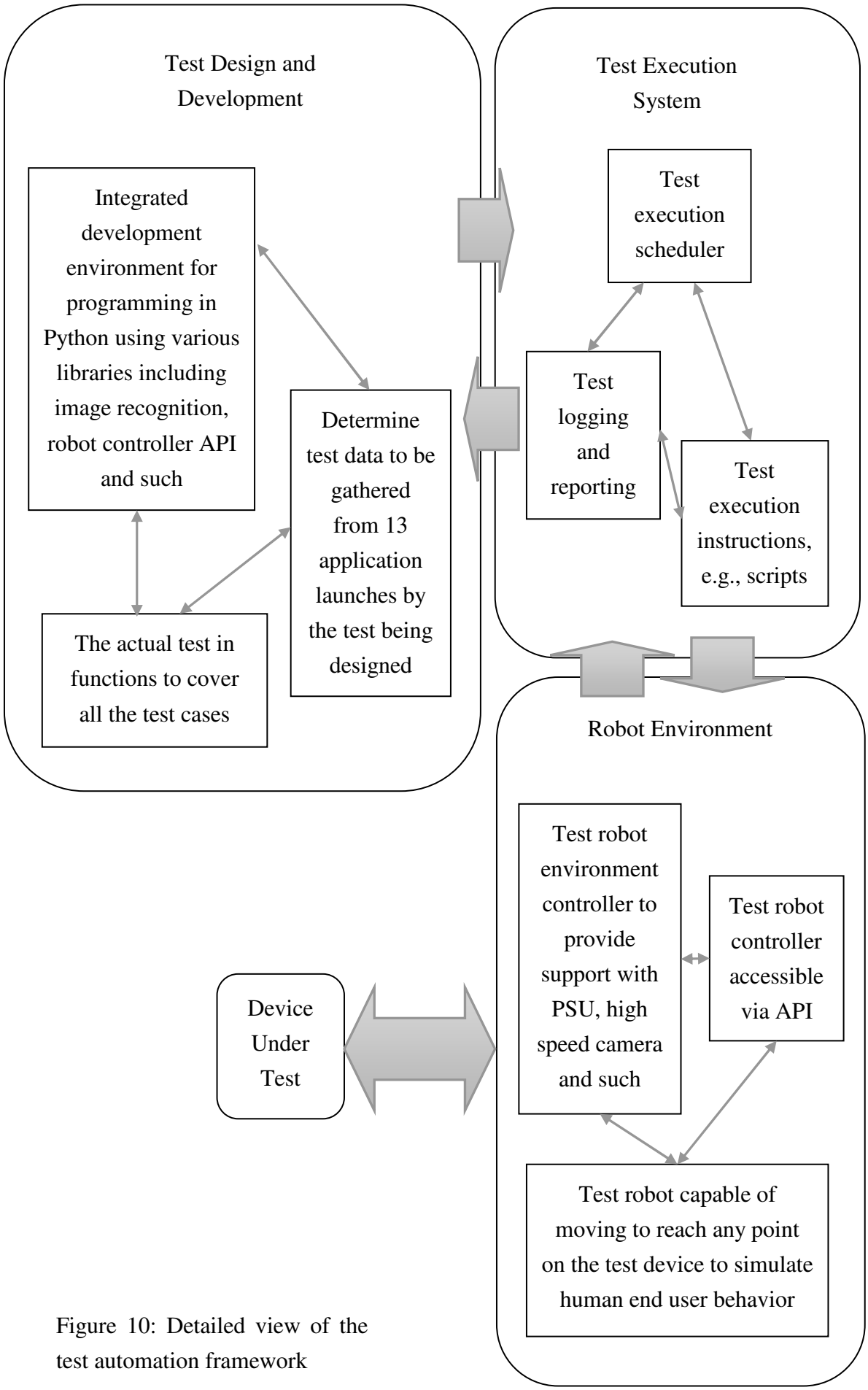

Figure 10: Detailed view of the test automation framework

The interactions shown in Figure 10 between the test execution system and the device under test can be defined by the requirement of the test procedure as intrusive or non-intrusive. The setup was also integrated to have reporting and logging capabilities allowing for the test execution to go without human supervision. The robotic environment provided access to the robot to simulate the human end user behavior through non-intrusive interactions with the device.

The automated test execution method made way for numerous number of repetitions of tests, in the precisely identical order with its individual steps every time, which is not possible to do in situations with human participants over an extended period of time. The applications were launched in a series starting with Calculator and ending with 3DMark. Using a job scheduler each series of 13 applications were executed every 4 to 5 hours break. Each application in the series was launched simulating a human user interaction by robot, e.g., finding the icon of the application to launch from the home screen and then tapping the icon to launch it. The closing of the application was simulated by following the steps a human user would use on a typical mobile phone.

Machine vision libraries enabled the robot to learn the right icon or image for the right application though its different stages starting with the icon of the application on the home screen through images of the different stages of the application. Thus, robot simulated the human user application launch by first going to the home screen, recognizing the right icon for the application to launch and then by tapping the icon. It then continued follow the process of launch by comparing the images available on the device screen though out the launch until the main view of the application was found. During each launch of an application a separate high speed camera was used to measure the changes on the device screen to determine the reaction time and response time of the launch. In addition to that the launch time, i.e., the time to recognize the main view of the application via robots own machine vision was also collected to complement the high speed camera data. The test environment was further equipped with power measurement devices to measure how much power in terms of watts were used during each launch of the application.

The data collected in the study consists of reaction time and response time (collected using high speed camera), perceived launch time calculated by machine vision (without the high speed camera) and power consumption in joules (Watts X Seconds). The data collected was expected to show how the device ages after the last

booting, e.g., the changes in data of how reaction time, response time, perceived launch time change as well as the changes in power consumption. If the time to launch an application increases over time then device will appear to be slower to the user, and if the power consumption increases over time after the last booting the device will appear to lose battery quicker which is also a common sign of poor performance.

4.2. Focus Areas Investigated in the Tests

Throughout the testing process in an effort to measure user experience software degradation following four criteria were considered. These four areas define how usability is perceived by an end user [Merholz, 2007]:

- Reaction Time: It is the time taken to show the first change as evident by the high speed camera after the robot taps an icon to launch the application. It is expected to change in course of time starting from the last reboot and throughout subsequent use.

- Response Time: The duration of the launch of an application, the time it takes to go through the various stages starting with the act of tapping the icon of the application being launched followed by visual feedback, audio feedback, tactile feedback, action feedback and finally the state where there is no more change detected on the screen by the high speed camera.

- Launch Time: This metric is used to complement the total launch time of the application in addition to the response time. It is calculated from the time a particular icon was tapped through the complete launch of the application as determined by machine vision capability of the robot via identifying a particular item on the screen of the mobile phone.

