

Iiro Rassi

# Test Platform Prioritization for Smartwatch Game Development: Case Everywear Games

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

11 May 2016

Author(s) Title Number of Pages Date	Iiro Rassi Test Platform Prioritization for Smartwatch Game Development: Case Everywear Games 45 pages + 1 appendix 11 May 2016
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Mika Tammenkoski, Chief Technology Officer Teemu Saukonoja, M.Sc.
<p>Testing in game development does not follow the same priorities and processes as in common software development. Game testing places a larger emphasis on usability aspects such as the fun-factor of the gameplay. Due to the different quality criteria, the approach to test planning is different as well. This thesis explores the possibility of prioritizing the testing platform based on the testing task at hand in order to improve the efficiency and reduce the resource costs of the test execution.</p> <p>Testing platforms in this context refer mainly to the simulator provided by the application development software and a real device. The client of the thesis, Everywear Games, develops games for the Apple Watch smartwatch. As a growing company they were interested in building up their processes in the most efficient way.</p> <p>The primary goal of the thesis was to formulate a set of principles for conducting test platform prioritization in a smartwatch game development project. This goal was fulfilled by the contents of the thesis. There were many discrepancies found between the operation of the simulator and the device, which were categorized as performance, physical and functional differences. These guidelines can be applied to other mobile development projects in other fields as well, if the quality criteria are similar.</p> <p>The secondary goal was to investigate the testing of a smartwatch game project in a real world environment. This goal was fulfilled by the testing conducted on the client's game applications. The findings of the investigation supported the theory, and showed no difference in the test results between the simulator and an actual Apple Watch device. This presents the tester with the possibility of selecting the most efficient tool for each testing task. The investigation was conducted with functional tests and, due to the aspects discussed in the theory section, other test types could provide more varied results.</p>	
Keywords	testing, game testing, game development, smartwatch, Apple Watch, test platforms

## Contents

Abstract

Table of Contents

1	Introduction	1
2	Testing in Game Development	4
2.1	Testing for Fun	4
2.2	Measuring Quality through Analytics	5
2.3	Test Resources	7
2.4	Test Levels	8
2.5	Defect Types	11
3	Software Platforms for Wearable Mobile Devices	13
3.1	Overview of Market	13
3.2	Apple Watch	16
3.3	Android Wear	18
4	Test Platforms in Mobile Application Development	20
4.1	Overview of Platforms	20
4.2	Relative Comparison of Simulators and Emulators	21
4.3	Testing on Physical Devices	22
5	Principles of Test Platform Prioritization	24
5.1	Limitations of Simulators	24
5.1.1	Performance Differences	26
5.1.2	Physical Differences	26
5.1.3	Functional Limitations	28
5.2	Efficiency and Preserving Resources	28
5.3	Utilizing Test Platform Prioritization	29
5.3.1	Practical Guidelines	29
5.3.2	Test Planning	30
5.3.3	Risk-Based Testing	33
6	Investigation	35
6.1	Concept and Setup	35
6.2	Findings	36

7	Conclusion	39
	References	41
	Appendices	
	Appendix 1. Defects Found in the Investigation	

## 1 Introduction

Mobile devices have changed the way we communicate with each other and after the invention of smartphones also how we interact with computers. Mobile devices have also greatly affected the way we play games. Out of all the applications available for users of Apple's smartphones today, 22% are games, making it the largest application category in Apple's App Store [1]. It might come as no surprise then that in the newest segment of mobile devices, the smartwatches, the same is true with games being the largest category for Apple Watch applications with a 12% share [1]. The comparison between smartphones and smartwatches is not completely straightforward though, since the platform, its applications and their use case are very different.

Game development as an industry has been growing rapidly in Finland after the success of Rovio and Supercell. There are approximately 260 active gaming companies in Finland, and 69 % of them are less than 5 years old [2]. This thesis was commissioned by a Finnish developer of smartwatch games, Everywear Games. Within their first year the company released two highly successful games for the Apple Watch, which are constantly updated with new content and features after their initial release. As a young company working with new technology they are interested in setting up their processes in the most efficient way. This study aims to find the best way to share the execution of testing tasks between simulators and actual devices. Testing with the device for which the finished product is targeted provides the most accurate results, but it also requires more time and the access to each of the targeted devices. The devices might not be readily available for testing, due to high cost or even not having been released to the public yet. When the Apple Watch was released in April 2015, many of the third-party applications available at the time had been developed by developers who did not have access to the actual device at all [3;4].

Game development shares many similarities with traditional software development. The end-product is a software application, and it is created using many of the same tools as other software projects. The dissimilarities come with the emphasis on content and user experience over technical functionality, which creates different kind of development process. This thesis adopts this perspective while investigating a specific

challenge faced in mobile development, and more topically with new first generation wearable mobile devices.

The practice of test platform prioritization for mobile application development is not commonly described in literature. Usually the literary guides for software development or testing of mobile applications merely mention that the application must be tested on actual devices in addition to the simulators provided by the software development suite. To receive absolute clarity of the quality of a mobile application, it is important to thoroughly test it with each of the devices it is intended to support. However, due to resource restraints or prioritization of other application requirements absolute clarity is not always desired. This is often the situation in game development projects, which makes them more suitable for practicing test platform prioritization. Some of the specific challenges and practices of testing in game development projects are more thoroughly presented in **Chapter 2** of this thesis.

Smartwatches and other wearable mobile devices are some of the newest types of consumer electronics and personal computing devices in the market today. The applications and their development process must be completely tailored to this new hardware platform, and for games especially there has been much doubt from both the industry and the public of how useful or rewarding these platforms are. While gaming applications are hugely successful for mobile devices, the same ideas and execution will not work on the smartwatch. **Chapter 3** discusses the different software platforms available for smartwatches and the individual challenges relating to testing each of them.

When less than total coverage is expected for the quality assurance efforts, the question arises of how to reach this target most efficiently. Test platform prioritization means selecting for each stage of testing the platform that will provide adequate results while minimizing repetition and the overlapping of testing tasks. Test platforms available for mobile applications and especially smartwatch applications are explained in **Chapter 4**.

Each software development project will have unique quality expectations and definitions of adequate quality for each stage of testing. The plan for test platform prioritization and its execution will therefore also be unique for each project. The goal of this thesis is primarily to provide guidelines that are generally applicable for prioritizing

test platforms in the production of game applications for mobile devices. These guidelines can also be applicable in other software development projects with similar quality requirements. The theory of test platform prioritization is presented in **Chapter 5**.

The secondary goal of the thesis is to examine the practical application of these theories to the functional system testing of a smartwatch game. **Chapter 6** introduces a case study aimed at fulfilling this goal.

## 2 Testing in Game Development

This chapter discusses some common challenges and distinct characteristics of testing and quality assurance in game development. Games, whether computer, mobile or console games, are software applications in their core, but their development usually does not follow the same kind of process as traditional software projects. Games have rich multimedia content and often aim to relay a story to the user, so the technical implementation is only part of the equation.

### 2.1 Testing for Fun

The usability aspect of games is strongly emphasized as game developers consider the fun-factor to be the top quality criterion. A technically perfect game that is not fun will not be successful, while on the other hand users are willing to overlook some technical issues if they are otherwise enjoying themselves. This accentuation of non-technical requirements has led to a decreased effort in technical testing when compared to other software projects.

As stated, the primary requirement for basically every game product is fun. Even if this fun-factor were to be refined into measurable qualities such as player retention, which refers to the number of users that continue to use the product after a certain amount of time, it would still provide an inefficient basis for test planning. The success of technical testing tasks is properly evaluated only when it is based on technical requirements.

More important in terms of the fun-factor are the cycles of the gameplay. Especially in mobile games it is common to repeat a certain game mechanic for a while, then reward the player with some kind of advancement and start another cycle, either with the same mechanic or another one. This is most apparent in roleplaying games where the player gathers experience points by executing tasks and then uses these experience points to “level up” or move to the next phase in the game. The gaming platform affects these cycles greatly. While in computer or console games it can be expected that the play session lasts 30 minutes or more, in mobile games the average sessions last only a few minutes. Most smartwatch applications are designed around the idea that the user will only spend a few seconds at a time using the application. The cycles of gameplay should fit around the average play sessions, so that the user has the best possibility to

advance in the game and enjoy the experience. While these qualities of the game are still subjective, as each user will play the game differently, they can be tested and will provide important assurance of the fun-factor of the game. It is difficult to enforce absolute measurements for game cycles, but they are very closely related to the design of the game, so their requirements should come from the initial design. Experience in the current project, or similar projects, will also provide the tester with a measurement of normal to compare their perception against. The average play sessions and testing sessions are also quantifiable in terms of time, so measuring, or at least considering, this will give some idea of how the gameplay flows.

## 2.2 Measuring Quality through Analytics

After the gaming industry's radical shift to the free-to-play, or "freemium", model of game design, where products are delivered to customers without cost but contain purchasing options inside the game, the efficient application of analytics has become vital to many gaming companies. This kind of revenue generation logic in a product is known as monetization. Analytics can provide invaluable insight into how users are using the product and, most importantly, how they are spending money inside the game. Figure 1 shows the popularity of free applications in Apple's App Store.

