Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Mikko Lahti

Game Testing in Finnish Game Companies

Master's Thesis

Espoo, November 30, 2014

Supervisor: Professor Perttu Hämäläinen, Aalto University

Thesis advisor(s): Perttu Hämäläinen, D.Sc. (Tech.)
Juha Itkonen, D.Sc. (Tech.)

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076
AALTO
www.aalto.fi

Abstract of master's thesis

Author Mikko Lahti

Title of thesis Game Testing in Finnish Game Companies

Degree programme Computer science and engineering

Thesis supervisor Professor Perttu Hämäläinen

Code of professorship T-111

Department Department of Media Technology

Thesis advisor(s) Perttu Hämäläinen, D.Sc. (Tech.)
Juha Itkonen, D.Sc. (Tech.)

Date	Number of pages	Language
30.11.2014	88	English

Games are very complex software systems with demanding requirements such as stability, good performance and compatibility. Extensive and diligent testing is required to satisfy these requirements and to deliver a successful game.

The purpose of this thesis was to research how Finnish game companies were currently handling game testing. This research problem was divided to several areas which included the objectives of testing, the testing process and the testing methods that were used.

To provide answers for these questions literature on software development, game development, software testing and game testing was analyzed and presented. Following this, seven Finnish gaming companies were interviewed and the results of these interviews were presented. These results include information such as the goals and objectives of testing, the evaluation and testing of the fun factor, game quality factors, organization of testing, required knowledge, skills and training in testing, planning and documentation of testing, testing in different phases of game development and testing sufficiency and criteria for ending testing.

As conclusions the game development process, the testing in different phases of development, the outsourcing of testing and the testing methods are discussed and some good practices are pointed out. Finally, the thesis presents further research topics.

Keywords game testing, testing, game quality assurance, quality assurance, game quality, game development, game production

Aalto-yliopisto
Perustieteiden
korkeakoulu

Aalto-yliopisto, PL 11000, 00076
AALTO
www.aalto.fi

Diplomityön tiivistelmä

Tekijä Mikko Lahti	
Työn nimi Pelitestaus suomalaisissa peliyrityksissä	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Valvoja Professori Perttu Hämäläinen	**Professuurikoodi** T-111
Laitos Mediatekniikan laitos	
Työn ohjaaja(t) Perttu Hämäläinen, Tekniikan tohtori Juha Itkonen, Tekniikan tohtori	

Päivämäärä	**Sivumäärä**	**Kieli**
30.11.2014	88	Englanti

Pelit ovat todella monimutkaisia ohjelmistojärjestelmiä, joilla on tiukkoja vaatimuksia kuten vakaus, hyvä suorituskyky ja yhteensopivuus. Jotta nämä vaatimukset saadaan toteutettua ja samalla tuotettua menestyvä peli, vaaditaan laajaa ja huolellista testausta.

Tämän diplomityön tarkoituksena oli tutkia, kuinka suomalaiset peliyritykset tekevät pelitestausta. Tutkimusongelma jaettiin useisiin alueisiin, jotka ovat testausten tavoitteet, testaus prosessi sekä testauksessa käytettävät menetelmät.

Vastausten etsiminen näihin kysymyksiin aloitettiin perehtymällä ja analysoimalla kirjallisuutta liittyen ohjelmistokehitykseen, pelikehitykseen, ohjelmistotestaukseen sekä pelitestaukseen. Tämän jälkeen haastateltiin seitsemää suomalaista peliyritystä. Haastattelujen tulokset esiteltiin työssä. Nämä tulokset sisältävät asioita kuten testauksen päämäärät ja tavoitteet, fun factorin arviointi ja testaus, pelin laatutekijät, testauksen organisointi, testauksessa tarvittavat tiedot, taidot ja koulutus, testauksen suunnittelu ja dokumentointi sekä testauksen riittävyys ja testauksen lopetuksen kriteerit.

Johtopäätöksinä keskustellaan pelikehitysprosessista, testauksesta prosessin eri vaiheissa, testauksen ulkoistuksesta ja testaus metodeista. Näistä poimitaan muutamia hyviä käytäntöjä. Lopuksi diplomityö esittelee mahdollisia jatkotutkimusaiheita.

Avainsanat pelitestaus, testaus, pelien laadunvarmistus, laadunvarmistus, pelin laatu, pelikehitys, pelituotanto

Acknowledgments

I want to thank Aalto University and the Department of Media Technology for providing me an opportunity to work on an interesting topic. I also wish to thank my parents for their support throughout my studies. A special thanks goes to all the companies taking part in the interviews, especially to the interviewed game testing professionals.

I would also like to thank Professor Perttu Hämäläinen and D.Sc. (Tech.) Juha Itkonen for their guidance and advice during the writing of this thesis.

Espoo, November, 30, 2014

Mikko Lahti

Abbreviations and Acronyms

AI – Artificial Intelligence, typically used for computer controlled game characters
DLC – Downloadable Content. Additional material for a game that has already been released.
GDD – Game Design Document
GPU – Graphics Processing Unit
Free-to-play – Games in which the base game is provided for free to the players. Revenue is typically gained from advertising or micro-transactions.
FPS – Frames per second OR First-person shooter
Fun factor (of a game) – The amount of fun a game is
Indie games – Independent video game. Games created by individuals or small teams without the financial support of a video game publisher.
Input lag – The delay between pressing a button on a controller and seeing the game react to the action.
MMOG – Massively Multiplayer Online Game
MMORPG – Massively Multiplayer Online Role-playing Game
Load testing – Used to determine the performance of a system under normal and anticipated usage patterns. This helps to identify the maximum operating capacity, possible bottlenecks and the elements that are causing poor performance.
Optimization (game graphics) – Optimization attempts to tweak the game in a way, so that it performs better (more fps). Such actions can be e.g., improving the underlying algorithms of the game, reducing model detail, reducing effect details.
IP – Intellectual Property
PR – Public Relations
Refactor – To modify software code without making changes in the functionality of the program.
NPC – Non-player character
RC – Release Candidate. When the game is close to being released, the game features are incrementally frozen, and new versions, release candidates of the game are built. From these, the final gold master version of the game is chosen.
Serious games – Games that do not aim for the entertainment market, but have different goals, e.g., to be educational.
TDD – Test-driven development
UI – User Interface
RTS – Real-time Strategy (game)
QA – Quality assurance
XBLA– Xbox Live Arcade. A web service provided by Microsoft, where users can digitally buy games of a smaller scale when compared to traditional retail games. The games also cost less than traditional retail games.