- Power Consumption: The cost of current (use of battery) at every launch was calculated as joules (obtained as the product of launch duration time in seconds with watts of electricity used as measured by the robot environment).

- Data Comparison and Scope: The test results varied depending on the application being launched in terms of reaction time, response time, launch time and power consumption. The data thus collected were compared to that of after rebooting the device. Furthermore, for every span of the device use, following a reboot to the next reboot each measurement was compared against the median measurement of reaction time, response time, launch time and power consumption for each application.

5. Data Analysis and Results

The analysis of the test results is presented in this chapter. The tests were carried out to understand from the end user's perspective as to how the user sees mobile device to age after it was rebooted or restarted the last time and since then as the time has progressed. The changes were thought to be as slowness in the device performance, change in the reaction time and response time, the change in battery consumption and how the performance changes following a reboot. The tests were carried out using a robot which simulated a regular user environment and activities on two different mobile phones from different makers and with different specifications. A set of thirteen different applications were launched and closed simulating a normal user behavior. In each iteration, a set of three measurements were made for reaction time, response time, launch time calculated using machine vision and power consumption.

5.1. Data Considerations

A total of 150 sets of application launches were performed. Each set consists of three launches, thus a total of 450 launches were tested. Due to various hardware related issues reaction time and response time for some application launches were not possible to be reported using high speed camera. In an effort to compliment the data collected on application launches through reaction time and response time using the high speed camera, the launch time was also calculated by machine vision camera. Thus, the machine vision camera was also used to calculate the time from tapping an icon to launch the application to the state when the application launch was completed as determined by matching against a particular section of the fully launched application screen. Each application launch was further subjected to power consumption measurements

Figure 11 shows a case of launching the Music application at different times of the device before a reboot and then following a reboot.

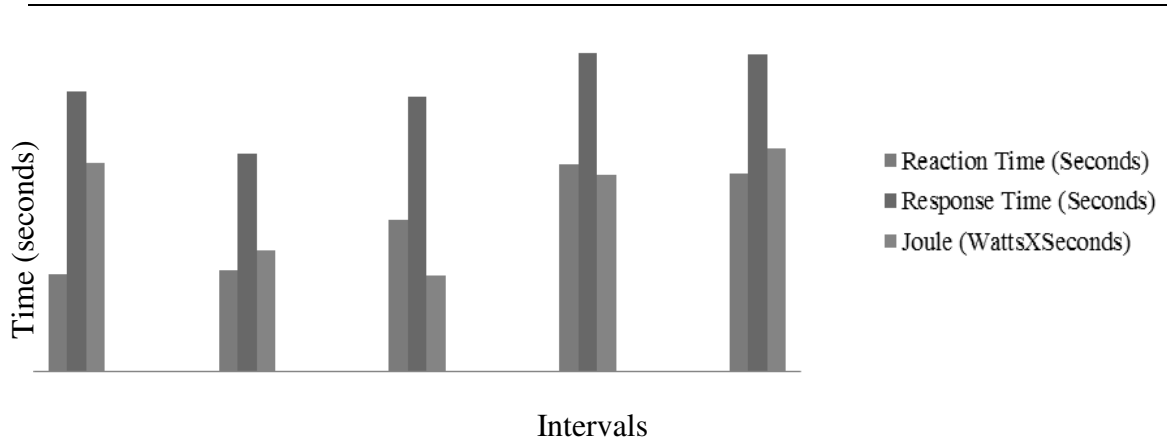

Figure 11: Music application launch at various intervals of the device use

In Figure 11 the points 1 through 5 represent the intervals at which the measurements were made as shown in Table 1:

Interval #	Device up time	Device idle time
1	157 hr 43 min 24 sec	590 hr 31 min 46 sec
2	49 hr 24 min 38 sec	196 hr 58 min 18 sec
3	65 hr 59 min 48 sec	260 hr 28 min 9 sec
4	150 hr 18 min 42 sec	558 hr 51 min 55 sec
5	172 hr 58 min 23 sec	640 hr 41 min 3 sec

Table 1: The intervals at which music application launches were executed

The device up time is calculated as the total time used in active use with direct user interaction and the idle time is the total time when not being used by a user in direct interaction but there are background services that are being run automatically without user intervention.

The first point in the Table 1 shows the amount of reaction time, response time and power consumption before a reboot was made, and at that point the device was used for about 158 hours while it was in standby or idle state for about 590 hours. The points 2 through 5 show various states of the device following a reboot. There is a rise in reaction time, response time and power consumption which can be interpreted by the user as the device being slow just before the reboot at the first point and then following

the reboot the device performs faster until the device reaches the point 5 on the graph when almost similar condition of the first point is repeated.

To further analyze the software aging process a bench marking application, 3D Mark was used. The Android version of the application contains three Ice Storm tests, out of which Ice Storm Extreme was chosen for comparisons using 1080p graphics test. Ice Storm Extreme test measure the device through raising the rendering resolution from 720p to 1080p, using higher quality textures and post-processing effects in graphics tests which creates a demanding load for the test device.

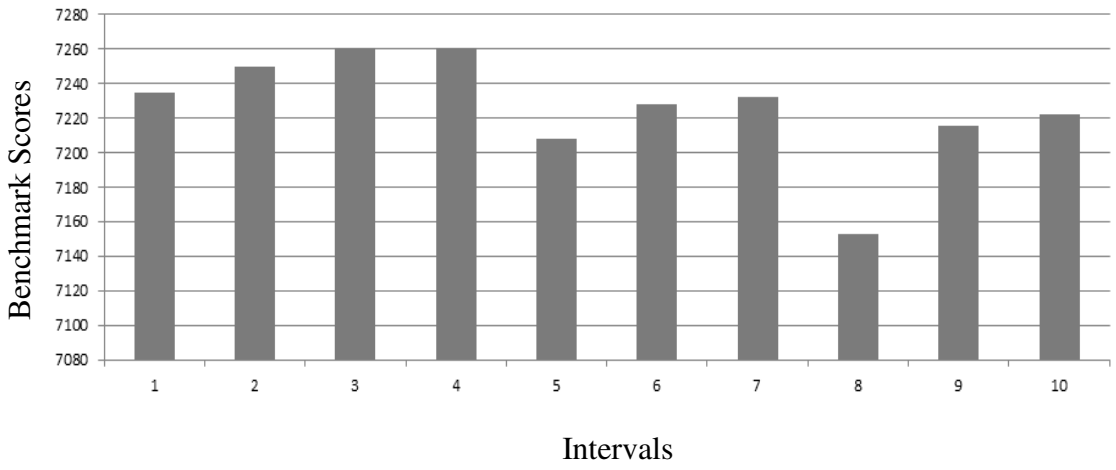

Figure 12: 3DMark-Ice Storm Extreme scores at various stages of the device use

Figure 12 represents a set of 3DMark Ice Storm Extreme scores at various states given in Table 2. The scores from 1 through 5 are measurements made before restarting the device while it has been in use since the last boot. The scores from 6 through 10 are measured following a reboot. While the average Ice Storm Extreme score in this test conforms to the average (a score of 7164) reported in the Futuremark website, however, it did not seem to show any effect of aging.