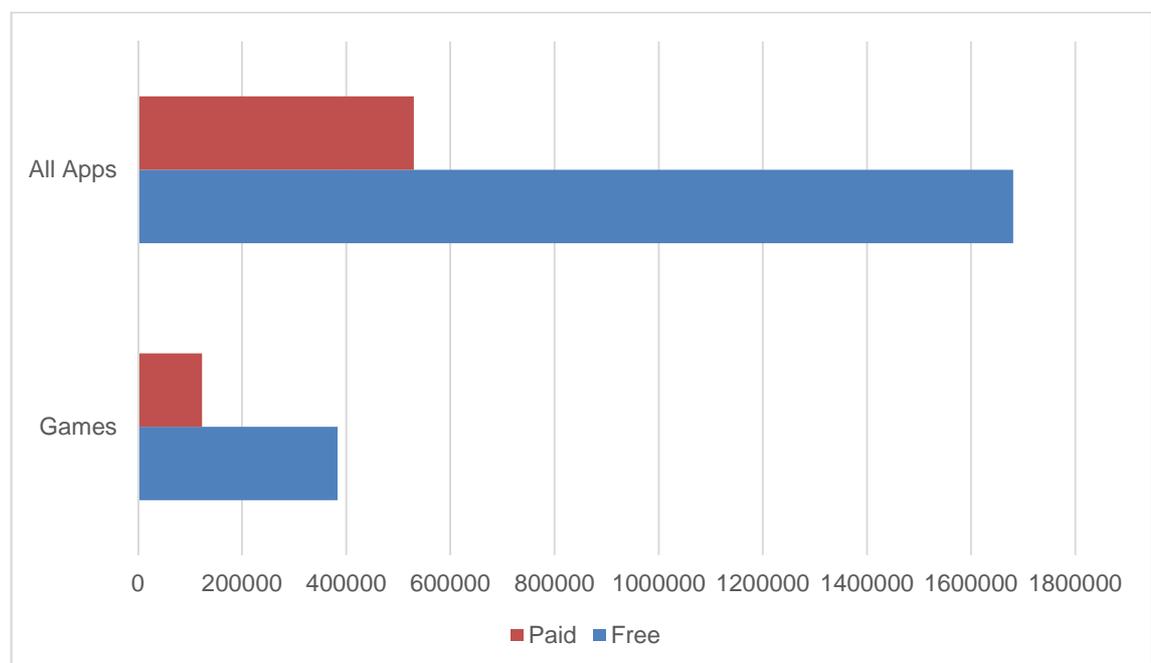


Figure 1. The amount of free and paid applications in Apple's App Store [5]

The distribution in both categories is very similar with free applications amounting to 75% of games and 76% of all applications [5]. Free applications also bring in over 90% of total application revenue [6].

A common technique associated with monetization and optimizing applications is A/B testing. This refers to making two versions, A and B, of a particular feature of an application and then exposing these to two different groups of users. In the context of monetization the features tested can be for instance the price of a premium item or the conversion rate of real money to in-game currency. Data of user behavior is collected and analyzed to find which version proved more successful. This can be done in a live application or service, with the users completely unaware that they are seeing something different from the other group, so it is a great tool to continuously improve the efficiency of the applications monetization throughout its entire lifecycle. Even though it has the word testing in its name, this technique has more to do with business intelligence than software testing. [7]

The success of the monetization can be one of the measures of the success of the whole product, and as such also a measure of its quality. Measuring or predicting monetization before the release of the product is very difficult, as it is mainly done through analytics, and therefore cannot be used as a basis for test planning in game development. Analytics is becoming more and more important for game development, and academic research has already been conducted for instance in predicting average playtimes of games [8] or player churn [9, 10], which refers to the rate of players abandoning the game before completion, but these topics do not fall into the scope or field of this thesis.

Another common application of analytics in mobile development is through “soft launches” of applications, meaning geographically limited releases. Due to the ease of market selection in the mobile application marketplaces operated by the software platform providers, many developers quietly launch their new titles in English-speaking markets with smaller amount of users, such as New Zealand or Canada, before the official release to a wider audience. This way they can gather feedback and statistics of the game for last minute improvements or evaluate the potential for the products success in addition to finding defects.

The soft launch can be done to an otherwise completed application with the intention of testing its potential for success. Sometimes developers might release a product slightly early to a smaller market and keep working on it for some time based on the results of the analytics before a wider release or possibly even a cancellation.

### 2.3 Test Resources

There is a common stereotype of games testing as a dream job for people who enjoy playing games in their spare time [11,17; 12,249]. Game testing is often a person's first position in the games industry, and it is used as a gateway to other positions. Because of this game testers are unlikely to remain in the position for long and, due to the amount of interest in the industry, they can be readily replaced. In bigger game development or game publishing companies this can lead to a lack of appreciation towards the position [13].

Due to their background the novice game testers can also lack knowledge of common software testing techniques, which hinders the efficiency of their technical test execution and their ability to efficiently assist the quality assurance. This problem could be alleviated or removed completely with adequate investment in training and on-boarding for the testers. Traditional software testing expertise is not always valued as highly as gaming industry or gameplay experience, but some gaming companies see benefit in both skills, like World of Warcraft creator Blizzard who have all of their testers certified by the International Software Testing Qualifications Board (ISTQB). The inclusion of in-game transactions and managing servers for online functionalities for instance are changing the quality requirements for game development and will require also more traditional software testing methods and processes. [14]

Game development projects also often utilize playtesting which is performed on a product that is complete enough to be playable [11,116]. This phase is similar to user acceptance testing or usability testing in common software development. Playtesting sessions often employ a large number of external testers and their main goal is to verify the primary quality criterion, the fun-factor. Because the testers are brought in so late to the project, they have limited knowledge of the product, and even if defects in the product are found, it might be too late in the project to fix all of them. [12,250]

## 2.4 Test Levels

The techniques used in the testing depend on the phase of the software development lifecycle. Most traditionally the test levels or stages of testing of a software development project are based on the Waterfall model of software testing, shown in Figure 2.

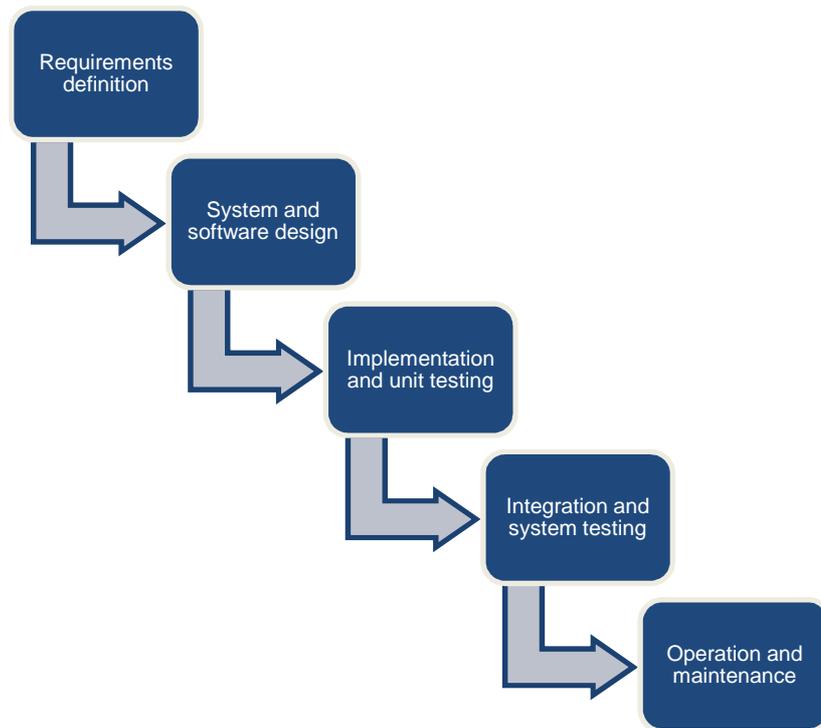


Figure 2. The Waterfall Model of Software Development [15]

Another model commonly used in software development is the V-model. It is considered an extension of the waterfall model and it incorporates the testing stages even more clearly than the Waterfall model. A representation of the V-model is shown in Figure 3.

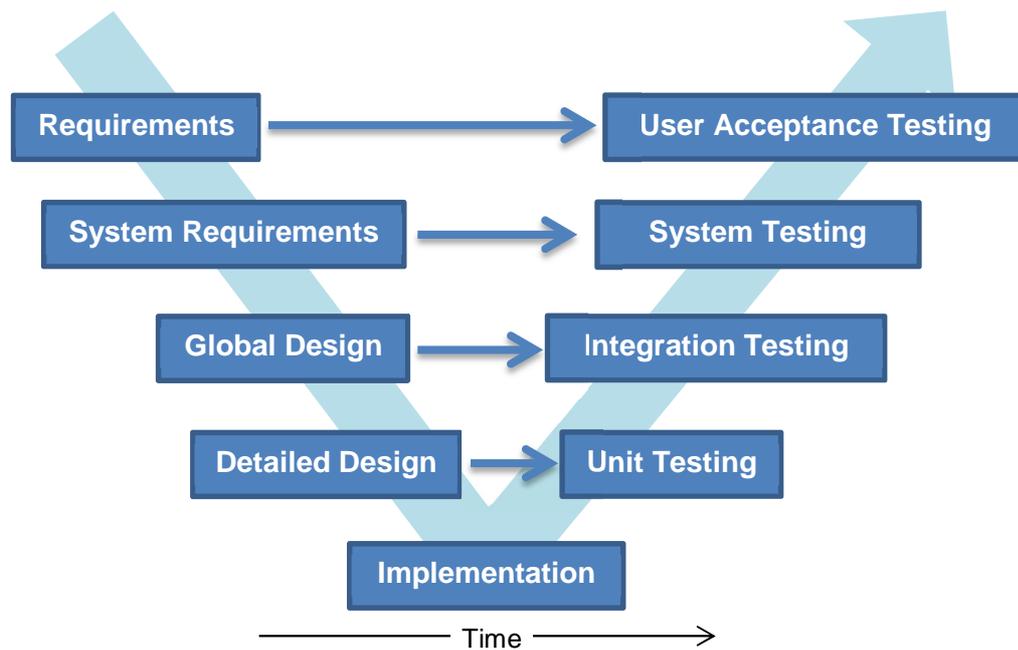


Figure 3. The V-Model of Software Development [16]

The V-model in its purest form gives the impression that the testing tasks are carried out at the end of the project when all of the development is finished. The original intention however is that the work for each test level on the right starts as soon as the basis for testing on the left has been produced. [17,13]

Regardless of the software development methodology being utilized, the testing tasks usually follow the example presented by these models. In agile development methodologies, such as Scrum, where the process is more iterative and the different stages of development are repeated throughout the project, the test levels are still divided in the same way [16].

When developers are writing parts of the software, the functionality of these individual parts need to be tested before they go further. This is called unit testing. Unit testing is often done either within the development suite or with external tools because these pieces of the application do not function on their own as they will in the final application. When these individual parts are combined to form a complete function of the application in development, integration testing is performed to validate that the parts work together as expected. After the functionalities of the application as a whole, or in a particular area of the application, are fully developed, system testing efforts are

executed to verify the application with all its parts functions to fulfil the purpose for which it was created. After system testing there is usually an acceptance test where the customer or the end-users verify whether the application meets their needs and requirements. All of these testing steps are categorized as functional tests. [17,19-24]

Functional tests also include smoke testing, which is utilized to verify that the newest build of the application that is going under testing is stable enough for the task, and regression testing, which is used to verify that the application has not regressed, meaning that new issues have been introduced to previously verified functions [11,138-139,331]. In addition to functional tests, non-functional tests can be executed. Non-functional tests verify non-functional requirements, such as performance efficiency, compatibility, reliability, security, maintainability and portability [17,334-335].