Table of Contents

Index of Tables

1 INTRODUCTION

The main motivation for the thesis emerged when the author attempted to find academic papers related to game testing, but was unable to find papers that felt relevant enough. With the lack of material, the author felt motivated to pursue this avenue of research to get more coverage of the topic. The main research problem can be summarized as:

How is game testing currently handled in gaming companies?

This research problem can be divided in to the following research questions:

- *What are the objectives of game testing?*
- *What process / processes are used?*
- *What methods are used?*
- *How is testing handled in Finnish gaming companies?*

The thesis is restricted to the games in the entertainment market, and does not include other forms of games, e.g., serious games (games used for an utilitarian purpose). The thesis attempts to answer these questions by first examining the current game testing literature. The literature review attempts to provide answers for questions one to three. The empirical part of the thesis consists of interviews of seven major Finnish gaming companies. From these interviews, short example cases are built, which are then compared for common and uncommon characteristics. The information acquired from the interviews is also compared to the information found from the literature. This provides additional information for questions one to three and provides answers for question four.

2 SOFTWARE DEVELOPMENT

Software development is the development of a software product, including all the activities from the first product related idea to the final version of the software product. The process itself can be split to the following activities: requirements engineering, design, implementation, testing & debugging, deployment and maintenance. These activities do not have to be performed sequentially, the organization and execution is defined by the selected software development process model. Pressman (2005, p.77-100) divides these process models to prescriptive process models, specialized process models and agile process models.

Requirements engineering attempts to provide a written understanding of the requirements for the software that is to be built. This includes understanding the business impact of the software, the needs of the customer and the way end-users will interact with the software. Some of the work products include: use cases, functions and feature lists, analysis models or a specification. These work products should be reviewed with the customer and end-users for correctness. It is very likely that the requirements will change throughout the project. (Pressman, 2005, p.174-175)

In design related activities, a representation or a model of the software is built, that provides detail about the software data structures, architecture, interfaces and other components that are necessary to implement the system. This representation of the software can then be assessed for quality and improved upon, before any code or tests have been generated. The software team assesses the design model by checking for errors, inconsistencies, omissions, possible alternative solutions and whether the model can be implemented within the constraints, schedule and cost that have been established for the project. The design model is the primary work product of the design phase and it establishes the quality of the software. (Pressman, 2005, p.258-259)

In implementation related activities, the team builds the components that were requested by the customer. The components can be built completely from scratch or be retrofitted from an already existing component library. The primary work product is the software product itself.

In testing and debugging the software is tested for errors made during the previous phases. Whereas testing finds the visible bugs in the software, debugging related actions attempt to find the code segments responsible for the malfunction, and to repair the offending segment. The focus of early testing is on single components or a small group of related components, applying tests to uncover errors in the data and processing logic of the modules. After the individual components have been tested, the testing moves on to integration level testing as the components are being integrated to work with each other. Finally, when the entire program has been integrated, higher level testing ensures the end-product satisfies the customer requirements. Testing is discussed in more detail in later chapters of this thesis. (Pressman, 2005, p.386-387)

Typically deployment is not an event that happens only once, but multiple times. In the deployment phase, the software is delivered to the customer who then evaluates the product and provides feedback based on these evaluations. This software can be either a complete entity or a partially completed increment. The deployment phase itself can be

split into three parts: delivery, support and feedback. In the delivery cycle the customer and end-users are provided with a software increment that provides additional functions and features. In the support cycle the customers are provided with the documentation and required human assistance for all functions and features that were introduced during all of the deployment cycles. In the feedback cycle the software team receives guidance that results in modifications to the functions, features and the approach taken for the possible next increment. (Pressman, 2005, p.56,148)

Software maintenance involves modifying the software product after the delivery in order to improve it, e.g., to fix faults, to improve performance or making any other improvements to the software. It can be split into four different activities: corrective maintenance, adaptive maintenance, enhancement maintenance or preventive maintenance. Corrective maintenance involves any activity in which software defects are removed from the software. Adaptive maintenance involves adapting existing systems to changes in their external environment. Enhancement maintenance involves making enhancements to the software product. Preventive maintenance involves re-engineering an application for future use. (Pressman, 2005, p.873-874) According to Pressman (2005, p.874), only about 20 percent of all maintenance work is spent fixing defects. The rest is either adaptive, enhancement or preventive maintenance.

2.1 Process models

Pressman (2005, p.77-100) divides the software process models to the following categories: prescriptive process models, specialized process models, and agile process models. The prescriptive process models are split further into linear process models, incremental process models and evolutionary process models.

The prescriptive process models introduce a set of framework activities that are organized into a process flow which may be linear, incremental or evolutionary. They prescribe a set of process elements: framework activities, software engineering actions, tasks, work products, quality assurance, change control mechanisms and the manner in which the process elements are interrelated to one another, referred to as a workflow. In the linear process models the requirements of a problem are reasonably well understood and the work flows from requirements gathering to deployment in a reasonably linear manner. The waterfall model is a good example of this model, in which the software development begins from the specification of requirements, progressing through planning, modeling, construction, deployment and ending in the maintenance phase all in a sequential flow. (Pressman, 2005, p.77-90)

In the incremental process model the software itself is produced in increments. In this case, the first delivered product has a limited set of functionality and features (typically the core product) and with each release this functionality is expanded upon. The incremental model consists of multiple linear sequences (elements of the waterfall model) that can be ran concurrently. Each of these linear sequences produces a deliverable increment of the software. This process model can be useful when insufficient personnel are available at the start of the project, if there is a danger related to the acceptance of the core product or if the project contains technical related risks. (Pressman, 2005, p.77-90)

Evolutionary process models take into account the fact that software must evolve over a

period of time. Thus the model attempts to accommodate a product that will evolve over time. The model produces an increasingly complete version of the software with each iteration, focusing on the flexibility, extensibility of the software and the speed of development. (Pressman, 2005, p.77-90)

Specialized process models are the models that do not fit to the aforementioned categories. However they can still apply many of the characteristics of the conventional models. For example, a component-based development process model incorporates many of the characteristics of an evolutionary spiral model, yet the applications themselves are composed from prepackaged software components. (Pressman, 2005, p.91-100)

According to Koutonen (2011, p.92,102), in a thesis involving 20 Finnish gaming companies, Finnish gaming companies use agile development methods extensively. Of all the companies that answered to the section of agile development methods in Koutonen's questionnaire, all but a single company used agile methods in their game development process. Chandler (2008, p.41-42) mentions that scrum and personal software process (PSP) have been successfully used by game developers in the recent years.