Interval #	Device up time	Device idle time
1	119 hr 6 min 47.71 sec	447 hr 46 min 58 sec
2	132 hr 56 min 39.6 sec	499 hr 11 min 11 sec
3	157 hr 43 min 23.84 sec	590 hr 31 min 46.12 sec
4	180 hr 1 min 11.29 sec	672 hr 21 min 33.13 sec
5	201 hr 10 min 43.46 sec	750 hr 28 min 16.97 sec
6	49 hr 24 min 37.5 sec	196 hr 58 min 18.17 sec
7	80 hr 29 min 23.57 sec	314 hr 18 min 2.68 sec
8	144 hr 39 min 2.08 sec	538 hr 2 min 6.3 sec
9	166 hr 27 min 58.16 sec	616 hr 59 min 17.58 sec
10	181 hr 0 min 54.69 sec	669 hr 30 min 53.62 sec

Table 2: The intervals at which 3DMark Ice Storm Extreme scores were measured

5.2. Reaction Time Analysis

Music being one of frequently used applications [McCarney, 2007] is a case in point for discussion. In the course of the study, the music application loaded with 100 songs was launched more than 135 time in phases of three for each time. In each phase a median was taken out of the three launches. For example, if in one phase of the runs the reaction times of the three launches were 1985 ms, 2050 ms and 1407 ms, the median was taken as the representative values as 1985 ms.

In Figure 13, 47 different reaction times are plotted which were recorded before a reboot was performed. Each value presented is a ratio of the median value of all the values before reboot. The measurements were started when the device had its uptime at 428808 seconds and idle time at 1612018 seconds and the testing continued through until the device reached 177878 seconds of uptime and 709098.17 seconds of idle time.

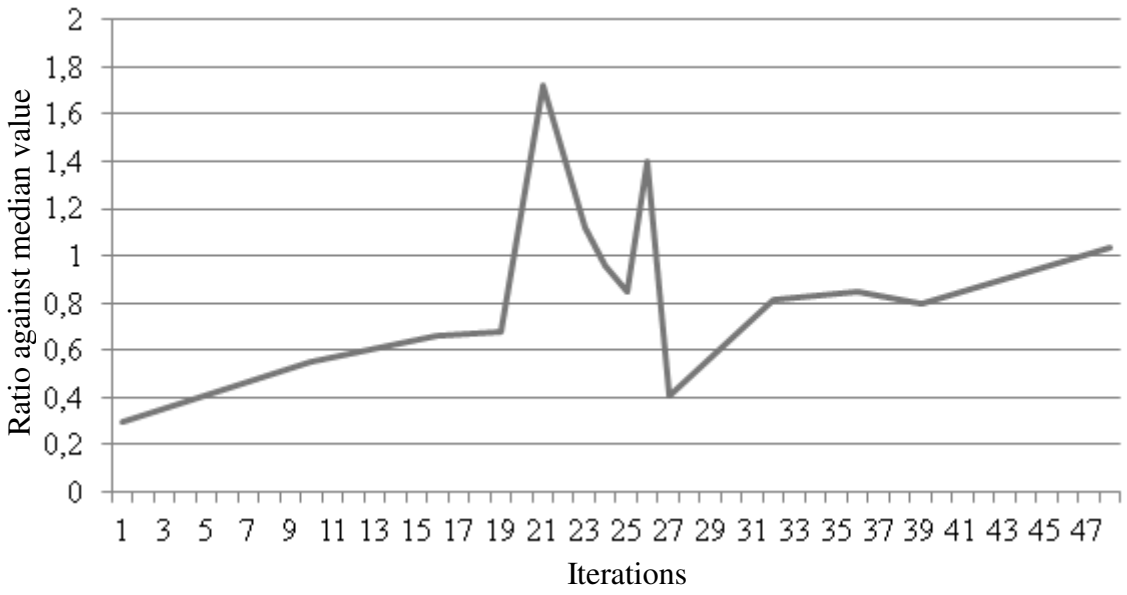

Figure 13: Music application launch reaction times before a reboot at various stages of the device use

As it can be seen, there is a growing trend toward the reaction time getting longer and deviating more from the median time. Thus, the user will experience the software aging in device as being slower than the initial experience as it took more time to launch the application.

Again, in Figure 14, the data represented are from launches following the reboot. The measurements began at device uptime 177878 seconds and 709098.17 seconds of idle time, and the measurements were continued through device uptime 668112 seconds and idle time 2469602.49 seconds. There seems to be improvement in launch time as can be seen from the highest value of 1.4 ratio which in previous case was 1.8.

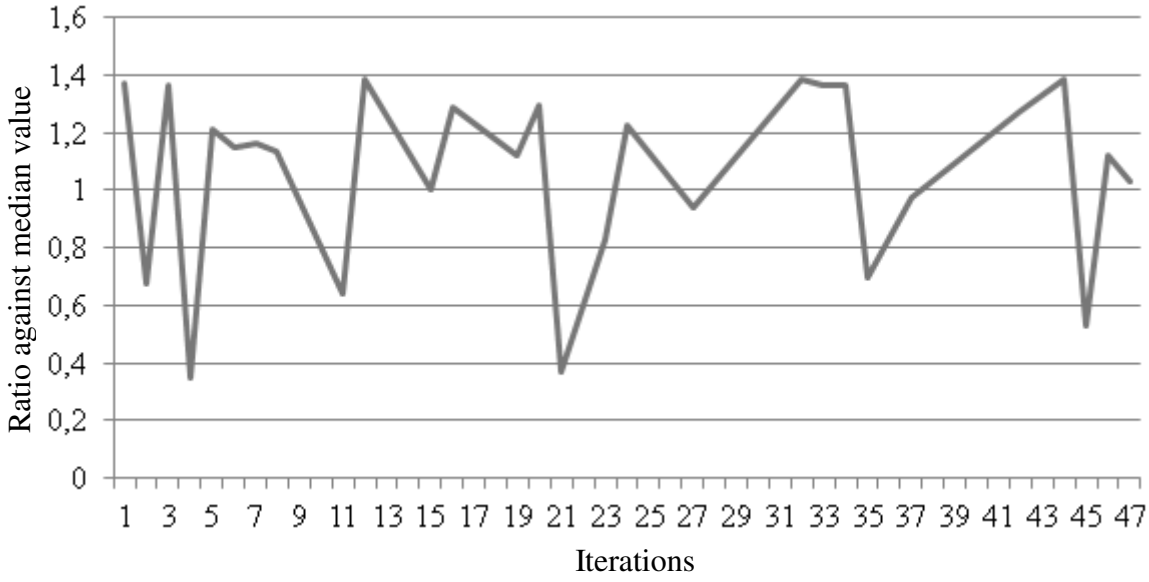

Figure 14: Music application launch reaction time after reboot at various stages of the device use