The functional testing stages of game development are similar, because the technical development methods are similar. The role of integration and system testing can be diminished, with more emphasis being put on playtesting the game after unit tests, when possible. The simplified approach to testing that is often used in game development utilizes unit tests conducted by developers in the development phase and then playtesting conducted by either the QA department or volunteer testers. This playtesting could also be categorized as system testing or acceptance testing in traditional software testing terminology, depending on how late in the project it is scheduled. Since the fun-factor and user experience are held in higher regard than absolute technical quality, getting feedback from these testing stages can be seen as more important than additional testing stages.

Game development projects also often utilize beta testing phases [11,116-119], which are comparable to acceptance testing conducted by either the stakeholders or the end-users in other software development projects. Beta testing can be internal or external. Internal beta testing is carried out with the developers own test resources using a functionally complete build of the application [11,116]. In external beta testing the developers do a limited release of the product with the expectation that defects exist and the users, acting as beta testers, will report them [18,6-7]. External beta tests can be closed or open, the former referring to a limited group of users and the latter to being accessible by all interested users.

Even though on the surface it might appear that game development commonly employs the same process as traditional software development projects in regards to testing, one key difference in these approaches is the motivation and planning of these testing efforts. In software testing the testing is typically planned on a test case level, where each test case represents one testing task. These test cases are derived from project requirements, test plan or other project documentation. The purpose of the testing is therefore to verify that the end-product meets the requirements set upon it in the beginning of the project. In game development projects it is not uncommon for these types of documents to be omitted as the project progresses organically from design to execution. Testing tasks are then carried out as needed when project functionalities become ready for testing.

## 2.5 Defect Types

Defects in games can be categorized in many different ways but these categories generally differ slightly from traditional software development projects. Games have a lot of graphic elements and graphics play a big role in game design, so game projects tend to have far more graphics related defects than other software projects. Especially in mobile development there are also more hardware specific defects, meaning defects that can only be replicated on particular device models. These are typically impossible to find in the simulator and, depending on the platform and how fragmented it is, these types of defects can be troublesome for the developers.

Defect categorization can also help in the handling of defects if they are classified based on which department is responsible for fixing them. Graphics defects are assigned to graphic artists, technical defects to developers and gameplay issues to game designers. This can speed up the handling of the defects, but it can also be difficult for the testers to recognize the correct root of the issue and some problems might require input from multiple departments. [Teemu Saukonoja, Thesis Supervisor, 8 December 2015, personal communication]

Without defect categorization it will be significantly harder to gain valuable insight from analysing the defect registry. Categorical analysis will show trends relating to different development phases and tasks and can help in improving and optimizing the production workflow within the whole company. It is possible to obtain this kind of knowledge by analysing all of the defects on a detailed level, but as the production

grows in size and amount of defects increases, it will be very difficult to keep this up. Without categorization it will also be more difficult to represent the situation numerically in order to employ analytical tools. The evaluation will be based solely on the reviewer's personal opinion. [19,453-455]

### 3 Software Platforms for Wearable Mobile Devices

This chapter presents the most popular software platforms and, due to their strong dependency, also some of the hardware platforms which are used in wearable mobile devices. A brief overview of the current smartwatch marketplace is presented with descriptions of each mentioned software platform before a more thorough analysis of the two most prominent systems.

#### 3.1 Overview of Market

A big trend in technology currently is wearable mobile devices, or “wearables”. The term “wearables” is most often linked to smartwatches, meaning mobile computers in the style of wrist watches, but they are not the only devices in this category. Other entries to this marketplace are for instance computer-assisted eyewear such as Google Glass or Microsoft HoloLens, or the popular activity monitors and fitness trackers which are usually also worn on the wrist but are not considered smartwatches due to limited functionality and lack of support for third-party applications.

Smartwatches are similar to mobile phones in that they contain powerful microprocessors and memory and have a small high quality digital screen. In addition to telling time, smartwatches are connected to a mobile phone to serve as a second screen for displaying relevant information and also have the ability to run specialized applications. The smartwatch applications can either be installed and run on the device or the processing can be done on the phone while using the watch mainly for display and user interface.

Compared to traditional application development, the development of mobile applications faces some additional challenges that relate to testing. The majority of the mobile phone market is divided between two operating systems, but within these two categories there are many different devices with different software and hardware specifications which affect how they run software. The newest version of Apple’s mobile operating system, iOS 9, supports 8 iPhone models, 10 iPad tablets and 2 iPod Touch music players [20]. All of these devices have different screen sizes, processors and memory capabilities. Updates to Apple’s operating system is offered to simultaneously to all supported devices within a certain geographical area but users

can still have different versions of the operating system in use, depending on how diligently they have updated their device [21].

It is not uncommon for desktop computers to have differing screen resolution or versions of the installed operating system, but mobile applications are more optimized for the specific platform, and therefore they can be more delicate. For instance, on a desktop computer with a Windows operating system, all applications will be run in a window. This window can be resized according to the resolution of the users system or the user's preference. Mobile applications on the other hand are always designed as full screen applications and they cannot be resized at will, so the layout of the application must be compliant with all of the devices that it can be installed on, and possibly also two different screen orientations, horizontal and vertical.

The issue of fragmentation is experienced in smartwatches as well, although so far to a lesser degree. Figure 4 displays the market shares of different smartwatch operating systems in 2015.

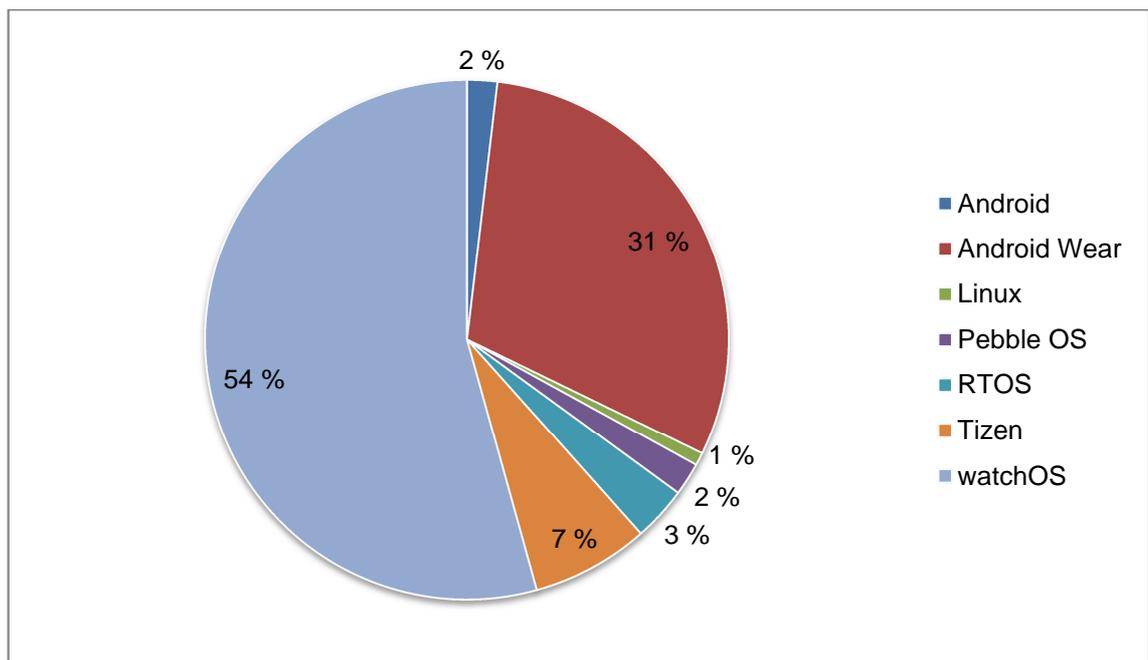


Figure 4. Market share of smartwatch operating systems in 2015 [22]

As can be seen in the graphic, Apple's **watchOS** appears to be leading the market quite clearly, even though they published their first smartwatch product only in the second quarter of the same year. These numbers are estimates, however, because

Apple has not published the actual number of their sales. The Apple Watch is currently offered in two different screen sizes, and the watchOS operating system has seen two significant release versions. Fragmentation therefore is not a major concern for Apple Watch application development at the moment.

The second largest market share belongs to Google's **Android Wear** operating system. Much like with Google's Android operating system for mobile phones, fragmentation is a much more prevalent problem. This is mainly due to Android Wear being used in devices made by multiple different manufacturers and sold by many different brands. The wide variety of devices is one of the strengths of Google operating systems, but each device has unique specifications. In addition to internal technical differences, such as Wi-Fi connectivity or GPS, the devices even come with different screen shapes, some being rectangular like the Apple Watch and others having a round screen to appear more like a traditional wristwatch. This makes it increasingly difficult to design applications that would support majority of the available Android Wear devices. [23]

This chapter examines Apple's iOS and Google's Android Wear in more detail. The other operating systems, as seen in the previous figure, hold much smaller market shares, and therefore tend to be less interesting to developers and publishers. **Tizen** is a Linux-based operating system used for a wide variety of consumer electronics and household appliances. The idea behind it is to provide a uniform user experience across all different platforms. Tizen is based largely on projects originally created by Samsung but today the Tizen Association which oversees the industry adoption of the platform contains members from many manufacturers and network operators. The operating system is largely open-source, although the licensing is not completely transparent and some parts of the platform fall under Samsung's licenses [24].

To this date the platform has mainly been utilized in Samsung devices. Samsung's smartwatch line, called Galaxy Gear, transitioned to using Tizen after their first Galaxy Gear model had shipped with Android 4.3. Galaxy Gear models shipped with Tizen are, in chronological order, Gear 2, Gear 2 Neo, Gear S and the current flagship model Gear S2 [25]. The Galaxy Gear S2 in particular is powerful enough for running game applications, but the Tizen application market is still very small and the platform is not a priority for developers. The most downloaded free application in Samsung's Gear store is a simple mobile game, Snake Classic S2 [26].