For a definition of agile development, the agile alliance (people including developers, writers, consultants) have provided a general manifesto for agile software development and defined 12 principles on how to achieve agility (Beck, et al., 2001). In short, agility means being able to have an effective response for change, encourages team structures and attitudes that make communication more fluent between different parties, emphasizes rapid delivery of operational software, de-emphasizes the importance of intermediate work products, adopts the customer as part of the development team, recognizes that planning in an environment of uncertainty has its limits and recognizes that a project plan must be flexible. There are a number of different agile process models (defined by different parties) that have similarities in philosophy and practice and all of them, some more than others, conform to the agile manifesto and the 12 principles. (Pressman, 2005, p.103-110) Keith (2010) describes how scrum, extreme programming (XP) and kanban with lean principles can be used in game development.

In scrum a cross-discipline team of six to ten people make progress in iterations in so called sprints, that typically last from two to four weeks. The sprint starts with a planning meeting, in which the team selects the features that are to be implemented in the sprint from a prioritized list of features that is referred to as the product backlog. When the implemented features have been selected for the sprint, the team will estimate the tasks that need to be performed for each feature and the time each task will require. These tasks are then placed to a sprint backlog. The team will only commit to features in a sprint which they judge to be achievable. During the sprint itself the team will meet in a daily scrum, which typically lasts 15 minutes. The meeting has a set time limit and will end whether all items on the agenda have been addressed or not. The goal of this meeting is to share the progress and impediments the members of the team have faced to other members of the team. At the end of a sprint the team has produced a playable version of the game, which doesn't necessarily pass all the tests necessary for the game to be ready to be shipped. The different stakeholders of the game (managers, directors, publisher staff) gather in a sprint review meeting to evaluate whether the goals of the

sprint were met and to possibly update the product backlog for the next sprint based on the experience of the previous sprint. The team will also hold a sprint retrospective after the sprint review meeting. In this meeting, the team reflects how effectively the team worked together in the last sprint and find ways to improve the practices that are in use. (Keith, 2010, p.35-57)

Extreme programming introduces new practices for software developers. Keith (2010, p.210-220) focuses mainly on test driven development and pair programming. In extreme programming programmers work in pairs on a task applying test driven development practices to build software in small, functional increments. The pair programming is organized so that one person is responsible for writing code while the other programmer watches and provides input where necessary. There are numerous practices to follow in strict test driven development. Programmers should create the absolute minimal amount of functionality to satisfy the user requirements in each iteration. Because of this practice constant refactoring of the code is needed. A number of unit tests should be written for each function in the software before the function itself has been coded. The unit tests and their code should be built in parallel. When all of the unit tests pass the code can be safely committed. This is achieved with the help of a continuous integration server, which runs all of the unit tests on commit, revealing a majority of the problems caused by the commit. When a build finally passes all of the unit tests, it is safe to synchronize the changes with the main repository. If there are errors, the integration server informs the team of this and it becomes the team's priority to fix this issue. This is typically the responsibility of the people who were responsible for the commit. This continuous integration practice should be performed frequently, even as often as every hour or two. As the project grows in size, the number of unit tests also increase, even to the thousands, and the tests continue to catch errors throughout their lifetime. They also make code refactoring and large changes to the existing software more feasible and less daunting for the developers. (Keith, 2010, p.210-220)

With a Kanban board one is able to visualize the flow of a production stream, timebox the production stream and balance the workflow. Balancing the workflow ensures everyone always has work to do and that there are not too many items under construction at the same time. It also makes continual improvements to the workflow possible. In the first step to produce a Kanban board, the desired production stream will be placed to the board. Columns represent the steps in the production stream, and work products will be placed as cards under these columns. (Keith, 2010, p.139-140) In an example provided by Keith (2010, p.141), a production stream for levels is introduced where the columns include backlog, concept, level design, high resolution art, audio design and tuning pass phases. The individual levels can then be placed as cards under these columns depending on which phase they currently are in. This visualizes the flow in the production stream. When the production stream has been visualized, the team can start to level the production flow by using lean tools to smooth out the fluctuations of production. Two basic lean tools that achieve this are timeboxing and the balancing of resources. When timeboxing is taken into use, each of these columns are given a timelimit, e.g., the audio design of a level is given a 10 day timebox. Choosing a suitable timebox for the steps in the production stream means balancing quality and cost, and measuring it with time. If the time is too short, the quality of the asset will be low, and if the time is too high, the quality of the asset will be high with a great cost to

the stakeholders. It is important to note, that the value of an asset to a customer and the cost of an asset do not have a linear relation, there is a point where the value to customer starts increasing at a slower pace (creating diminishing returns). The pre-production phase is responsible for shaping this curve and the best range for a production timebox. The chosen timebox is an average of best- and worst-case times, which are not precise. In addition, the kanban board can also include buffer phases, which only exist to give variance to the available time of completion of a phase. Finding this sweet spot is the goal when timeboxing the production stream. As each step in the stream usually requires a time box of a different length, it is possible that gaps and pileups of work will be generated. To avoid this, the team needs to balance the available resources by allocating more resources to the phases that have longer cycles. The goal can be e.g., to have a ten day cycle for each phase. When the production stream has been balanced, it is possible to forecast the rough speed at which the production stream produces an item. By continually improving this stream, it is possible to increase the rate at which finished assets are delivered.