5.3. Response Time Analysis

Similarly, as was in the case of reaction time analysis, response times for 51 different median launches are plotted in Figures 15 and 16. The different response times represented in Figure 15 are from before a reboot. Every point in the graph represented a ratio of the median value of all the values before the reboot. The measurements were started when the device had its uptime at 428808 seconds and idle time at 1612018 seconds and the testing continued through until the device reached 177878 seconds of uptime and 709098.17 seconds of idle time. It shows a growing trend toward the reaction time getting longer and deviating more from the median time.

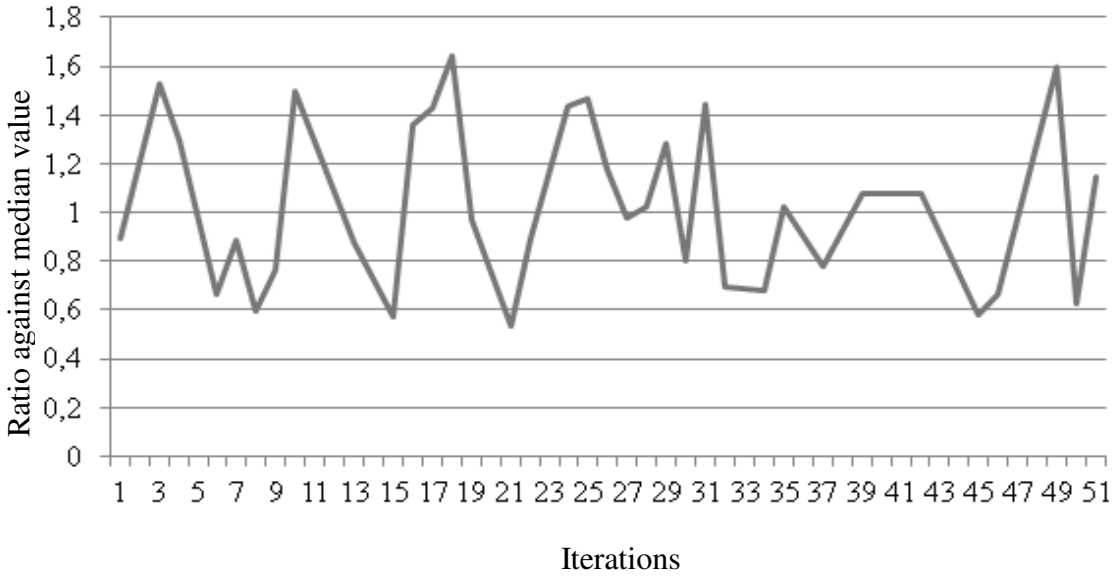

Figure 15: Music application launch response time before reboot at various stages of the device use

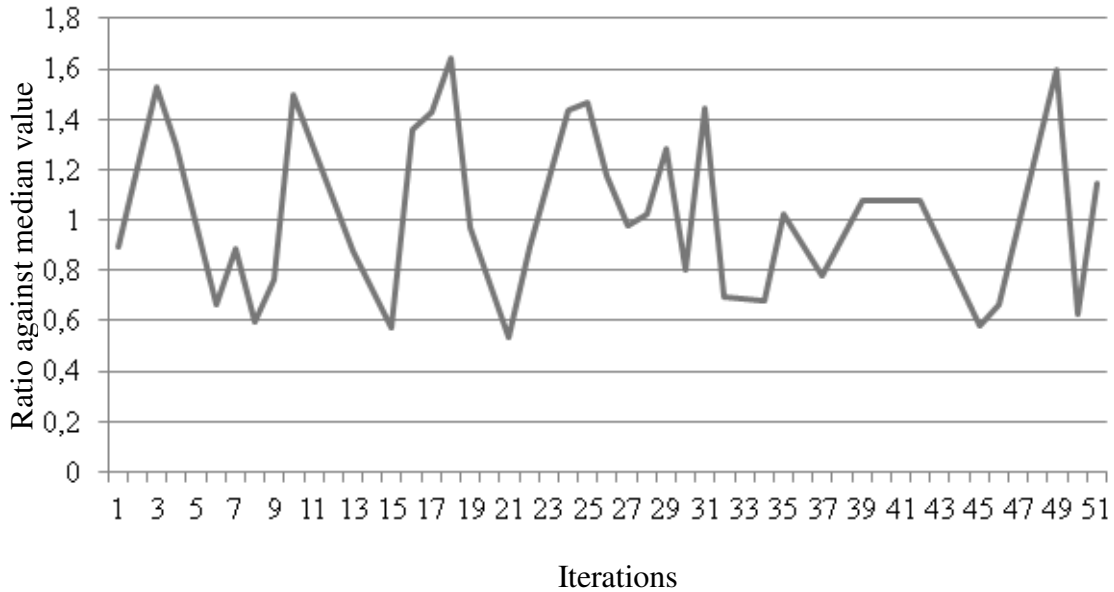

Figure 16: Music application launch response time after reboot at various stages of the device use

Again, in Figure 16, the data represented are from launches following the reboot. The measurements began at device uptime 177878 seconds and 709098.17 seconds of idle time, and the measurements were continued through device uptime 668112 seconds and idle time 2469602.49 seconds. There seems to be improvement in launch time as can be seen from the highest value of 1.4 ration which in previous case was 1.8.Thus, the user will experience the software aging in the device for being slower than the initial experience as it took more time to launch the application.

5.4. Launch Time Analysis

Even though launch time data was planned to be used to complement the data collected from the high speed camera the results showed that such data collection were not accurate enough to present the real picture. The following issues were encountered:

- The camera used to enable the machine vision capabilities were calibrated to low frames per second rate. It was done so to minimize the time taken to analyze a particular object in the view. Thus lower frame rate meant that it had less information or slow supply of information to collect from the testing than the actual frame rate of the mobile phone.

- In the course of the launch process when the actual item to recognize the application launch was completed actually appeared on the screen, it still took time to analyze the item on the screen in order to recognize it and the recognition process time varied depending on the algorithm used. Moreover, the time taken to recognize an item also were added to the actual launch time of the application.