The **RTOS** segment comprises of devices running a Real-Time Operating System that supports third-party applications. One notable mention is **FreeRTOS**, an open source platform that was used in the now defunct Finnish smartwatch company MetaWatch's devices. This platform also served as the base for Pebble's **PebbleOS**. Pebble smartwatches were originally launched through crowdfunding site Kickstarter and was considered to be one of the devices that ignited the current smartwatch trend. The following Pebble models have been primarily launched through Kickstarter. The first Pebble models had a black and white e-paper screen, and the latest Pebble Time product family is equipped with a 64-color e-paper display. In addition to the low performing displays, other hardware is not as powerful as the competitors either. Pebble's strengths are more in the long battery life and flexibility in supporting different phones and applications. There is a distinct game market for Pebble as well, with mostly very simple renditions of old classics, but also some very original titles. [27]

## 3.2 Apple Watch

### **watchOS 1**

Apple has a solid track record of setting the standard for new devices in the marketplace. The mainstream demand for MP3-players and tablet computers only started to build up after the launch of the iPod and the iPad, respectively. Even when they are not the first to market with a new device, their device is usually the one that the wider audience remembers [28]. With smartwatches, many consumers were eagerly waiting for Apple's concept before making their mind about the whole segment.[29] The anticipation was so great that the news articles about an upcoming Apple smartwatch were being published months before the first announcement of such a product was made in September 2014. [30;31]

When the first generation Apple Watch was released in April 2015 it also was running the first version of the proprietary operating system, watchOS. This operating system is based on Apple's mobile phone and tablet operating system iOS. An earlier update to Apple's development software Xcode had given users the new WatchKit application programming interface that is used to create applications for the watchOS.

A very significant impediment for application development was that even though watchOS 1 contained 20 stock applications that were running natively on the watch,

aftermarket third-party applications could only be stored and processed on the connected iPhone. The first WatchKit applications essentially were iPhone applications with a WatchKit extension and a user interface on the Apple Watch. User operates the watch interface, but all the actions are sent via Bluetooth to the phone application that does the computing and sends the result back to the watch. [32]

Apple might not have been completely confident of the first Apple Watch's capabilities and therefore chose to limit the type of new applications initially, but this hindered the watch applications' performance. Regardless, some developers took the technology constraints as a design challenge. They also felt that these restrictions forced them to create something completely new for this platform, instead of adapting old mobile phone applications and workflow. [33]

## **watchOS 2**

The following version watchOS 2 was released in quick succession in September 2015 only 5 months after the initial launch of the Apple Watch. This time the watchOS 2 released much more of the watch's capabilities to the developers with the support for native applications. This meant that developers could start making applications that were processed on the watch without the delay of sending data back and forth to the phone. The native applications are given access to most of the Apple Watch's features and sensors, so this has given developers much more to work with. In late April 2016 Apple announced that as of 1 June 2016 only native Apple Watch applications would be accepted to the App Store, signaling the end to first generation watchOS applications [34;35].

watchOS 2 also allowed third-party complications, referring to interactive icons or widgets that can be attached to the clock screen of the watch. These icons can be used to relay quick and short information to the user, for instance if an application requires user action or a new message has been received. In March 2016 the watchOS 2.2 update was release. The update brought mainly improvements to existing features and the possibility to link multiple watches to one iPhone. [36]

## **Future Apple Watch 2 and watchOS 3**

The market has learned to expect Apple to release a new model in each device category every year. With the first Apple Watch having been launched in April 2015, in the first quarter of 2016 there is already great expectation of an upcoming announcement of the second generation Apple Watch. At the moment of this writing no official information of Apple Watch 2 has been made available, but it is very likely that they would try to release it already in 2016, because the first Apple Watch was deemed by many critics to be technically so underwhelming that they would rather wait and see the next iteration [37].

### **3.3 Android Wear**

The other major player in the smartwatch market alongside Apple is Google's Android Wear operating system. An overwhelming majority of the current mobile phone market is shared between Apple and Google, and as smartwatches are usually tethered to a mobile phone, it is logical for them to also dominate the smartwatch market. Unlike the Apple Watch, which can only be use with an Apple iPhone, Android Wear supports both Android and iPhone devices. Support for iPhones is not complete though, the main limitation being support for third-party applications. [25]

Smartwatch devices running Android were available from June 2014, well before the Apple Watch, and, as Android is available to a wide range of manufacturers, new devices are being released constantly. Because of this fragmentation is a much larger issue on the Android platform than on iOS. Android Wear devices come in many different sizes and even shapes, as some devices feature a circular display to better mimic traditional watch aesthetics. Some traditional wristwatch manufacturers have also released smartwatch devices that blur the line between these two categories. Therefore Android Wear developers have to consider many more device setups and design aspects than they would with iOS devices.

Having more manufacturers delivering their own devices to the market also promotes new ideas for the platform. Smartwatches are still young, and while the technology is not ready to fulfil all of the requirements the public has had for devices, the ways in which the actual devices can be utilized is also still taking shape. Android Wear is a

popular platform for application development, but so far games have not been a priority for the developers or the publisher. [38;39]

## 4 Test Platforms in Mobile Application Development

In this chapter the platforms utilized for the testing of mobile applications are discussed. The main focus is on device testing and the software simulations of actual devices, but other platform options are summarized in the overview as well.

### 4.1 Overview of Platforms

The main focus of this thesis is to examine the differences between testing on a mobile device and testing on a simulator, a software application that aims to represent the final outcome on a desktop computer screen. The two platforms will be discussed in more detail under their own headings below.

Both simulators and real devices can be used with test automation in mobile phone and smartwatch projects. The proprietary iOS and Android development tools both have in-built automation tools for unit and user interface testing. Test automation for other test levels can be accomplished with third-party tools. Most third-party automation tools need an external framework to be able to send actions to either the simulator or device. Once the device has been correctly configured for testing, it can be used for automated tests the same as the simulator.

A popular framework for running test automation scripts on a real device is Appium. Appium is an open-source framework and it can be used with both iOS and Android applications, including Android Wear and watchOS smartwatch applications. Appium is very versatile as it supports multiple different programming languages and development tools. For game development especially, the automation can be made simpler by utilizing also a tool like SikuliX.

SikuliX will enable the test engineer to use graphical references of objects, which overcomes the problem of getting access to the object IDs, also known as xpaths, from the device. Commonly in test automation, the automation script must have a reference to an object in order to interact with it. SikuliX uses screenshots as references and tries to find them from the screen and perform actions such as clicks when they are found. Through the automation framework the click on the screen are translated into taps on

the device. It is not exactly the same as real finger presses, but the computing is done on the device so the result is more realistic than on a simulator.

Another platform to consider is the third-party tools. There are also many third-party application development platforms utilized in game development, as well as other software development. Unity is currently the most popular of the available tools for game development, especially for mobile platforms. The strength of a third-party tool like this is the ability to develop a game application that can be deployed to multiple different platforms. Developers do not need to rebuild the application from scratch for each different platform with multiple different programming languages. When developing a game for iOS or watchOS with Unity, it is not possible to use the Xcode simulator to test the application. Testing is relying on how the application behaves inside Unity and builds that are deployed to a real device. [40;41]

#### 4.2 Relative Comparison of Simulators and Emulators

Because mobile applications are developed on desktop computers and not mobile devices themselves, testing these applications requires either an external mobile device to run the application on or software on the production computer that mimics the form factor and behavior of the desired device. Software applications used to run the development version of a mobile application on the production computer can be categorized as simulators and emulators. The difference lies in how thoroughly they mimic the desired platform. Simulators, as they are recognized in the context of mobile application development, run the source code of the mobile application on the production machine. The simulator will then present the application in the desired form factor mimicking a specific type of mobile device. The processing for the application is done by the development computer.

An emulator, in addition to giving a visual sample of the developed mobile application as the simulator does, will also mimic the hardware of the desired mobile device. Processing for the application will be done according to the specifications of the selected test device. Emulators will therefore give a better representation of how the application will function when installed on the physical device. On the other hand they are usually also heavier and slower to setup and start, so the time needed for executing tests will be longer. Therefore they both have their merits depending on the task at hand. [17,50]

This distinction between simulators and emulators is important to consider when testing mobile applications. Unfortunately, depending on the software platform the developer may not be given a choice in the matter. The development platform for Apple's iOS and watchOS devices, Xcode, only offers a simulator for testing applications. Google's own software development kit, Android Studio, only offers the possibility to use emulators, as do most other Android development tools. For Android there are third-party simulators available, such as Robolectric, Andy or Bluestacks, but for iOS and watchOS development there are no emulators currently available. For Android development these simulators can provide benefit in daily testing activities like unit tests by offering much simpler and faster setup and usage [42].

### 4.3 Testing on Physical Devices

Real devices can also be used for testing applications in the development phase. It is advisable to test the application with the devices it is intended to support before publishing the application, because the simulators might not accurately simulate each device's attributes. All of the different device features, most importantly the physical features, cannot be recreated in a simulation. The differences in features between the simulator and the device explained in detail in Chapter 5.

The device-based testing is carried out with real devices, any kind of device analogs or prototype devices are not supported. In the Apple iOS development suite, the devices used for testing have to be registered as developer devices. Similar processes for testing devices exist on other platforms as well. When testing on an actual Apple Watch device, it must be connected to an actual iPhone. With watchOS 2.2 it is now possible to link more than one Apple Watch to one iPhone [36]. This can be helpful for testers, as both Apple Watch models can be tested with one phone. There are more phone models than watch models however, and their testing must not be neglected either.

One way to increase the device coverage of the testing tasks, or gain access to devices unavailable devices, is to utilize external testing service. Many companies offer cloud solutions where developers can upload their applications either for automatic test cases to be run on them on multiple different devices, or for testers to execute manual tests on them. The benefit is having great amounts of resources readily available, but the downside, at least with manual testing, is usually not having an established

relationship with the tester and having only limited possibility to assist or guide them in their testing tasks. This type of external manual testing is mainly employed in the beta testing phase. Many companies and communities are set up exclusively around the beta testing of games.

## **5 Principles of Test Platform Prioritization**

The theory of test platform prioritization in relation to mobile application development is presented and discussed in this chapter. The limitations of simulators as testing platforms play a key role in the prioritization but the preservation of resources is also a consideration. Guidelines and tools for utilizing these principles are also introduced.

### **5.1 Limitations of Simulators**

In their article “Mobile Application Testing: A Tutorial” [43] Gao et al report the findings of their investigation of the utilization of different testing techniques for mobile application testing. Their findings are referenced in Table 1 below.