The personal software process mentioned by Chandler (2008, p.41-47) is a process-improvement approach. It attempts to teach engineers to manage the quality of their code, to make commitments that they can actually fulfill, improve their estimation and planning skills, and reduce the amount of bugs in their code. (Chandler, 2008, p.44) The improvements happen in distinct levels from PSP0 to PSP3, where each level has sets of logs, forms, scripts and standards to help the engineers to improve the software process. The scripts define what to do in each part of the process, the logs and forms work as templates for recording and storing data of the process in use, and the standards provide guidance to the engineers. (Humphrey, 2000A) The team software process (TSP) can be combined as a component for PSP. Where as PSP focused on engineering disciplines, TSP focuses on team and management disciplines. It contains a team-building process that addresses team disciplines such as goals, plans, commitments, roles, resources, quality ownership, plan ownership, plan detail and a team-working process that addresses management disciplines such as cost of quality, review status, review quality, communication, change management, the ability to follow the process. (Humphrey, 2000B) The goal is to build teams that are capable of directing themselves, able to establish the project goals, formulate a plan to meet these goals, and track their progress towards these goals (Chandler, 2008, p.44).

3 SOFTWARE TESTING

According to an IEEE (1990) definition, software testing is an activity in which a system or component is executed under specified conditions, the results are observed or recorded and an evaluation is made of some aspect of the system or component. For another definition, Hetzel (1988) states that testing is any activity that is aimed at evaluating an attribute or capability of a program or system and determining that it meets the results expected of it. Testing should not be mixed up with software quality control or software quality assurance. Quality control ensures that each work product meets the requirements placed upon it by using inspections, reviews and tests throughout the development process (Pressman, 2005, p.746). Quality assurance assesses the effectiveness and completeness of quality control activities with the help of auditing and reporting activities (Pressman, 2005, p.747). It is also important to separate testing from debugging. Whereas testing as an activity finds defects in the program, debugging as an activity attempts to find the code segments responsible for the defects with a goal of repairing them.

When creating software, a mistake made by a person in either the specification, design or code phase becomes a defect in the executable code unless it is caught and fixed. When the code that is affected by this defect is ran, a failure can be observed when this fault turns into a visible bug. The bug prevents the program from operating as intended, there is a variance between the desired and actual behavior. Testing reveals failures, the major goal is finding the defects and removing them. (Williams, 2006) Most of the difficulty in software testing comes from the complexity of software itself. For anything but trivial programs, it is not possible to test a program fully and be assured that the program is bug free. As it is not possible to test everything, choices regarding what to test must be made and the quality of the program must be actively assessed instead of merely confirming it. The goal is to have a high likelihood of finding the defects that are most likely to cause a failure for a large segment of users, with as little testing as possible.

3.1 Verification and validation

Software testing is one of the practices in the so called verification and validation (V&V) processes. IEEE (1990) defines verification to be a process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. In other words, the process checks that the system meets the initial design requirements, specifications and regulations. Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements (IEEE, 1990). In other words, validation is used to evaluate whether given features are traceable and satisfy the customer requirements (Pressman, 2005, p.388). For another definition, Boehm (1984) has defined the terms with the help of questions. Verification answers the question, are we building the product right? It makes sure the product behaves in the way the authors wanted it to. Validation answers to the question, are we building the right product? It makes sure the product is what the customer asked for, making comparisons against requirements.

Even though testing is an important part of verification and validation, other activities are also required in V&V. These include e.g., formal technical reviews, quality and configuration audits, performance monitoring, simulation, feasibility studies, documentation reviews, database reviews, algorithm analysis, development testing, usability testing, qualification testing, installation testing (Wallace and Fujii, 1989, p.10-17).

3.2 Black box and white box testing

Most of the testing performed on games is black box testing by nature (Schultz and Bryant, 2011, p.126) (Morgan, 2012, p.5). In black box testing the inner structure of the software under test is not known. The tester doesn't have any knowledge of the program's source code. Defects are found by merely using the inputs available for the typical user and observing the output produced by the program. In the case of video games, these can range from controllers (game pad, keyboard & mouse, motion sensor), audio (microphone), video (camera), packets over the network and stored data from memory devices (memory cards, hard drives) (Schultz and Bryant, 2011, p.127). Video games also differ from the typical program in that they are ran continuously until the player stops playing the game. This creates a so called feedback loop, in which the player's input affects the produced output of the game, and the player makes adjustments to input depending on the output of the game (Schultz and Bryant, 2011, p.128).

Black box testing has multiple benefits. Firstly, the testers can easily get in to the mindset of the end user, as they do not have to work with the source code, they interact only with the game. Secondly, as the testers can solely focus on the outcomes of the game, the testers themselves will not get distracted by processes related to testing. The testers also get a better view of how different inputs affect the output of the game and check if these were the desired outcomes. Finally, black box testing is relatively easy to perform, the tester doesn't require any specific knowledge of programming and mostly just needs to record the outputs of a given input. (Morgan, 2012, p.5).

In white box testing, the inner structure of the program is visible for the testers, giving the testers the ability to execute the source code in ways the player is not able to. When performing the tests, specific modules and the different code paths these modules can take are executed. The input is the data that can be passed to the piece of code under inspection and the results can be checked by inspecting the values the code returns and the affected global and local variables (Schultz and Bryant, 2011, p.129). It is not feasible for a white box tester to read a piece of code and predict every interaction it will have with other pieces of a code even in a relatively small amount of code. In the case of video games, even mobile games contain enough complexity, making this approach unfeasible. It is also not feasible to test a game using only white box testing methods, as it is very difficult to take into account the complexity of the aforementioned player-game feedback loop (Schultz and Bryant, 2011, p.128). White box testing can include unit, integration, requirements validation and system levels of testing.

Regardless of the fact that not every code path can be explored by using white box testing, there are still situations in which the method is more practical than black box testing. Schultz and Bryant (2011, p.128) mention five different situations in which the

aforementioned is true: the tests the developer did prior to submitting new code for integration, the testing of code modules that will become part of a reusable library across multiple games or platforms, the testing of methods or functions that are essential parts of the game engine or middleware product, the testing of modules that might be used by third party developers and the testing of low-level routines that the game uses to support specific functions in the newest hardware devices.