Thus, the launch time calculation by machine vision camera always took more time than the actual launch time and failed to complement the data collected from the high speed camera.

5.5. Power Consumption Analysis

As were presented in Sections 5.2 and 5.3 for reaction and response time analysis, Figures 17 and 18 represent a set of 57 iterations for power consumption measurements. Each of these iteration values are presented as ratio of the median of all the values. It is also to be noted that each iteration value is the median of three launches.

In Figure 17 the data is represented from launches between device uptime 428808 seconds and device idle time 1612018 seconds through device uptime 770623 seconds and device idle time 2876343.83 seconds before the reboot. The data shows a rising trend of power consumption as the device was being used over the period. This would mean that the mobile device would lose battery comparatively more quickly than the cases in the beginning.

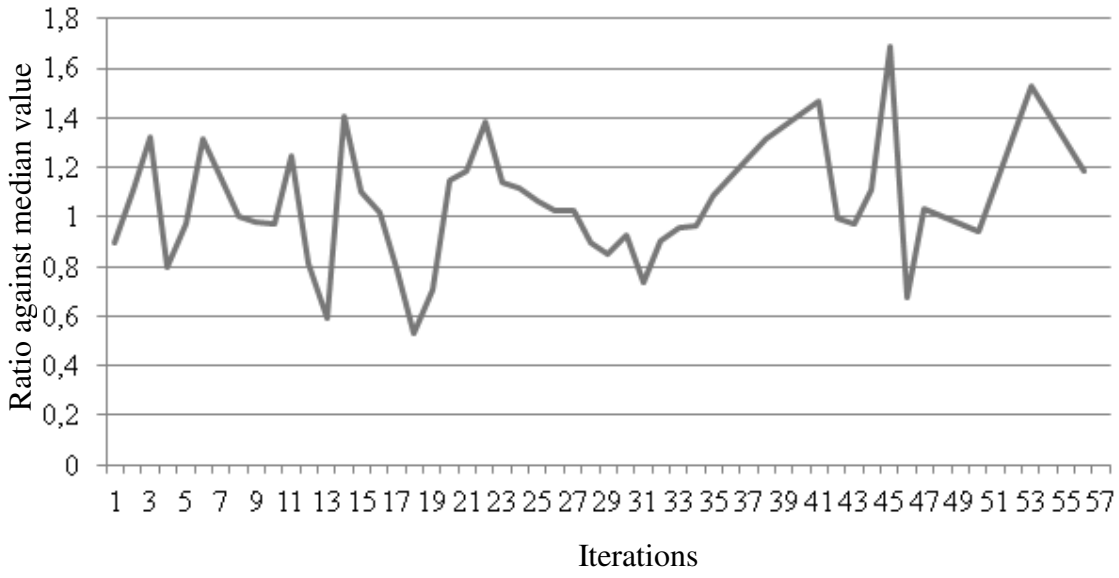

Figure 17: Music application launch power consumption before reboot at various stages of the device use

Figure 17 represents the power consumption information during the following span of the device use: device uptime 177878 seconds and device idle time 709098.17 seconds through device uptime 668112 seconds and device idle time 2469602.49 seconds.

A similar trend is also noticed in the Figure 18 as the power consumption increases with the time in the course of the device use. Thus, they seem to show that the further the device is in use from the last reboot the faster the power consumption will increase.

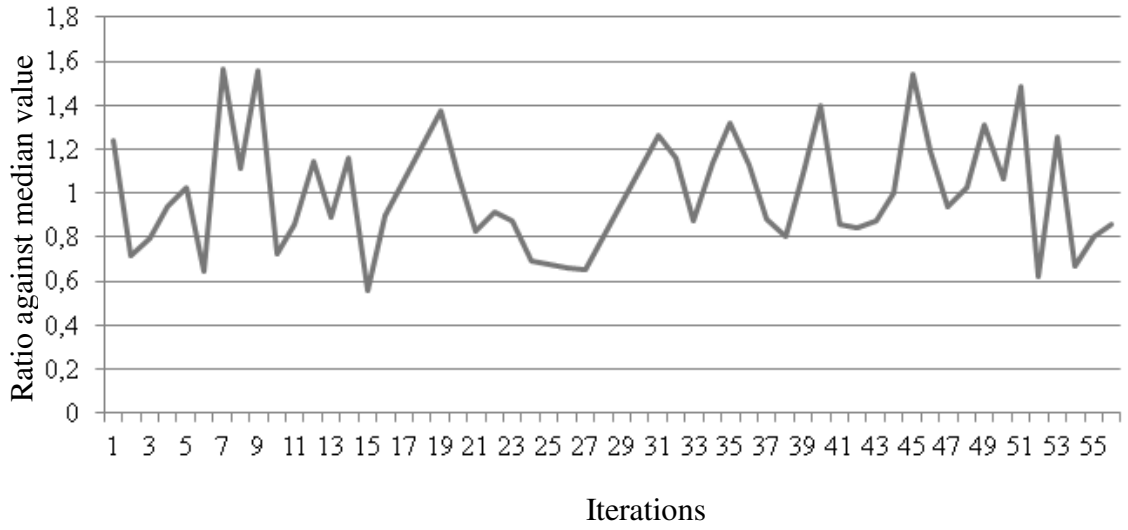

Figure 18: Music application launch power consumption following reboot at various stages of the device use

However, as the mobile phone is designed to share its resources with all the processes that are run on the device more research is needed to determine what are the processes being run in the back ground and how much each of the processes consume power besides the application launch.

5.6. Testing Beyond Stock Android

It is evident that the scope of Android platform in mobile devices and its various customized versions offered by the vendors on their devices are many [Jones, 2013] and testing different devices go beyond the current scope of study. However, the features of this testing method are equally applicable to other devices just as well as it applied to the mobile phone running the original version of Android or the stock Android available for Nexus 4 devices [Google Developers, 2014].

The testing method used in this study was operating system agnostic and it did not require any direct communication with the device, e.g., adb commands to simulate user interaction on Android platform [Android Developers, 2014], use of Touch Injection API to simulate touch input in Windows Phone [Microsoft TechNet, 2014] and such. The high speed camera used to measure changes on the device screen was also controlled by external and commercially available software that requires no direct

contact with the device. In order to make the necessary power consumption measurements external power supply unit was used instead of the battery included in the device but it had no effect on the devices operating system processes nor on the user device interaction other than providing the required current which battery included in the device could have done. Therefore, this testing method is highly recommended for any mobile device that can be placed on the table within the reach of the robot arm.

6. Summary

Using an out of the box Android mobile phone, LG Nexus 4, in a robot controlled environment a series of test were performed to understand how a user experienced software can be measured. Furthermore, the robot environment was constructed to simulate human interaction with the device without directly influencing the regular process of the mobile device, e.g., adb commands or monkey runner [Android Developers, 2014]. An external power supply was also used to provide power instead of the battery included in the device to measure the power consumption.