Table 1. Mobile Application Testing Approaches [43]

<b>Testing</b>		<b>Emulation-based testing</b>	<b>Device-based testing</b>
<b>Functionality and behavior</b>	<i>Function features</i>	Single emulator or simulator-based	Device-based single client
	<i>Mobile user operations</i>	Yes	Yes
	<i>Mobile gestures</i>	Limited	Yes
<b>Quality of service</b>	<i>Load testing</i>	Limited scale	Limited scale
	<i>Performance testing</i>	Function-based	Single client
	<i>Reliability/ availability</i>	Single client	Single client
	<i>Scalability</i>	No	No
<b>Interoperability</b>	<i>Crosses devices</i>	No	No
	<i>Crosses platforms</i>	No	Yes
	<i>Crosses browsers</i>	Yes	Yes
	<i>Crosses networks</i>	Limited	Limited
<b>Usability and internationalization</b>	<i>Internationalization of mobile user operation</i>	High cost and manual	High cost and manual
<b>Security and privacy</b>	<i>User security and privacy</i>	Limited	Limited
	<i>Communication security</i>	No	Yes
	<i>Transaction security</i>	Yes	Yes
	<i>Session security</i>	Yes	Yes
	<i>Server security</i>	Limited	Yes
<b>Mobility</b>	<i>Location-based function and behaviors</i>	Based on simulated location	Preconfigured location
	<i>Location-based user data and profile</i>	Simulated user profile and data	Single user profile and data
<b>Compatibility and connectivity</b>	<i>Browser compatability</i>	Single mobile browser	Single mobile browser
	<i>Network connectivity</i>	No	Singled network connectivity
	<i>Platform compatability</i>	Single platform	Any
<b>Multitenancy</b>	<i>Tenant-based functions and behaviors</i>	Yes	Yes
	<i>Tenant-based QoS</i>	Limited scale	Limited scale
	<i>Tenant-based interfaces</i>	Yes	Yes

The table lists the applicability of different testing functions for testing on an emulator and a real device. The platforms appear very similar in this comparison, with the real device separating mainly on network and security functionality.

The most important differences between the platforms in terms of testing can be categorized as performance differences, physical differences and functional differences.

### 5.1.1 Performance Differences

The biggest differences between the two testing platforms come with the performance aspects. Modern mobile phones especially are very capable personal computers, but processing power, memory and storage space all are vastly different on a desktop computer [44]. As explained in chapter 4, an emulator tries to emulate the mobile devices performance so as to give a better representation of how the application would behave on a real device. As no emulators for Apple devices are available, the performance of and iPhone or Apple Watch application in a simulator will not give reliable evidence of the end result.

The performance difference does not prevent comparable results being obtained from functional testing tasks. When testing internal logic, correctness of calculations or graphical objects, the computational proficiency does not affect the test outcome. Non-functional requirements, such as performance or reliability, can be covered by non-functional test in a separate testing phase. This does not mean the test cannot be executed in simultaneously or in parallel, the intentions of each test should always be considered and understood. It is also important to consider that when the mobile application is built for the simulator it is compiled and run in x86 architecture. The mobile device or smartwatch uses ARM architecture, so from that standpoint the tests on these two platforms are executed against different code, as the application is compiled separately for both of them [44].

### 5.1.2 Physical Differences

The physical form factor is an obvious difference between simulators and the real device. The simulator mimics the intended device's resolution on the computer screen, to give the tester an idea of how the application would appear on the mobile device. Using a simulation has a grave effect on the usability testing especially when it comes to smartwatches. A simulation on the large computer screen is a very different experience from having a tiny screen on the user's wrist.

There are also technical differences in the two methods of viewing the application. The pixel density of the screen on an Apple Watch is 330 pixels per inch (PPI) for the 38 millimeter model or 333 PPI for the 42 millimeter model. The Apple desktop and laptop computers, which are the most common devices for application development, have a

pixel density between 218 and 226 PPI in the modern high pixel density Retina line or between 110 and 135 PPI in the older models. This means that even though the resolution of the application is the same on all test devices, the size of the image output is different on all of them [45]. The simulator takes these factors in to account to some extent and maps the image pixels to different sizes depending on whether the destination device and the production device have Retina displays [46]. This can accentuate the size difference of the output on different test setups.

In addition to the pixel density the way in which the individual colors are arranged to produce the pixels also varies between screen types. Review and analysis of the visual aspects of a smartwatch or mobile phone application is therefore completely accurate only when done on the actual device. The two different sizes of Apple Watch models also have different screen resolutions, requiring different layout of the application. The graphics resources can either be made specific to each size, or one resource can be scaled to the other size as well. The quality is not affected greatly due to the scaling, but it can provide less predictable results than having specific resources, especially when there are more complicated scenes with multiple different graphic elements.

User interaction with mobile devices happens by finger taps and gestures, physical buttons, internal sensors and possible even voice commands. Other than voice commands, none of these interactions can be realistically simulated with simulators or emulators. Finger taps are done with mouse clicks, multi-finger gestures are not possible to simulate all, physical buttons are operated either by mouse clicks or keyboard shortcuts and sensors, such as accelerometers, are mimicked by sending the application fake data. The Apple Watch introduced “Force Touch” and “Digital Crown”, the former being a more forceful finger press and the latter a physical controller that can be turned or pushed. The Xcode simulator has the functionality to mimic these, Force Touch being activated by a keyboard shortcut and the Digital Crown being operated with the mouse wheel, but at least from the usability testing standpoint the experience is not the same. Technically the application will not care from which type of controller the input data is received, as long as the input is correct, but it is plausible for performance issues for example to arise from the rate of the user’s input being different between platforms.

### 5.1.3 Functional Limitations

The simulator does not provide all of the features that are available in the actual device. Features not available in the iOS simulator include push notification, App Store links, in-app purchases, media player and external accessories. The Camera application is also not available in the simulator, so even if the computer running the simulator has a camera, it is not possible to test features that use the standard Camera application. [47]

The simulator however does have features for inflicting scenarios that could be hard to replicate in an actual device in a testing setting. The tester can for instance slow down the animations to mimic performance problems, set the device location to simulate a drive around the Cupertino area where the Apple headquarters is located to feed data to the application or show different types of color coding in graphical elements to recognize possible issues. [47]

Even though it is possible to simulate the location of the device, it is not the same as moving a physical device. Especially network issues regularly come up when moving to from one place to another, closer or farther away from a cell tower, out-of-reach of a wireless network or the phone that is connected to the smartwatch. Testing how the application behaves when these types of interruptions happen is important and should not be neglected. [48,58]

## 5.2 Efficiency and Preserving Resources

The main reasons why all tests are not carried out on real devices and why simulators are needed are the amount of time needed to install and launch a build on a device and the challenge of having every type of supported device readily on hand. The game developers at Sneaky Crab tested application installation times to an Apple Watch device with different connections [49]. Their findings are shown in Table 2 below.

Table 2. Installation times to Apple Watch [49]

Time	Wireless	Apple Watch App
4:23	Bluetooth	Background
3:37	Bluetooth	Background
2:39	Bluetooth	Background
2:00	Bluetooth	Background
1:26	Bluetooth	Foreground
1:12	Bluetooth	Foreground
1:06	Wi-Fi	Background
1:01	Wi-Fi	Foreground
1:00	Wi-Fi	Foreground
1:00	Wi-Fi	Background

The table shows great variance in installation times, reaching from over four minutes to just around one minute. Bluetooth is the default way of communication between the watch and the phone with the Wi-Fi connection only used when the Bluetooth is turned off from either device. Utilizing the Wi-Fi connection provided great benefits to the resources, but even more could be gained from using the simulator where applicable. Especially in the beginning phases of development, where new builds are more prevalent, selecting the testing platforms carefully can save a lot of time. In addition to losing time, waiting four minutes for the application to install can lead to testers losing focus by forcing unnecessary interruptions to the workflow.

### 5.3 Utilizing Test Platform Prioritization

The following presents guidelines to base the prioritization of platforms on in different testing stages, as well as tools to aid in the planning of the prioritization considering different levels of test plans.

#### 5.3.1 Practical Guidelines

Based on the technical qualities outlined in the previous chapters, universal recommendations can be made for test platform selection in different phases of the game development life cycle. For unit testing which is conducted early on in the development and on a more volatile product, significant time benefits could be gained

from prioritizing the simulation-based testing. The unit testing task usually work well in the simulator, as the main focus of the tests is verifying the internal functionalities in the product code. There are some features, as outlined in the previous chapter, which cannot be tested on a certain platform. These particular tests should be identified as early as possible and planned accordingly.

Where more hands-on testing is to be conducted, and the usability and fun-factor aspects of the application are to be evaluated, such as the functional testing phases, best results are gained with using a real device. At this point in the development process there should not be so many new builds happening as to cause unreasonable delays due to installation times. Access to testing devices could be limited though, and for functional tests it is feasible to also use the simulator. The tests to be executed could be divided between the platforms based on the priorities of each test. The planning of test platform prioritization is covered in the next chapter.

Acceptance tests by definition should be conducted on the targeted platform by either the end-users or the stakeholders. Beta testing or playtesting would also serve little purpose to conduct on a simulator, as the usability aspects would differ greatly from the intended platform. If a higher volume of feedback is desired and access to devices is not abundant, this downside can be taken into account with the acknowledgement that the results will not be optimal.

### 5.3.2 Test Planning

Utilizing test platform prioritization means making a conscious choice of which testing platform to use in each stage of the testing process. In order for this choice to not be arbitrary, it needs to be based on thoughtful analysis of the applications and its features in terms of testing efforts and adequate experience of the utilized testing platforms. The test platforms can be considered for new or changing functionality in the test planning phase or it can be planned separately. Table 3 below shows what the result of a functionality level test platform prioritization might look like.

Table 3. An example of a test platform prioritization table on a functionality level

		Which platform to utilize in this testing level, simulator or device?			
Application	Functionality	Unit test	Integration test	System test	Acceptance test
"Game A"	Menu	Sim	Sim	Sim	Dev
	Character Creation	Sim	Sim	Dev	Dev
	Map Screen	Sim	Sim	Dev	Dev
	Cut Scenes	Sim	Sim	Sim	Sim
"Game B"	Fighting Gameplay	Sim	Sim&Dev	Sim&Dev	Dev
	Driving Gameplay	Sim	Sim&Dev	Sim&Dev	Dev
	Multiplayer Gameplay	Dev	Dev	Dev	Dev

In this example the test platform has been considered based on the requirements of individual functionalities on each test level. For "Game A" the functionalities to be tested are more graphical in nature. For those functionalities that contain more user interaction, device testing is emphasized. When the scope is as broad as this, the evaluation of test platform requirements can be based more on the complexity and apparent risk of the functionality if the technical features of the functionality are too mixed to serve as a motive. Planning testing activities, and possibly test platform prioritization, based on project risks is discussed more in the next chapter on **Risk-Based Testing**.