Morgan (2012, p.6) mentions a few benefits of white box testing. Firstly, white box testing makes it virtually possible to test any feature, as long as the testers themselves understand the code of the feature in question. This level of control may not be possible for the tester using black box testing on a given feature, as the tester can only use the standard controllers. This also makes it possible to find bugs that could not be found using other methods. Secondly, some features can only fully be checked by using white box testing, such as middleware and hardware drivers. Finally, as the testers must get acquainted with the code in detail, they may discover improvements to the code at the same time, such as optimization or redundant code that either provides nothing or unneeded features.

Morgan (2012, p.7) recommends that both white box and black box testing should be used in testing a game as this gives a better test coverage resulting in a higher quantity of bugs being found. However, Morgan also notes that due to high time constraints in game development and the large amount of interactivity in a video game, black box testing should be used more often.

3.3 Levels of testing

The testing activity can be split into four different levels of testing: unit testing, integration testing, validation testing and system testing. (Pressman, 2005, p.390)

In unit testing the tests are focused on a single component testing that the module functions individually as a single unit. It is largely comprised of testing techniques that exercise specific paths in a module's control structure to ensure complete coverage and maximum error detection. (Pressman, 2005, p.390-391). In essence, the testing verifies that the code works as intended at a very low structural level. Unit testing is typically done by the programmers themselves. (Williams, 2006)

As the modules are individually tested, the next phase is to integrate the modules to create the complete software package. Integration testing in this phase is needed, because even though the modules have been confirmed to work individually, there are no guarantees that the modules function properly together when integrated e.g., data might get lost in an interface, messages might not be passed properly, interfaces may be implemented incorrectly. (Williams, 2006) During integration testing, the main focus is on input and output between the different modules and the major control paths of the entire package (Pressman, 2005, p.391). Integration testing is typically done by the programmers themselves (Williams, 2006).

After the integration of the software package has been satisfactorily tested, validation testing ensures that the software meets the functional, behavioral and performance requirements (Pressman, 2005, p.391). In other words, it ensures that the functionality that has been specified in the requirements specification has been implemented and that

it functions as specified in the specification. Validation testing is typically done by independent testers (Williams, 2006).

In the final level of testing, system testing, the system must be tested in an environment similar to its actual usage. This means combining the software with the system's other elements, these include e.g., hardware, people, databases. System testing ensures that all elements of the system, and the environment the system is placed, function properly together and that the desired functionality and performance is still achieved. It can be e.g., recovery testing, security testing, stress testing, performance testing. (Pressman, 2005, p.391,409-410) System testing is typically done by independent testers (Williams, 2006). In general, the longer it takes to find an existing fault and the further the development process advances, the more difficult it is to find the defect that causes the failure. Because of this, the sooner the defect is found, the better.

4 GAME DEVELOPMENT

"*Game development can be seen as a specific form of software development where certain product and/or service is designed and developed.*" (Manninen, et al., 2006, p.5)

Over time, games have become more and more complex. The consumers desire for an experience that tops the one felt in a previous game. This means the developers have to increase the amount of features in a title, create more code that is more complex and create more assets of a higher quality level. The typical console game gets released on multiple different platforms in different languages. For these reasons, game developers have started to adapt different software engineering processes to game development (Chandler, 2008, p.41). Some of these processes were presented in chapter 2.1 Process models of this thesis.

Video games can vary greatly from each other. They can be small mobile games that take weeks to develop or massive-multiplayer online (MMO) games that can take over four or five years to develop (Schultz and Bryant, 2011). They can be indie games created by a relatively small team or publisher-driven games created by a team consisting of hundreds of team members. They can be of completely different genres, one being a race driving game and the other being a fantasy adventure game. They can be so called serious games (e.g., educational) or aim for the entertainment market. They can be on different platforms and use completely different form of controllers. They can be license based games or built complete around an original intellectual property (IP).

The average budget for a PC or online casual game is below 400,000 USD with a team consisting of 10 – 20 people working on it with a development time of six months or under. However, less casual games such as MMORPG or FPS games have budgets varying from 10 – 50 million USD, have teams consisting of hundreds of people and a production time ranging from 2 – 5 years. (Ravago, 2009) Another source places the average budget for a multiplatform next-gen game somewhere around 18 – 28 million USD and over 40 million USD for a high profile game (Crossley, 2010).

4.1 Business models

Depending on the chosen business model or models, the nature of the required development and thus required testing work may change. As an example, with the introduction of digital distribution, games of a smaller scale now have a way to get to the market and be profitable. Perry (2008) provides an extensive list for possible ways for gaming companies to gain revenue:

1. **Retail.** Selling traditional boxed copies at retail stores.
2. **Digital distribution.** Selling games over the internet, bypassing the traditional retailer and distributor. Has made it possible to create and sell games of a smaller scale, e.g., XBLA, mobile, indie games.
3. **Episodic games.** The game is released in episodes, which must be bought seperately or as a single purchase.

4. **Subscription model.** A subscription must be paid to reserve the right to play the game. The game must provide enough value that the player sees the cost worth paying for. Typical of MMO games.
5. **Micro-Transactions.** Small purchases made inside the game for different benefits or cosmetic changes. Typically used in free-to-play games, where the base game itself is provided to the players for free.
6. **Pre-Sell the Game.** Sell the game in advance for the fans of the game. When the game finally comes out, those who pre-bought it, get the title for "free".
7. **In-game advertising.** Gaining revenue from any form of in-game advertising (e.g., billboards in the game).
8. **Around-game advertising.** Gaining revenue from adverts that circle the window in which the game resides.
9. **Advertgames.** The entire experience of the game is an advert, e.g., America's Army.
10. **Finder's fee.** If you are paying a finder's fee for the provided customers, this fee can be payed from the first profits created by the customer.
11. **"Try Before you Buy" / Trialware / Shareware / Demoware / Timedware.** Providing a restricted or shortened version of a game for free, to increase sales of the retail product.
12. **Skill-Based Progressive Jackpots.** A tournament is held, where the winner takes the pot. The revenue is a percentage of the jackpots.
13. **VIP access.** The VIP members pay a fee for special privileges.
14. **Sponsored Games / Donationware.** Serious games, that are funded by donations.
15. **Pay per play / Pay as you go / Pay for Time.** The player pays for being able to play the game, e.g., by having a time limit or a set number of lives.
16. **Trading of Virtual Items.** Revenue is generated from taking a cut of all trades made by players.
17. **Foreign distribution deals.** Gaining revenue from selling foreign distribution rights in advance.
18. **Sell Access to your Players.** Gaining revenue from the user base, by e.g., special offers, personal profile questions.
19. **Freeware.** With enough user base, there can be offers for the company, software or specific technology.
20. **Loss Leader.** Gaining revenue from selling additional products related to the game, that is not making profit.
21. **Peripheral Enticement.** The game relies on an additional peripheral, that generates revenue in addition to the game.
22. **Player to Player Wagering.** The players can wager against each other, while the