The results show that the test environment can be used to test any mobile device because of its operating system agnostic approach to testing. Reaction and response time are good indicators of how over time the device ages. Power consumption data collected also provides some direction into how the aging can be further measured. This chapter presents a summary of the study in terms of the following main areas: operating system agnostic test environment, reaction time, response time and power consumption. Finally, some suggestions are also made on the future continuation of the work.

6.1. OS Agnostic Test Environment

The test environment was set up from three major perspectives to execute test on the device under test namely: test design and development, test execution system setup and the robotic environment that simulated an end user interaction with the device.

Python programming language was used in addition to various other libraries that enabled features of image recognition, robotic API command execution instructions to administer robot actions in various tests of launching 13 applications as the combination was the best suit to make use of the freely available libraries required to get the most benefits of the robotics and data gathering process. In the test development process the data to be gathered was also outlined and documented.

The test script to execute the test was written keeping in mind that there can be changes in the test data and thus was kept as modular as possible to minimize the maintenance requirements. The test execution scheduler in the setup allowed for all of the test or some parts of it to be executed in any number of iterations at any possible intervals without aborting an ongoing test process.

The robotic arm in the execution system setup was capable of moving in the axes of x, y and z and was set up on a table big enough for the robot arm to move freely over it. This came as a tremendous advantage as the table could hold regular mobile devices, tablets and large touchscreen displays. The operating system independent web API commands enabled for guiding the robot to perform operations like tapping, double tapping, swiping and such in a completely non-intrusive manner just as a human end user would.

As a result, the device being tested was completely separate from the robot operations rather the robot was used to simulate human interaction with the device using python code in unit testing set up. Furthermore, an external power supply unit was used to provide power to the device eliminating the need for using the battery the device initially was equipped with. High speed camera was also used to measure changes on the device screen using commercially available software that was completely independent of the device being tested. Thus the test environment is equally implementable to any device with touchscreen user interface that can be fit on the robot arm station and in the reach of the robot arm.

6.2. Reaction Time, Response Time and Power Consumption

Reaction time was measured using high speed camera and commercially available software was used to calculate the duration. Reaction time, the time taken to by the high speed camera to recognize the first sign of change, was found to be different for different applications. The reaction time was also different in different cases of the application launch like: cold launch or warm launch, while cold launch, launching for the first time after a reboot, took longer but warm launch, launching the application after it was first launch and then left to be in the background before it was re-launched, took less time.

However, the Android operating system automatically removes applications that have been least-recently used when the recently used applications list get longer. In the present scope of the study warm and cold launches for each application were not especially considered. Nevertheless, reaction time is a good indicator of how a user might notice how a particular application gets slow in launching, hence indication of software aging.

Response time, the time determined by the high speed camera as the total period to launch an application, also gave good indication of software aging. In addition to the

issues of warm and cold launches the response time was also prone to issues like Internet, GPS, accelerometer, Bluetooth accesses and such. For example, launching a map application just does not depend on the software launch of the application but also the availability of the Internet access, GPS positioning and such other services so that a user can make use of the application. In its current scope, the study did not consider those issues. Future testing could include ways to provide steady Internet access and simulated GPS position information so that more accurate measurements can be made.

A commercially available power supply unit was used to power the mobile phone instead of the attached rechargeable battery. This external power supply unit enabled for supplying a controlled amount of voltage and current to the device and thus allowing to measure the power consumption in watts. The process of attaching the external power supply was simple and is applicable to any device running on battery to measure power consumption irrespective of the operating system on the device.

6.3. Automated Testing vs. Actual User Testing

This study was conducted in an automated test environment without a direct need for human intervention as users because the tests that were executed simulated human end user interactions. Thus, it varies from the regular usability testing where user-centered interaction is used to evaluate a product by testing it on users.

However, the central idea of the usability testing is well in line with this study as the test retains the process of how real users use the system throughout the execution [Nielsen, 2012]. Furthermore, the goal of this project was to establish a test environment where user experience software aging can be tested and measured through a repeated process of launching applications on a mobile device after the last booting to the next reboot.

The repeated nature of the testing also discourages a human user from participating in the test as it is frustrating for one to repeatedly launch and close an application on a mobile phone without the freedom to use the application as the user chooses. On the other hand, an automated test is very precise in following a set of steps to launch an application compared to a human user and this accurate repetition was necessary to eliminate any errors in launching an app to determine how each launch differentiates from other. The way how application launches differ was the key to understanding user experienced software aging in this study which was done under systematic observation with controlled conditions.

For the sake of this study actual human user involvement in the testing would have added no significant improvement to the outcome. The automated test environment was also established to go beyond operating system specific testing as it uses non-intrusive interactions as a human end user would without the need for a human tester to learn the new operating system to use it and be vulnerable to the subtle biases that affect moderated and un-moderated usability tests [McCarney, 2007].

6.4. Future Improvements

Owing to the level of availability of resources and time constrains the tests were limited to the specific factors of application launches from an end user's perspective. However, user experienced software aging can be further explored by collecting wider set of data at various levels using wider choice of devices. Different devices could be further tested in various sizes, makes and features. A variety of interaction methods may also be considered, e.g., interaction with stylus, active stylus, electronic pens and such. Tests could also be extended to a wide range of devices that are operated by different operating systems.

As the applications used in this study are general purpose applications a similar application or their equivalents can also be found in almost all the major mobile operating systems, e.g., iOS, Windows Phone, BlackBerry, Kindle and such [Netmarketshare, 2014]. For each application tested both the warm and cold launches can be separately tested, and the tests can be executed for an extended period of time, e.g., six months or a year. Internet access can be further controlled to provide a steady access over Wi-Fi with a set range and bandwidth, location information could be simulated so that the process of getting GPS information should not interfere with application process.

Third party applications could be installed to measure how it affects the system resources and memory usage and finally the device performance. In addition to automated testing human user testing could also be engaged to compare against the results acquired by automated tests.