For "Game B" in this example the functionalities are more complicated and interactive. Device testing is emphasized even more, and for multiplayer functionalities all of the testing needs to be carried out on the device. This example is purely fictional and, depending on the technology used, even multiplayer functionalities could in some cases be tested on a simulator. This type of a test platform plan should in most cases be rather easy to create based on a good understanding of the functionalities under

test. A more detailed plan could be created on a test case level. An example of this is shown in Table 4.

Table 4. An example of a test platform prioritization table on a test case level

"Game A"			Test Platform	
Product Area	Test Level	Test Case	Simulator	Device
Character Creation	Unit Test	CC.UT.01	X	
		CC.UT.02	X	
		CC.UT.03	X	
	Integration Test	CC.IT.01	X	
		CC.IT.02	X	
		CC.IT.03		X
	System Test	CC.ST.01		X
		CC.ST.02		X
		CC.ST.03		X
		CC.ST.04	X	X
	Acceptance Test	CC.AT.01		X
		CC.AT.02		X

Test platform planning on this level requires more work but also a higher level of detail overall in the test planning. When writing the test cases for the development project the test platforms can be considered as a variable for each case. Another option is to plan the prioritization separately, especially if working with existing test cases or if more information on the technical implementation of the features is expected.

A plan for the execution of the test cases should also take into consideration the testing platforms. If 10 tests in a phase are to be executed with a real device and 25 on a simulator, it would best benefit the workflow if these cases were executed in order of platform. Setting up the platform in the middle of testing can cause loss of time and unnecessarily interrupt the testing. It can also be easier to spot defects when repeatedly executing tests in one platform, so that there are less changing variables.

### 5.3.3 Risk-Based Testing

One way to introduce test platform prioritization is through the utilization of risk-based testing. Risk-based testing is a methodology to prioritize test execution using in-depth analysis of risk in each product area. Risks are evaluated based on a set of variables with accumulate to give the final risk severity grade. The risk grade determines the approach and depth of the testing of this product area. An example of a risk evaluation using the same features as in Table 3 is shown in Table 5 below.

Table 5. An example of a risk evaluation in risk-based test design [50]

"Game A"		Effect		Probability	
Product Area	Business criticality	Visibility	Complexity	Change Frequency	Risk
<i>Weight</i>	3	10	3	3	
Menu	2	1	1	1	$6+10+3+3=22$
Character Creation	3	2	3	3	$9+20+9+9=47$
Map Screen	2	1	3	3	$6+10+9+9=34$
Cut Scenes	1	3	2	2	$3+30+6+6=45$

In this example each product feature is assigned 4 variables on a scale of 1-3:

- Business Criticality
  - How important is this feature for the business?
- Visibility
  - How likely are the users to notice errors in this feature?
- Complexity
  - How complex the structure and operation of this feature?
- Change Frequency
  - How often are changes implemented to this feature?

These variables are assigned a weight based on what is deemed the most important or damaging. In the example in Figure 7 the visibility of the feature errors is weighted more heavily than other aspects. Each variable is then multiplied by the weight factor and then summed up to produce the risk level of the feature. To further assist in the

test strategy, common levels of test coverage can be predetermined for the risk levels, as illustrated in Table 6 [17:135].

Table 6. An example of test formality recommendations based on risk level

<b>Risk Level</b>	<b>Recommended Test Phases</b>
High <b>45-57</b>	Unit, integration, system, acceptance tests
Medium <b>32-44</b>	Unit tests, user acceptance tests
Low <b>19-31</b>	Only unit tests

The risk-based testing strategy considers a different testing approach for each level of risk. This testing approach can be planned to include the test platform prioritization. The most severe risk would be tested with most platforms, while the least severe could be tested on simulator alone. Due to the limitations of the platforms the technical aspects of each function should be considered as well when choosing the testing platform, but this could provide a general basis for the prioritization. Table 7 shows another example where test platforms have been considered as well.

Table 7. An example of test formality recommendations factoring in test platforms

<b>Risk Level</b>	<b>Recommended Test Phases</b>
High <b>45-57</b>	Unit, integration tests on simulator System, acceptance tests on device
Medium <b>32-44</b>	Unit tests on simulator, acceptance tests on simulator
Low <b>19-31</b>	Only unit tests on simulator

Risk-based test prioritization assures that the testing tasks are scaled accordingly which should lead to increased efficiency of testing efforts. Setting up the risk assessment and the following risk-based testing plan does require a substantial administrative effort before-hand, but its advantages increase along with the scale of the project. Risk-based testing has been utilized in game development projects and because of the inherent need to heavily prioritize the testing tasks in game testing, it has proven a suitable technique for this particular field. [14, 51]

## 6 Investigation

This chapter presents the case study and investigation on the practical application of the theory of test platform prioritization. The testing setup is explained and the results of the investigation are analysed and compared to the theoretical suggestions.

### 6.1 Concept and Setup

The basic principles for test platform prioritization in regards to mobile game development were explained in the previous chapter. In order to verify the efficacy of these recommendations, a practical investigation on the testing platforms for a watchOS game project needed to be conducted. This investigation was carried out by testing the games released by the client of this thesis.

The client currently develops their games solely for the Apple Watch, and their two released games were the target of this trial. The first game, Runeblade, was published in April 2015 at the same time as the first Apple Watch. The way Everywear Games develops their games includes weekly minor maintenance updates and monthly larger content updates throughout the game's life time. This meant that, having been on the market for nearly a year, the game had grown in features quite a lot from its first inception.

The second game, Time Unit, was released in December 2015, near Christmas time. The development method was similar to the first game, and this game will also continue to be updated with new features regularly. Both games having been released to the customers meant that they had been tested to great extent already by both the developers and the end-users. Time Unit being in an earlier stage of its life cycle post-release meant that the product was not yet as complete functionally as Runeblade, and was expected to contain more errors.

The testing tasks executed for this trial were by design functional system tests of new features as well as functional regression tests of the old features. The testing was conducted using a combination of prepared test cases and exploratory testing practices. Both of the applications had, in addition to the Apple Watch application, a connected iOS application that runs on the iPhone. For Runeblade the companion application had a multitude of functionalities that were important for the gameplay and

the gaming experience, whereas Time Unit's companion in its current state was mainly informational. The iOS applications were also tested during the trial.

The testing setup consisted of a MacBook laptop running Xcode version 7.2.1 with the Xcode simulator version 9.2. For device testing the application was installed on an iPhone 6 and a 38mm Apple Watch with watchOS2. The testing was carried out on the client's premises using their hardware and release builds of the applications with debug controls activated. Using the debug controls was important for the test execution because they allowed the tester to trigger game events and simulate the play time so that it was not necessary to play through the whole game multiple times to test all of the functionality.

## 6.2 Findings

A total of 9 defects were reported based on the findings of the investigation, 2 in Runeblade and 7 in Time Unit. The distribution of these defects supports the initial expectation that defects would be less prevalent in the much earlier released Runeblade. The defects fell into 3 separate categories, which are shown in Figure 5 below.

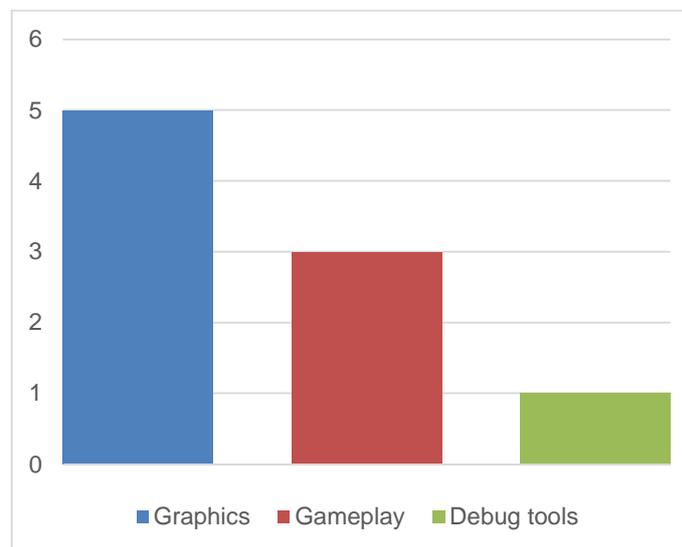


Figure 5. Categorization of Found Defects

Graphics defects, which were most common, were errors in the graphical elements of the game. These defects generally do not disrupt the gameplay and therefore their

severity depends on how apparent they are to the users. An example of a found graphical defect was that on one screen of Time Unit the presented image did not fit completely on the smaller Apple Watch screen and the user was not given the opportunity to scroll the screen to reveal the rest of the image. One graphic defect was also found in the iOS application for Time Unit, while all other defects were found in the watchOS applications.

The gameplay defects are errors in the gameplay mechanics of the application. They tend to disrupt the gaming experience in some way, so they are likely more severe than graphical defects. Depending on the type of game and the function which it affects, these kind of defects can result to loss of progress or even financial losses to the user. The gameplay defect discovered in Runeblade caused the user to lose collectable in-game currency in an important point quite early on in the game. While it was not a financial harm and all of the new users that would be affected by this would not necessarily notice the damage, it had the possibility to influence the playing experience in a crucial point where the user was learning how the game works. What makes this particular defect even more interesting was that it was a regression defect, a functionality that had been working previously but rendered non-functioning by a recent update.

The single debug tool defect related to the internal testing tools that developers add to a version of the application that is still under development. These tools can be used to afflict actions or alter variables to grant the tester the ability to simulate gameplay events or access certain parts of the application without having to play through the entire game. Since these tools will be removed or hidden from the final application, errors found in this area will only affect the developers themselves. The full defect report gathered from this investigation can be found as Appendix 1 of this thesis.

Regardless of which platform was used when initially uncovering each defect, all of them were repeatable on the other platform as well. The issues relating to the scaling of graphics on the smaller Apple Watch were naturally not found on the larger Apple Watch but regardless of whether testing on a simulator or the device, no difference in the apparent quality was found.

Based on this evidence the investigation did not reveal any apparent differences between the testing platforms when conducting functional testing. Deploying the

application build from Xcode to device did take a significantly longer time than deploying to simulator. Deployment to device took between three and four minutes, whereas for simulator the process took under one minute. After the deployment was done, the application startup times on both platforms were similar, so this would have a large impact on testing only when changes would need to be made to the application during testing.