revenue comes from a percentage of each wager or from the cost of items being wagered.

23. **User Generated Content.** Gaining revenue from the content the users are selling by receiving a cut of each sale.
24. **Provide storage space.** Gaining revenue from providing player's storage related to the game.
25. **Private game server.** Gaining revenue from hosting private game servers.
26. **Licensing Access.** Gaining revenue from licensing the game to third parties (e.g., cyber cafes, tv shows).
27. **Selling Branded Items.** Selling additional material related to the game.
28. **Buy Something, get the game for Free.** The game is in a bundle with another item.

One important aspect which seems to be missing from this list is the developers ability to release additional expansion packs and DLC for the game. DLC is typically delivered via a digital distribution platform, however, if it majorly expands the game, it is typically marketed as an expansion pack for the game, which may also have a retail release.

Manninen, et al. (2006) also mention revenue gained by software platform developers, hardware platform developers and contract services. Tool developers gain revenue from creating game engines and other middleware, which they in turn sell to the game developers. They typically license the products, and provide consulting services to the developers if requested. The hardware platform developers may lose money when selling the consoles, but they license the platform for the game studios or publisher. They typically collect a fee for each game sold. Finally a contract company has specialized in an area of game production, e.g., motion capture, sound effects or models. A studio can hire them to produce material as an outsourcing partner.

4.2 Major roles in game development

4.2.1 The big picture

Manninen, et al. (2006) introduce several key actors in game development. These include game developer, platform developer, publisher, funding body, distributor, retailer, and consumer. The game developers are the team who is in charge of the game development process. They design and develop the game. Platform developers provide the platform (hardware e.g., a console or software e.g., a game engine), on which the game runs. The publisher typically funds the project, while the funding body can be additional third party partners, who will also support in funding the game. These can include venture capital, non-profit organizations and individual agencies. The publisher itself can also provide other roles (e.g., by providing quality assurance services), these are defined in the contract made between the publisher and developer. Consumer is the customer who in some form provides the revenue, e.g., purchasing the game, they are typically not in contact with either publisher nor developer. The distributor distributes

the game to either retailers or straight to the consumers. If a retailer is used, the retailer will handle distributing the game to the final destination, the consumers.

With digital distribution the retailer and distributor are no longer needed as actors in game development. Instead, the platform which provides the games to the consumers delivers the game digitally via some form of download. This type of digital platform e.g., Steam, Uplay, Xbox LIVE typically collects payments from game developers that wish to have their game on the platform, however, it is cheaper for the developers than using a traditional distributor / retailer. Most console games today are still released in a physical format without a connection to a digital distribution platform (the console can be kept offline).

Hight and Novak (2007, p.165-172) introduces the various roles and major responsibilities of the development team itself. These roles include the producer and the management team, the design director and the design team, the technical director and the programming team, the art director and the art team, the audio director and the audio team and finally the qa director and the testing team.

4.2.2 The management team

The producer provides the leadership for the development project and is the main information conduit between the production team and anyone external to the team e.g., the publisher (Chandler, 2008, p.18; Hight and Novak., 2007, p.165). Their job is to report progress, follow up / solve different issues as they arise and to ensure that the game is done on schedule, within budget, and as close as possible to the vision presented in the game design document. If the project is large enough, the project might require additional associate producers, whose responsibility is to oversee specific areas of production. These include art production, level design, localization, online/multiplayer interface, cinematics, audio, and licensor/publisher communications. (Hight and Novak, 2007, p.165) In addition to these so called developer producers, the publisher may also have a producer of their own. This producer works with the external developers representing the publisher's interests. They typically oversee departments that are not directly involved in game development. (Chandler, 2008, p. 20-21)

Chandler (2008, p.19-23) defines two additional production roles: the executive producer (EP) and the associate producer. Executive producers typically have five to 10 years of experience, oversee multiple projects, manage multiple producers and focus on broader production tasks, e.g., establishing employee training programs, negotiating contracts, evaluating external vendors. The associate producer (AP) typically has one to three years of experience, assists the producer with any production-related tasks and may oversee a specific area of production. A single project may have multiple APs.

4.2.3 The design team

The design director, creative director, game director and lead designer are all responsible for the design of the game. The design director is the main contributor to the game design document and holds the vision of the game, thus also being responsible that the shipped product is fun to play. If some members of the development team require additional information of a given feature, the design director should be the first

person to be contacted. In some cases the design director might be responsible for overseeing the design team, although this task can also be handled by the assistant producer. This is true especially in large projects. (Hight and Novak., 2007, p.165-167) The creative director typically communicates this creative vision to the team and ensures that the vision is followed throughout the project. The tasks of the lead designer typically includes managing the daily tasks of the design team, directing the team as needed and managing the communication between the creative director and the designers. (Chandler, 2008, p.30-32) The design team typically consists of level designers, scripters, interface designers, writers, researchers and game tuners (Hight and Novak., 2007, p.167).