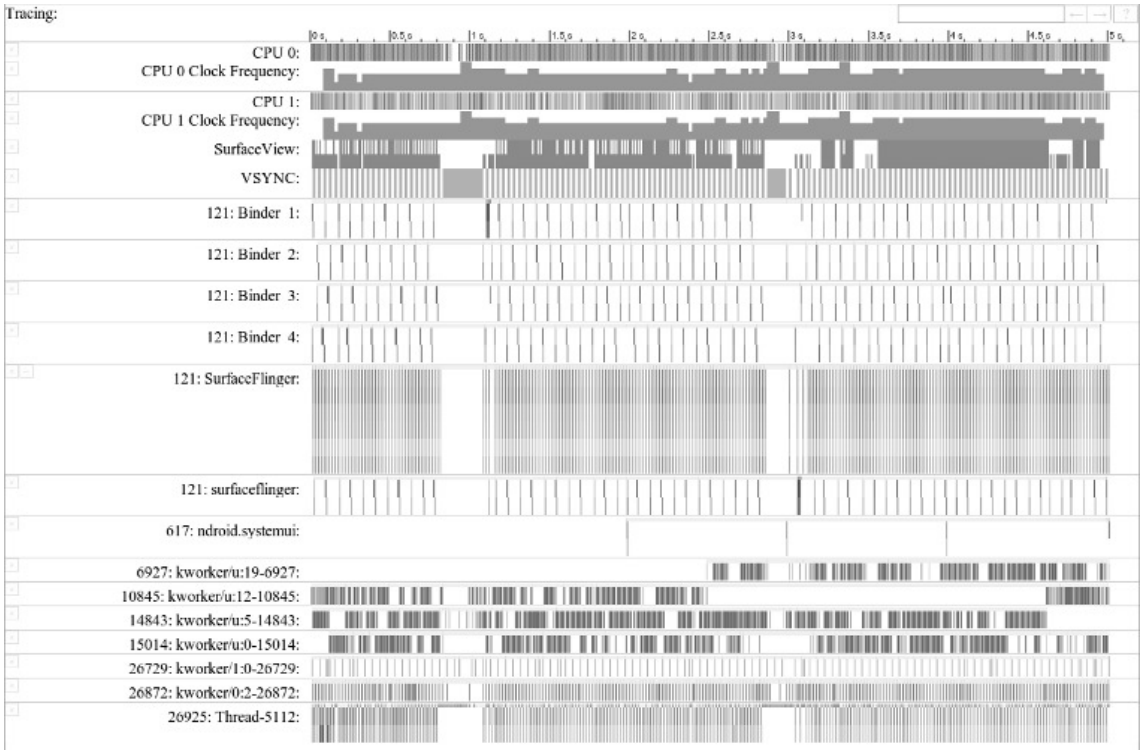

Figure 19: Systrace report collected for 5 seconds of process execution

Tools like Systrace [Android Developers, 2013] can be further used to execution data for various applications and the Android system. Figure 19 shows a detailed color coded view of system trace report collected for 5 of a process execution. Random access memory (RAM) can also be further investigated to learn more about memory usage as mobile devices contain much less RAM memory and each application shares them for their various stages of the application.

Dalvik in Android is designed to perform garbage collect which can be further investigated to learn more about when and where various applications allocates and releases memory. Application switching can further be measured as to how the mobile device performs over time in quickly switch between apps and how much memory consumption is recorded for each application at various times following the last reboot which then further could be compared to the similar state following restart.

7. Conclusion

This study was made to set up an operating system agnostic testing environment and run a set of tests on the mobile phone to measure how a user would experience software aging from one booting of the mobile device until a reboot is applied. The tests were designed and developed to be executed in an automated test environment without the direct need for a human user to be involved.

Furthermore, the test steps were designed to simulate a human end user interactions with the device, e.g., to launch an application on the mobile phone using gestures like tap, double tap and such with a test robot. Various application launch information though out the repeated launches from one booting of the system to the next reboot was successfully collected to measure how each launch differentiates from one to the other.

A mobile device with stock Android OS was tested in the test environment and application launch data was collected. Furthermore, the test execution was carefully designed and developed to make the testing just as non-intrusive as a human end user's interactions with the test device. Due to this feature of operating system agnostic automated testing, the test environment can further be extended to any other mobile devices that do not run on Android operating system.

The operating system independent feature of testing automation also liberates human testers from the burden of learning a new operating system and its use to actually measure the user experienced software aging of the particular test device. The test environment was further established using freely available software, commercially available testing robot and equipment thus making it easy and feasible for future implementation and integration in mobile device development as well as in user experienced software aging measurements.

A set of further improvement suggestions were also made in the previous chapter to enable more accuracy and wider range of coverage over issues impacting the software performance that resulting in user experienced aging of the software.

References

[Ahonen, 2014] T. Ahonen, TomiAhonen Almanac 2014. Retrieved from http://communities-dominate.blogs.com/brands/2014/02/final-2013-smartphone-market-share-numbers-full-year-and-quarterly-q4-data-by-top-10-brands-plus-os-.html. Accessed on March 2014.

[Android Developers, 2013] Android Developers, Analyzing Display and Performance, Developers Tool for Android. Available from http://developer.android.com/tools/debugging/systrace.html. Access on December 2013.

[Android Developer's Website, 2013] Android Developer's Website, Building Your First App http://developer.android.com/training/basics/firstapp/index.html. Accessed on November 2013.

[Android Developers, 2014] Android Developers, Monkeyrunner, Developers Tool for Android. Available from http://developer.android.com/tools/help/monkeyrunner_concepts.html. Accessed on March 2014.

[Basili and Perricone, 1984] V. Basili and B. Perricone, Software Errors and Complexity: An Empirical Investigation. Communications of the ACM 27 (1), 1984, pages 42-52.

[Cotroneo et al., 2010] D. Cotroneo, R. Natella and R. Pietrantuono, Is software aging related to software metrics? In: Software Aging and Rejuvenation (WoSAR), 2010 IEEE Second International Workshop, 2010, pages 1-6.

[Damm and Lundber, 2006] L. O. Damm and L. Lundberg, Results from introducing component-level test automation and Test-driven Development. Journal of Systems and Software 79 (7), 2006, pages 1001-1014.

[Dev Center – Windows, 2014] Dev Center – Windows, Windows Phone Application Analysis for Windows Phone 8, 2014. Retrieved from http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202934(v=vs.105).aspx. Accessed on April 2014.

[Duka and Hribar, 2010] D. Duka, and L. Hribar, Fault Slip Through measurement in software development process. In Proceedings of ELMAR, 2010, 177-182.

[Fewster and Graham, 1999] M. Fewster and D. Graham, Software Test Automation, Effective Use of Test Execution Tools, Addison-Wesley, 1999.

[Forlizzi and Battarbee, 2004] J. Forlizzi and K. Battarbee, Understanding experience in interactive systems. In Proceedings of the 2004 Conference on Designing Interactive Systems (DIS 04), 2004, page 261.

[Futuremark, 2014] Futuremark Hardware Channel, LG Nexus 4 Review, 2014. Retrieved from http://www.futuremark.com/hardware/mobile/LG+Nexus+4/review. Accessed on March 2014.