The usability differences between devices were evident during the investigation. Even though the Xcode Apple Watch simulator provides tools for simulating most of the functions of the Apple Watch, additional effort was needed to apply these. The actual Apple Watch device shuts down the screen and makes the application inactive if the user does not interact with the device for a short time or if the user lowers the device below the perceived line of sight. The simulator has a toggle for "Sleep" which causes similar reactions in the application but this behavior which is commonplace on the device had to be consciously inflicted when testing in the simulator.

There were considerably more platform issues when testing on the simulator. Multiple times during deployment the Xcode produced the following error message:

*Error Launching 'xxx WatchKit Extension'. Installation error. Check the iPhone console for more details.*

No explanation for this error was found. It usually appeared when first trying to build the application, and not on the following attempts. The simulator also got stuck when the application was left running for an extended period of this uninterrupted, rendering the application unresponsive. These two aspects favored utilizing the device for all testing tasks.

## 7 Conclusion

The primary goal of this thesis was to formulate and document a set of general guidelines for prioritizing test platforms in a mobile game development project. Chapter 5 fulfils this goal, even though the application of the theory is singular at the moment. The secondary goal of this thesis was to examine and document the practical application of the theory of test platform prioritization for functional testing in a smartwatch application development project. This goal was fulfilled by the investigation presented in Chapter 6. The findings of the investigation also support and elaborate the theory presented.

Test platform prioritization as presented in this thesis has practical applications but it is not viable for every project. It can also be utilized in projects outside of the gaming field. The investigation showed no difference in testing results between the Apple Watch device and the Xcode Apple Watch simulator in functional system and regression tests. There were some functionalities of the application that could not be tested on the simulator so testing without the device would have left some gaps in the test coverage. It would be a more feasible strategy to conduct unit testing tasks, when possible, with only the simulator. Installing the application builds to the device takes considerably longer and, because unit tests are conducted in a phase where more changes are still made to the project, this would lead to significant benefits to resources and work flow.

The prioritization of testing platforms can be carried out on an ad hoc basis or it can be planned ahead utilizing tools such as the ones presented in this thesis. More tools can be created or discovered in the future to cover a wider range of scenarios and development frameworks. Test platform prioritization for unit testing would be an interesting topic for future study since unit tests are very different in nature to functional system tests and with unit tests there is a greater possibility of affecting the time usage through prioritization.

Test automation would be another field where test platform prioritization could yield interesting results. Running automated tests on an actual device would be significantly more challenging compared to a simulator. There are frameworks for controlling mobile device functions through the desktop computer interface, such as Appium, which can be used for automation, but maintaining the test sets and the devices in working order

for executing automated scheduled regression or smoke tests would certainly present complicated issues. Before setting up this kind of a system it would be important to first discover if running the automated tests on a physical device would produce greater results to justify the additional effort.

## References

- 1 Rubin R. watchOS: The Apple Watch Ecosystem Takes Shape [online]. App Annie; 10 December 2015.  
URL: <http://blog.appannie.com/watchos-apple-watch-ecosystem-takes-shape/>.  
Accessed 19 April 2016.
- 2 Lappalainen E. Pelien valtakunta. Jyväskylä, Finland: Atena Kustannus Oy; 2015.
- 3 Hayward A. The first 10 Apple Watch games you should play [online]. MacWorld, Games. San Francisco, California: IDG Consumer & SMB; January 2016.  
URL: <http://www.macworld.com/article/2920127/software-games/the-first-10-apple-wath-games-you-should-play.html>.  
Accessed 9 January 2016.
- 4 Kastrenakes J. Thousands of Apple Watch apps have been made without using the device — will they be any good? [online]. The Verge. Washington, DC: Vox Media Inc; 23 April 2015.  
URL: <http://www.theverge.com/2015/4/23/8484133/apple-watch-apps-developer-interviews-launch-day>  
Accessed 25 April 2016.
- 5 PocketGamer.biz. App Store Metrics – App Prices [online]. Bath, England: Steel Media Ltd; 2016  
URL: <http://www.pocketgamer.biz/metrics/app-store/app-prices/>.  
Accessed 28 February 2016.
- 6 Sourcebits. Paid vs. Free Apps in the App Store vs. Google Play [online]. The Source. San Francisco, California: Sourcebits; 16 July 2014.  
URL: <http://sourcebits.com/app-development-design-blog/paid-vs-free-apps-app-store-vs-google-play/>.  
Accessed 28 February 2016.
- 7 HoneyTracks. Improving game monetization with A/B testing [online]. Munich, Germany: HoneyTrack GmbH; 2016  
URL: <https://honeytracks.com/improving-game-monetization-with-ab-testing/>.  
Accessed 9 January 2016.
- 8 Sifa R, Bauckhage C, Drachen A. The Playtime Principle: Large-scale cross-games interest modeling. Dortmund, Germany: 2014 IEEE Conference on Computational Intelligence and Games; 26-29 August 2014.
- 9 Nozhnin D. Predicting Churn: Data-Mining Your Game [online]. Gamasutra. New York, New York: UBM Tech; 17 May 2012.  
URL: <http://gamasutra.com/view/feature/170472/>.  
Accessed 12 April 2016.
- 10 Hadiji F, Bauckhage C, Drachen A, Kersting K, Sifa R, Thureau C. Predicting player churn in the wild. Dortmund, Germany: 2014 IEEE Conference on Computational Intelligence and Games; 26-29 August 2014.

- 11 Schultz CP, Bryant RD. Game Testing: All-In-One. 2<sup>nd</sup> ed. Herndon, Virginia: Mercury Learning and Information; 2012.
- 12 Keith C. Agile Game Development with Scrum. Boston, Massachusetts: Pearson Education; 2010.
- 13 Thang J. The Tough Life of a Games Tester [online]. IGN. Chicago, Illinois: Ziff Davis LLC; 29 March 2012.  
URL: <http://www.ign.com/articles/2012/03/29/the-tough-life-of-a-games-tester>.  
Accessed 22 November 2015.
- 14 Cardwell P, Gilmartin M, Merritt D, Parkinson D, Vasquez M, Wibberley B [online video]. Back to the Future of QA. Panel Discussion at Game Developers Conference. San Francisco, California: Game Developers Conference; 19-21 March 2014.  
URL: <http://www.gdcvault.com/play/1020481/Back-to-The-Future-of>.  
Accessed 23 April 2016.
- 15 Royce WW. Managing the Development of Large Software Systems. IEEEWescon; 1970.
- 16 Scrum Academy. The Differences between Testing in Traditional and Agile Approaches [online]. International Agile Tester Foundation chapter 2.1. Scrum Association; 2016.
- 17 Hass AM. Guide to Advanced Software Testing. 2<sup>nd</sup> ed. Norwood, Massachusetts: Artech House; 2014.
- 18 Crooks CE, II. Mobile Device Game Development. Boston, Massachusetts: Charles River Media; 2004.
- 19 Farrell-Vinay P. Manage Software Testing. Boca Raton, Florida: Auerbach Publications; 2008.
- 20 Apple. What's New in iOS [online]. Cupertino, California: Apple Inc; 2015.  
URL: <https://www.apple.com/ios/whats-new/#compatibility>.  
Accessed 1 March 2016.
- 21 Whitney L. iOS 8 hits 85% adoption rate; Android Lollipop only at 18% [online]. CNET Mobile. San Francisco, California: CBS Interactive; 5 August 2015.  
URL: <http://www.cnet.com/news/ios-8-hits-85-adoption-rate-android-lollipop-only-at-18/>.  
Accessed 28 February 2016.
- 22 IDC Research, Inc. IDC Forecasts Worldwide Shipments of Wearables to Surpass 200 Million in 2019, Driven by Strong Smartwatch Growth [online]. Framingham, Massachusetts: IDC Research, Inc: 17 December 2015.  
URL: <http://www.idc.com/getdoc.jsp?containerId=prUS40846515>.  
Accessed 19 January 2016.
- 23 Charara S. Signs of Android Wear fragmentation: No Wi-Fi for some smartwatches [online]. London, England: Wareable Ltd; 22 April 2015.  
URL: <http://www.wareable.com/android-wear/>

[android-wear-fragmentation-smartwatch-wi-fi-update-1070](#).

Accessed 25 April 2016.

- 24 Walker-Morgan D. Tizen 2.0 SDK comes in "Magnolia" [online]. London, England: Heise Media UK Ltd; 19 February 2013.  
URL: <http://web.archive.org/web/20130609065612/http://www.h-online.com/open/news/item/Tizen-2-0-SDK-comes-in-Magnolia-1806013.html>.  
Archived from the original on 9 June 2013.  
Accessed 23 April 2016.
- 25 Charara S. Tizen v Android Wear: Which smartwatch OS is right for you? [online]. London, England: Wearable Ltd; 1 February 2016.  
URL: <http://www.wearable.com/smartwatches/tizen-os-vs-android-wear>.  
Accessed 23 April 2016.
- 26 Gil L. The best Samsung Gear S2 apps so far [online]. London, England: Wearable Ltd; 18 January 2016.  
URL: <http://www.wearable.com/samsung/best-samsung-gear-s2-apps-1804>.  
Accessed 23 April 2016.
- 27 Crecente B. Time killers: The strange history of wrist gaming [online]. Polygon. Washington, DC: Vox Media Inc; 7 December 2015.  
URL: <http://www.polygon.com/a/smartwatch-history-guide-evolution>.  
Accessed 26 April 2016.
- 28 Cheng J. The original iPod, 10 years later: a re-review [online]. Ars Technica. New York, New York: Condé Nast; 23 October 2011.  
URL: <http://arstechnica.com/apple/2011/10/2001-to-2011-ars-re-reviews-the-original-ipod/>.  
Accessed 19 January 2016.
- 29 Kelion L. Apple Watch unveiled alongside new larger iPhones [online]. BBC News, Technology. London, England: BBC; 9 September 2014.  
URL: <http://www.bbc.com/news/technology-29128083>.  
Accessed 7 January 2016.
- 30 Mayne M. Apple iWatch: Price, rumours, release date and leaks [online]. T3. Bath, England: Future PLC; 20 June 2014.  
URL: <http://www.t3.com/news/apple-iwatch-rumours-features-release-date>.  
Accessed 7 January 2016.
- 31 Richtel M, Chen BX. Tim Cook, Making Apple His Own [online]. The New York Times, Technology. New York, New York: The New York Times Company; 15 June 2014.  
URL: <http://www.nytimes.com/2014/06/15/technology/tim-cook-making-apple-his-own.html>.  
Accessed 7 January 2016.
- 32 Apple Developer. WatchKit [online]. Cupertino, California: Apple Inc; 2015.  
URL: <https://developer.apple.com/watchkit/>.  
Accessed 2 February 2016.