Level designers are responsible for creating the environments (levels) in the game. If the game uses a scripting language, scripters build the scripts which handle npc behavior, events in the game or other in game logic. Interface designers define the control scheme. This includes defining onscreen buttons, menus, defining the flow of control in the game, mapping game functions to the controller's buttons and defining the visual / auditory / touch feedback system for the player (Hight and Novak, 2007, p.167). Writers and researchers have the responsibility of providing the dialog and a lot of the background story / additional information for the game. Finally, game tuners are responsible for the pacing and difficulty of the game. Typically this is done by editing the enemy toughness, frequency and placement. Game tuners may observe people playing the game in gameplay focus testing to identify problematic areas in the game. (Hight and Novak, 2007, p.167)

4.2.4 The programming team

The architecture of the game is the responsibility of the technical director. The technical director is also responsible for managing the software development team, creating the technical design document, building the game and delivering a high quality game that follows the requirements set in the game design document and the technical design document. The programming team may contain one or more lead programmers for specific areas of the game, depending on the size of the development project. Typical areas of responsibilities and specialization include networking, artificial intelligence, path finding, physics, 3d rendering and shading, lighting, animation, sound, streaming, interfaces, database, security and various tools required by the project. (Hight and Novak, 2007, p.168)

4.2.5 The art team

The typical art team positions include the art director, lead artist, concept artist, level designer, asset artist, animator, technical artist and marketing artist (Chandler, 2008, p.23-26). The art director is the main author of the art style guide and visual aspects of the game are one of the main responsibilities of the art director. In addition, the art director may manage the art team or the art director may merely provide guidance to the team, leaving the administrative work to the associate producer. (Hight and Novak, 2007, p.169) The lead artist manages the quality of the art assets and the tasks of the art team, ensures that the artistic vision is maintained, and handles the communication between the art team and the art director. In the absence of an art director, the lead artist

assumes the responsibilities of the post. (Chandler, 2008, p.24) The art team typically works closely with the design team, and some artists may even belong to the design team. The typical tasks include modeling, texturing, animation, interface art, camera and lighting, environments, characters, cinematics, and tools. The art team is typically larger than the rest of the development team, and art is one of the largest components in current-generation games. (Hight and Novak, 2007, p.169)

4.2.6 The audio team

The audio director / audio manager is responsible for all of the audio in the game from sound effects, music to dialogue. Typically a lot of development studios use third-party contractors to design and supply at least a part of the game's sound, thus the audio director is responsible for maintaining a network of musicians, composers, sound designers and recording houses. Some of these contacts may have little or no experience in crafting audio for games, so the audio director must provide guidance for the audio team. (Hight and Novak, 2007. p.170)

4.2.7 The testing team

The quality assurance director / manager (Levy and Novak, 2010, p.75) is responsible for developing the testing plan and managing the testers during a game project. It is typical, especially when the developers are working under mile-stone contracts, that testing is handled by the publisher of the game. If the development studio is large enough, it becomes more vital to bring testing closer to the development team, and the studio may have a testing team of their own. The members of the testing team have a responsibility of reporting all issues found with the game and to provide insight about the overall quality level of the game. (Hight and Novak, 2007, p.171) The structure and responsibilities of the testing team and different forms of testing are discussed in more detail at later chapters.

4.3 Game quality factors

Traditional software is typically built to a need which the end user of the software requires. E.g., the user might need a specific tool to convert the data to another format or to manage a database, etc. The tool can be evaluated against how well it performs the job it should perform. However, games that target the entertainment market are not built for a current or future need for a tool. Instead, the reason the game exists is to entertain and immerse the player. If the player is not enjoying the game frequently enough, it can be seen as not fulfilling the “task” it was built for. This difference in the goal of the two products changes the way quality is evaluated. E.g., in a traditional software program the efficiency should be maximized, however in a game, the most efficient process provided for the player may not be the most fun. Systems in the game should be built to maximize the amount of player enjoyment.

The term fun factor, the amount of fun a game is, was chosen to be used in this thesis and the interviews. The term was used by e.g., Schultz and Bryant (2011) and Levy and Novak (2010). The terms player enjoyment, game enjoyment or engagement can also be found in literature. All of these terms mean mostly the same thing. It is difficult to

analyze and measure this fun factor of a game. Some aspects of a game can't be considered universally to be fun. In this case, a certain group of players will like it where as others will not. It is also difficult to measure the amount of fun a person is currently having with a game and the amount the player is immersed in a game at a given moment. While it is possible to ask the players if they had fun, it is hard to quantify and during a play session, this will break the player's immersion. Methods for evaluating player enjoyment in games are presented in later chapters.

Manninen (2007, p.205), in a chapter related to evaluating game ideas, presents a few different factors of a game, that can be seen to be related to the quality of the game. These include the originality of the game, the consistency and uniformity of the game (vision of the game), the amount of interaction in the game and the amount of interest the game generates in a player. The game can be original in its features, style, graphics, etc. It can be either completely, partly or hardly original at all. The originality of the game is important as this defines the unique selling points the game has. If the game has no unique selling points, customers who have already experienced a similar game or games will be less and less enthusiastic of the game, when more similar games enter the market. The game should remain consistent and uniform throughout, so that the vision of the game can be seen, the theme of the game stays consistent from start to finish and the entirety of the game works together as a whole. The game should contain a fair amount of interactivity as this is expected of a game. In a game with low interactivity, the customer may feel like they are watching a movie or reading a book instead of playing a game. The amount of interest the game creates in the player (e.g., the subject of the game, world, characters, choices) is also important as this ensures the player wishes to see these parts of the game reach their conclusion and hook the player to the game.

Schultz and Bryant (2011, p.70,128) mention a few different quality factors in a game that are likely to be important for many players. These include the quality of the feedback loop between the player and the game, quality of the story, quality of the game mechanics, quality of the in-game audio, quality of the visuals (e.g., style, realism), the appeal of the visual style, quality of the downloading and updating experience, quality of the artificial intelligence, quality of the game interface (UI and controller input), quality of game performance and the use of humor and exaggeration in the game. According to Schulzt, et al. (2011, p.128), the feedback loop between the player and the game should be just random enough to be unpredictable, thus adding to the game's fun factor.

4.4 Game development process

A game is typically developed and approved in stages, which are defined by milestones. The milestones are typically a form of contract between publisher and developer. When a developer fulfills a milestone, they are paid a predetermined amount of money. (Fullerton, et al., 2004, p.347) If the milestone is not achieved the developer will typically not receive the money, and the publisher may decide to discontinue support for the project.

