1. Contents

[1: Introduction 2](#_Toc526966868)

[2: Fault Repairs 2](#_Toc526966869)

[3: Environmental Adaption 3](#_Toc526966870)

[4: Increasing functionality: 4](#_Toc526966871)

[5: Conclusion 5](#_Toc526966872)

[References 6](#_Toc526966873)

# 1: Introduction

After we already introduced our first mobile game to the market it wont take long for us to face new problems. Due technology and especially mobile games are short-lived these days, we can not take profit from our product with out maintaining it for a long time. Changing environmental conditions, occurring errors and growing user expectations for more and more content and functionality will force us to spend effort to maintaining our mobile game constantly. In this paper we will describe how exactly we want to do this containing how much time and effort we will spend in the different topics, which steps we plan to do and what goals and requirements we want to fulfill including the advantages of doing so and the resulting conditions for further steps and developments.

# 2: Fault Repairs

The first area we are looking at is fault repairs. We have two main systems that will be sending in faults from users. The second one is the user’s comments. Google play has a built-in system for users to leave comments and ratings. We will be watching these comments and looking for possible faults there. Our first line of bug detection is the “Report Bug” ability that we built into the application. Under certain conditions the user will be directed to a “Bug Report” form. The ways that a user could get there includes: the bottom default option of a “Try Catch” were the coding does not know what to do, unexpected shutting down of the program, the program caught in an endless loop, and a user’s option to report an issue. The bug report itself will send us several different pieces of information. These will include: the error code ( when one is generated ), the name of the map that is being played, the starting condition of the map, the current condition of the map, a record of major events in the game ( major events would be placing or destruction of buildings and a resource being depleted in a square, or growing to a square ), and the last X numbers moves the player has made ( X for example could be 25 ), and a comment left by the users describing the “Bug” / fault. With this information we should be able to recreate the conditions that caused the fault and identify its’ cause. This fault will be added to the list of changes that will be addresses in the next iteration of the software.

This is one of the reasons for, and advantages of, our choice of using an incremental method of development using agile test-driven development (TDD). The fault repairs after the application is released is essentially a continuation of our software development process. It adds things to do for the next iteration, namely repair this fault, in addition to the changes already planned like additions to the game and improving the existing code. Our initial estimate for fault repair maintenance costs is about 24%, or about ¼ of the total maintenance costs (Davidsen and Krogstie 2010). We will be able to refine our estimate for each application as time goes on. Fault repair and environmental adaption make up almost half the total maintenance. Therefore, being a 2 person company, these two maintenance responsibilities will be assigned to one person and the other task of functionally addition or modification will be the other persons maintenance responsibility.

# 3: Environmental Adaption

Environmental adaption is the next area of maintenance. With out project being a mobile application, a game to be precise, it is in no way a critical system. It is for entertainment purposes. At some point the number of players and the income it generates will make it obsolete and no longer in need of any maintenance. While the game is still generating good income there are two things that will cause the need for environmental adaption. These would be changes in the operating system and changes the in mobile devises that is it being played on. Mobile devises and their operating systems are evolving quickly. Firstly, lets consider operating systems. For the most part they are backward compatible, but there are times when a command becomes depredated, at which point the code will need rewriting. Another possibility is the inclusion of a new command that does the same task better, this would create the need for refactoring. Looking at changes in mobile devises, size and device resolution are always changing. This can cause display issues and will be a constant cause for adaption.

The other side of the display causing the need for adaption is new methods for input to the mobile device. This could create the need for major revisions of the game and create new ideas for games that could be created. Generally environmental adaption is estimated at 19% of maintenance costs or about 1/5 of the total (Davidsen and Krogstie 2010).

# 4: Increasing functionality

The biggest part of our system maintenance will be the developing and adding of new functionality. Round about 55% of software maintenance are in the field of developing new functionality. Our mobile game will be in a very simple state when released. Main functionality will be working but won’t be large-scaled. Therefore, we need to develop and validate new content fast to satisfy the users expectations. How we will develop the new functionality will be the same software development approach as we used for the main application, incremental with Agile methods including TDD. This was described in earlier papers and do not have to be mentioned in this paper again. We will test the new units we developed on their own and the interfaces between new and old, but we won’t do another system test because this would need to much time and we trust in our own skills and previous tests. As soon as we have a certain amount of functionality and possibilities for the users, we will announce and release our user level system which will allow all our users to start to compete. This step will give us the possibility to produce a chargeable update to generate more income. Due the users will be busy with each other the pressure on us as development team to release more functionality fast will be reduced. We can use this time to improve the game more and more and work on a bigger update with extended functionality like an artificial intelligence as enemy or a direct real time multiplayer mode which could be another chargeable update. So, the biggest challenge will be the start-up phase and the first updates to establish our game on the market. As soon as we think the game reached a peak we can start to work either on another mobile game and profit from our earlier reputation as mobile game developer or use the earned money and improved skills for a new project in another field.

# 5: Conclusion

We will split the different parts of software evolution between the 2 of us. Combining fault repair and environmental adaption together will be close to the amount of work in increasing functionality, so this is how we will split these tasks to start with. As the project proceeds we will adjust tasks as we find out our actual numbers. As we are maintaining our first release we will also be starting to create our next game and hopefully generate enough income to not only pay ourselves but to also higher additional staff. The decision to use an incremental method of software design combined with agile methodologies should help made the transition for the pre-release work and post-release maintenance go smoothly. It expands the feedback on our software and gives suggestions on where we can improve our game. The information we have built up in out TDD should also make the maintenance of our software easier than if we did not have those tests and results. The decisions on how we did our software development help us to continue with our software evolution.

# References

Sommerville, I. (2016). *Software engineering*. Harlow, England; New York: Addison-Wesley, 2016.

Davidsen, M.G., and Krogstie J. 2010 “A Longitudinal Study of Development and Maintenance.” *Information and software Technology* 52 (7): 707 - 719