- 33 Dredge S. Game developers on Apple Watch: smartwatches are all about context [online]. The Guardian, Tech. London, England: Guardian News and Media Ltd; 24 April 2015.  
URL: <http://www.theguardian.com/technology/2015/apr/24/game-developers-apple-watch-smartwatches>.  
Accessed 19 January 2016.
- 34 Apple Developer. Upcoming Requirement for WatchOS Apps [online]. Cupertino, California: Apple Inc; 22 April 2016.  
URL: <https://developer.apple.com/news/?id=04222016a>.  
Accessed 28 April 2016.
- 35 Page C. Apple demands watchOS developers make all apps native by 1 June [online]. The Inquirer. London, England: Incisive Business Media Ltd; 25 April 2016.  
URL: <http://www.theinquirer.net/inquirer/news/2455871/apple-demands-watchos-developers-make-all-apps-native-by-1-june>.  
Accessed 28 April 2016.
- 36 MacRumors. WatchOS 2 [online]. Glen Allen, Virginia: MacRumors.com, LLC; 25 April 2016.  
URL: <http://www.macrumors.com/roundup/watchos-2/>.  
Accessed 28 April 2016.
- 37 Briden B. Apple Watch NOT "Compelling" Says Steve Wozniak [online]. Know Your Mobile – News. London, England: Dennis Publishing Ltd; 18 April 2016;  
URL: <http://www.knowyourmobile.com/wearable-technology/apple-watch/23532/apple-watch-not-compelling-says-steve-wozniak>.  
Accessed 20 April 2016.
- 38 Naziri J, Temple R. 40 best Android Wear smartwatch apps 2015 [online]. TechRadar. Bath, England: Future PLC; 22 December 2015.  
URL: <http://www.techradar.com/news/wearables/best-android-wear-smartwatch-apps-2015-1281065/5>.  
Accessed 25 April 2016.
- 39 Hindy J. 10 best Android Wear games [online]. Android Authority; 17 January 2016.  
URL: <http://www.androidauthority.com/best-android-wear-games-667988/>.  
Accessed 25 April 2016.
- 40 Unity Technologies. Unity iOS Basics [online]. Unity Documentation. San Francisco, California: Unity Technologies; 2016.  
URL: <http://docs.unity3d.com/Manual/iphone-basic.html>.  
Accessed 28 April 2016.
- 41 Deniozou T. Export from Unity to an iOS Device [online]. Gamasutra. New York, New York: UBM Tech; 26 August 2014.  
URL: [http://gamasutra.com/blogs/ThaleiaDeniozou/20140826/224186/Export\\_from\\_Unity\\_to\\_an\\_iOS\\_Device.php](http://gamasutra.com/blogs/ThaleiaDeniozou/20140826/224186/Export_from_Unity_to_an_iOS_Device.php).  
Accessed 20 April 2016.
- 42 Wharton J. Android Needs a Simulator, Not an Emulator [online]. jakewharton.com: 16 June 2014.

- URL: <http://jakewharton.com/android-needs-a-simulator/>.  
Accessed 13 November 2015.
- 43 Gao J, Bai X, Tsai WT, Uehara T. Mobile Application Testing: A Tutorial. *Computer*. 2014;47(2):46-55.
- 44 Roadfire Software. Will an iOS app run on a device the same way it does on the iOS simulator? [online]. Fishers, Indiana: Roadfire Software; April 2015.  
URL: <http://roadfiresoftware.com/2015/04/will-an-ios-app-run-on-a-device-the-same-way-it-does-on-the-ios-simulator/>.  
Accessed 7 March 2016.
- 45 Storey D. Understanding Pixel Density, Resolution and Retina Displays [online]. *thenewcode.com*; 2015.  
URL: <http://thenewcode.com/564/Understanding-Pixel-Density-Resolution-and-Retina-Displays>.  
Accessed 7 March 2016.
- 46 Apple Developer. Interacting with iOS and watchOS [online]. Simulator User Guide. Cupertino, California: Apple Inc; 21 March 2016.  
URL: [https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS\\_Simulator\\_Guide/InteractingwithiOSandwatchOS/InteractingwithiOSandwatchOS.html](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/InteractingwithiOSandwatchOS/InteractingwithiOSandwatchOS.html).  
Accessed 28 April 2016.
- 47 Apple Developer. Testing and Debugging in Simulator [online]. Simulator User Guide. Cupertino, California: Apple Inc; 21 March 2016.  
URL: [https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS\\_Simulator\\_Guide/TestingontheiOSSimulator/TestingontheiOSSimulator.html](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/TestingontheiOSSimulator/TestingontheiOSSimulator.html).  
Accessed 28 April 2016.
- 48 Knott D. Smartwatch App Testing. Victoria, Canada: Leanpub; 14 March 2016.
- 49 Sneaky Crab. Speeding up slow app install times when debugging on a real Apple Watch [online]. San Jose, California: Sneaky Crab Inc; 28 May 2015.  
URL: <http://www.sneakycrab.com/blog/2015/5/28/speeding-up-slow-install-times-when-debugging-on-a-real-apple-watch>.  
Accessed 28 April 2016.
- 50 Schaefer H. Risk Based Testing, Strategies for Prioritizing Tests against Deadlines [online]. *Methods and Tools*. 2005;13(4):18-36.
- 51 Miller LL. Considering Risk in Video Game QA [online]. Gamasutra. New York, New York: UBM Tech; 2014.  
URL: [http://www.gamasutra.com/blogs/LindsayLautersMiller/20140929/226492/Considering\\_Risk\\_in\\_Video\\_Game\\_QA.php](http://www.gamasutra.com/blogs/LindsayLautersMiller/20140929/226492/Considering_Risk_in_Video_Game_QA.php).  
Accessed 25 April 2016.

## Appendix 1: Defects Found in the Investigation

ID	Project	Name	Defect Type / Product Area	Description	Environment	Comments	Date
1	Rune Blade	<b>Debug +100 keys not working</b>	Debug menu	In the debug menu, the button for "+100 keys" does not increase the keys count. "+10 keys" works ok.	Device / Simulator		15.3.2016
2	Rune Blade	<b>Crystals not received in phone app after retire</b>	Gameplay	After beating Wrathful Doppelgänger to finish the first generation and reach level 100, the gathered crystals are not passed on to the phone app. Dialog screen after victory shows "you pass on these resources: [crystal icon] +0, TAP +105%, DPS +45%". Statistics on phone app show "Crystals From Guardians 12"	Device / Simulator	Regression	15.3.2016
3	Time Unit	<b>Graphics cut off at bottom of screen in Hero-menu on 38mm</b>	Graphics	When using 38mm Apple Watch or simulator, in the Hero-menu in the WatchKit app the Hero's feet and the bottom of the action button are outside the screen. Scrolling is not possible.	Device / Simulator (38mm)	Works ok on 42mm simulator	16.3.2016
4	Time Unit	<b>Previous screen flashes when returning from sleep</b>	Graphics / Life cycle	Navigate to any screen other than the map screen. Wait for watch screen to turn off Activate the screen by lifting the watch Game returns to Map Screen, but previous screen flashes quickly at the bottom of the screen	Device (38mm) / Simulator	The effect is more severe when returning from the watchface via Glances. ^This can also be replicated in simulator.	16.3.2016
5	Time Unit	<b>Resources change to another in the resource gathering minigame</b>	Graphics	When picking up a whole column of resources in the minigame, the top most icon that is dropped in the empty place can glitch and occasionally change to another resource icon after already in it's place.	Device (38mm) / Simulator	In simulator this happens consistently with complete columns, in device behavior is a little more inconsistent and occasionally might happen with incomplete rows as well	16.3.2016

ID	Project	Name	Defect Type / Product Area	Description	Environment	Comments	Date
6	Time Unit	<i>name of the defect</i> <b>Extra actions token shows wrong graphics after +4</b>	<i>where was it found</i> Graphics	<i>what happened</i> In a fight, use Zinger secondary ability to grant extra actions to Supernova, and then use Supernova secondary ability to boost next action on Zinger TWICE to give Zinger 4x boost.  Repeat the same process on the next round to give Zinger 8x boost. Then use Zinger's 8x boosted secondary ability to give 8 extra actions to Supernova. Supernova now has 9 moves, and the token icon shows up as a default (?) grey bubble, not yellow icon like normally.  Also, after using 1 move with Supernova, grey bubble changes from 9 to 7.	<i>found in environment</i> Device (38mm) / Simulator		<i>discovered</i> 16.3.2016
7	Time Unit	<b>Dr Blomstrom name tag on Heroes page pushed out of screen</b>	Graphics	After unlocking Dr Blomstrom, go to Heroes menu in phone app. On Dr Blomstrom's page, the description text is so big that it pushes the "name box" partially outside to content area.	iPhone 6 / Simulator	In simulator with 6s Plus the problem appears to a lesser degree (larger screen)	16.3.2016
8	Time Unit	<b>Wrong Heroes shown in phone when opening then out of sequence</b>	Gameplay	After having only unlocked the second hero, use Debug menu to toggle fifth hero Dr Blomstrom. In watch app the 1st, 2nd and 5th heroes show up in the menu.  Open Heroes menu in phone. Instead of Dr Blomstrom, the info card for Zinger is shown. The heroes appear to open in sequence on the phone, even if game action don't reflect this.	Simulator / Device	Possibly a non-issue, if game mechanics block this from happening naturally.	16.3.2016

ID	Project	Name	Defect Type / Product Area	Description	Environment	Comments	Date
<i>automatic</i> 9	Time Unit	<i>name of the defect</i> Superpower icon showing up after Reset	<i>where was it found</i> Gameplay	<i>what happened</i> Buy Power Punch Superpower for Supernova. Reset game using Debug options. Go into a fight with Supernova. Superpower icon is not visible. Toggle all remaining heroes using Debug options. Go in to a fight with Supernova + 2 others. Green check or superpower timer shows up next to Supernova, but does not function or change. If Supernova is not selected for the fight, Superpower icon shows next to the Hero that is in the middle even if it's not Supernova.	<i>found in environment</i> Simulator / Device	Since this happens after using Debug reset, the scenario is unlikely to come up in normal gameplay.	<i>discovered</i> 16.3.2016