[Gartner, 2013] R. van der Meulen and J. Rivera, Gartner Says Worldwide IT Spending on Pace to Reach 3.7 Trillion in 2013, July 2013. Retrieved from http://www.gartner.com/newsroom/id/2537815. Accessed on January 2014.

[Google Developers, 2014] Google Developers Website, Factory Images for Nexus Devices, https://developers.google.com/android/nexus/images. Accessed on January 2014.

[Google Developers, 2014] Google Developers Website, Factory Images "occam" for Nexus 4, https://developers.google.com/android/nexus/images#occamjwr66y. Access on January 2014. Access on January 2014.

[Gray, 1985] J. Gray, Why do computers stop and what can be done about it? Technical Report 85.7, PN87614, Tandem Computers, Cupertino, 1985.

[Gray, 1990] J. Gray, A Census of Tandem System Availability Between 1985 and 1990. IEEE Transactions on Reliability 39 (4), 1990, pages 409-418.

[Grottke et al., 2008] M. Grottke, R. Matias Jr. and K. S. Trivedi, The Fundamentals of Software Aging. In: Proc. 1st International Workshop on Software Aging and Rejuvenation, 19th International Symposium on Software Reliability Engineering, IEEE, 2008, pages 180-189.

[Hassenzahl and Tractinsky, 2006] M. Hassenzahl & N. Tractinsky, User Experience - a research agenda [Editorial]. Behavior & Information Technology, 25(2), 2006, 91-97.

[Hiteside and Wixon, 1987] J. W. Hiteside and D. Wixon, The dialectic of usability engineering. In: INTERACT 87 – 2nd IFIP International Conference on Human – Computer Interaction, 1987.

[Huang et al., 1995] Y. Huang, C. Kintala, N. Kolettis, and N. Fulton, Software rejuvenation: analysis, module and applications. In: Proc. Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, 381–390.

[iOS Developer Library, 2013] iOS Developer Library, Performance Overview: Performance Tools, 2013. Retrieved from https://developer.apple.com/library/ios/documentation/Performance/Conceptual/PerformanceOverview/PerformanceTools/PerformanceTools.html#//apple_ref/doc/uid/TP40001410-CH205-BCIIHAAJ. Accessed on March 2014.

[Jones, 2013] C. Jones, Android Fragmentation: Samsung Is Dominant But Will That Always Be The Case? 2013. Retrieved from http://www.forbes.com/sites/chuckjones/2013/12/15/android-fragmentation-samsung-is-dominant-but-will-that-always-be-the-case/. Accessed on January 2014.

[Kaaresoja and Brewster, 2010] T. Kaaresoja and S. Brewster, Feedback is… Late: Measuring Multimodal Delays in Mobile Device Touchscreen Interaction. In: Proceedings of ICMI-MLMI'10, Beijing, November 2010, pages 8-12.

[Law et al., 2009] E. L. C. Law, V. Roto, M. Hassenzahl, A. P. Vermeeren and J. Kort, Understanding, scoping and defining user experience: a survey approach. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems ACM, 2009, 719-728.

[Mahmood et al., 2000] M. A. Mahmood, J M. Burn, L. A. Gemoets and C. Jacquez, Variables affecting information technology end-user satisfaction: a meta-analysis of the empirical literature. International Journal of Human-Computer Studies 52 (4), April 2000, pages 751-771.

[McCarney, 2007] R. McCarney, J. Warner, S. Iliffe, R. van Haselen, M. Griffin and P. Fisher, The Hawthorne Effect: a randomized, controlled trial. BMC Med Res Method, 7 (30), 2007, pages 645-657.

[Meeker and Wu, 2013] M. Meeker and L. Wu, 2013 Internet Trends. In: Internet Trends D11 Conference 2013. Web. http://www.kpcb.com/insights/2013-internet-trends.

[Meeker et al., 2013] M. Meeker and L. Wu, Kleiner Perkins Caufield Byers, 2013 Internet Trends, 2013. Retrieved from http://www.kpcb.com/insights/2013-internet-trends. Accessed on January 2014.

[Merholz, 2007] P. Merholz, Peter in Conversation with Don Norman About UX & Innovation. Adaptive Path, 2007. Retrieved from http://www.adaptivepath.com/ideas/e000862/. Accessed on December 2013.

[Microsoft TechNet, 2014] TechNet Wiki, Simulating Touch Input in Windows 8 Using Touch Injection API. TechNet Articles, 2014. Retrieved from http://social.technet.microsoft.com/wiki/contents/articles/6460.simulating-touch-input-in-windows-8-using-touch-injection-api.aspx. Accessed on March 2014.

[Miedes and Muñoz-Escoí, 2012] E. Miedes and F. D. Muñoz-Escoí. Dynamic Software Update. Technical Report ITI-SIDI-2012/004. 2012.

[Netmarketshare, 2014] Netmarketshare, Market Share Statistics for Internet Technologies, Mobile/Tablet Operating System Market Share, 2014. Retrieved from http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1. Accessed on April 2014.

[Nielsen, 1995] J. Nielsen, Usability inspection methods. In Conference Companion on Human Factors in Computing Systems. CHI '95. ACM Press, New York, NY, 377-378.

[Nielsen, 2012] J. Nielsen, Usability 101: Introduction to usability. NN/g Nielsen Norman Group, January 2012. Retrieved from http://www.nngroup.com/articles/usability-101-introduction-to-usability/. Accessed on December 2013.

[Nightingale et al., 2011] E. B. Nightingale, J. R. Douceur, and V. Orgovan, Cycles, Cells and Platters: An Empirical Analysis of Hardware Failures on a Million Consumer PCs, Microsoft Research, 2011.

[Parnas, 1990] D. L. Parnas, Education for Computing Professionals, IEEE Computer 23 (1), January 1990, 17-22.

[Pienaar and Hundt, 2013] J. A. Pienaar and R. Hundt, JSWhiz: Static Analysis for JavaScript Memory Leaks. In. Proceedings of the IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 2013, pages 1-11.

[Sessions, 2009] R. Sessions, Object Watch, A White Paper by Roger Sessions on The IT Complexity Crisis: Danger and Opportunity, 2009. Retrieved from http://sistemas.uniandes.edu.co/~isis4617/dokuwiki/lib/exe/fetch.php?media=principal:itcomplexitywhitepaper.pdf. Accessed on December 2013.

[Sommerville, 2007] I. Sommerville, What is software? Software Engineering (8th ed.). Addison-Wesley, 2007.

[Tobias, 1995] P. Tobias and D. Trindade. Applied Reliability, 2nd edition. Kluwer Academic Publishers, Boston, 1995.

[ZDNet, 2013] Microsoft pulls Windows RT 8.1 update from the Store http://www.zdnet.com/microsoft-pulls-windows-rt-8-1-update-from-the-store-7000022144/. Accessed on December 2013.