Keith (2010, p.130) gives three other major reasons for a stage based structure. Firstly, publishers require detailed concepts, and these concepts should form a vision that holds throughout the project. The project should never stray too far from this vision. Secondly, typically games need to deliver eight to twelve hours of single player gameplay. Keith likely refers to AAA games, as e.g., smaller downloadable games or indie games are typically of a shorter length. In any case, according to Keith (2010, p.130), this time requirement leads to a large amount of production content, that is to be created using the tools built during the pre-production phase. Finally, there is only a single shipping date at the end of a long (24+ month) development cycle. In some cases, especially in movie license based games, missing this deadline is not an option.

Each development company follows its own adapted and modified game development process (Manninen, et al., 2006). However the development process of these games can still be seen to follow a generic basic structure, which they all share. The process can be seen as consisting of different phases. Keith (2010, p.131) defines the phases accordingly:

- Concept
- Pre-production
- Production
- Post-production

Manninen, et al. (2006) have additional phases in their model. The phases are:

- Concept – Specification and planning
- Pre-production
- Production / Development
- Quality assurance / Validation and testing
- Release & Launch / release to manufacturer phase
- Post-release / Maintenance

4.4.1 Concept

In the concept phase, the idea of the game is developed in an iterative fashion by generating and disregarding ideas on a regular basis. Building prototypes to test an idea is a possible approach in this phase. The result of this phase is a high-concept document, which does not contain all the details, but gives the team a clear understanding of what the game is about. (Manninen, et al., 2006; Keith, 2010, p.131)

4.4.2 Pre-production

In the pre-production phase, the team attempts to find out what is fun and what works in the game in an iterative and incremental process. With this, the core mechanics of the game are finalized. A second goal is to improve the process and tools that are used to

build the game assets in the production phase, so that when actual production begins, production is as streamlined as possible. This phase can also deliver a playable demo or a prototype of the game, that represents the final production quality. (Manninen, et al., 2006; Keith, 2010, p.131)

4.4.3 Production

In the production phase the team uses the core mechanics of the game and the processes / tools discovered during the pre-production phase to create the game in its entirety. The focus is on efficiency and incremental improvements. The team avoids changing the core mechanics of the game, because a lot of the assets, e.g., levels and animations are built based on the information agreed on the pre-production phase. Changing these mechanics during production means potentially a lot of the assets that have already been created need to be updated, thus this would be time-consuming and expensive. Keith (2010) includes an additional post-production phase in the development stages he describes. In this phase, the content produced in the production phase is now of a shippable quality and the team focuses on polishing the entire game experience, incrementally improving the game. (Manninen, et al., 2006; Keith, 2010, p.131)

It is important to note, that even though quality assurance has been defined as a specific phase in the entire production process, Manninen, et al. (2006) claim that quality assurance tasks and testing are usually integrated into all production phases. Testing can be seen as a continuous process that focuses on the quality of the game and attempts to assure the best possible quality with the available resources (Manninen, et al., 2006).

The release phase contains the last phases of production. In this phase, more and more of the game features are frozen (changes are not permitted) and release candidate versions of the game are built from which the final gold master version of the game is chosen. User documents, different forms of demoing the game, support services and localization are also finalized. (Manninen, et al., 2006)

4.4.4 Post-production

The post-release phase contain activities that support the game after it has been released to the world. This includes patches, upgrades and add-ons (DLC, downloadable content), PR and marketing, handling of the game's community and the professional game development community. (Manninen, et al., 2006)

According to Keith (2010, p.132) the stages are not isolated to distinct periods of time, that is, they can overlap with each other. Concept development can occur at the same time with pre-production, even though the majority of the concept work is done upfront. In fact, the concept itself can be refined over the course of the entire project. Pre-production related activities can continue in smaller amounts in production and post-production phases. Production related activities can still continue during post-production phase. In other words, instead of a specific start date for a phase, there is a gradual build of the activities required for the new phase and a gradual decrease of the previous phase activities. Manninen, et al. (2006) also point out, that the phases and tasks do not follow a strict waterfall model, in which the phases are sequential and

follow each other in a particular order. He emphasizes that most of the of the game design and and development is iterative, these iterative looping cycles are formed by design-implementation-testing phases, that can occur in any point of the production. These iterative cycles make it possible for the team to get feedback regarding the current state of the implementation.

5 TESTING IN GAME DEVELOPMENT

The main goal of testing is to tell the development team what is currently wrong with the game. As such it is not possible to build quality in to the game by simply performing testing. The quality of the game consists of the quality of the code, the art, the audio and the feedback loop between the game and the player. Starting from the beginning of the development project, testing can get problems fixed sooner and with less costs. (Schultz and Bryant, 2011, p.100)

5.1 The goals of game testing

The publisher's goal is to ensure that the game is of sufficient quality before being released. If the quality of the title is not good enough, this will be reflected in the reviews of the game and directly influence the number of units sold. For the same quality related concerns, the development team of the game wish to find the defects and other problems with the game. Even a single shipped title that is received poorly can damage the reputation of the development company. The company may even run out of funds if the shipped title was not successful enough. Also, if the developers and the publisher have made a contract, where money is received by meeting milestones, the performed testing will prove that these milestones have been met. Console and mobile manufactures want to make sure that the quality standards they have defined are met before the game is shipped (Schultz and Bryant, 2011, p.44). Their goal is to ensure that the games released under their platform meet a predetermined quality level, so that single titles will not tarnish the reputation of the entire platform, thus damaging the sales of the platform itself.

5.2 The roles in game testing

Levy and Novak (2010) identify following major roles in testing:

- producer
- QA (quality assurance) manager
- lead tester
- floor lead / primary tester (term used by Schultz and Bryant, 2011, p.103)

The producer oversees the project over the entire length of development. The main tasks of the producer are keeping care of different deadlines / milestones and to take care that the project runs on the budget it was allowed. (Levy and Novak, 2010, p.74)

The QA manager's primary tasks are to determine what kind of production testing is needed, to devise a schedule for testing and to make sure the testing costs do not go over budget. The QA manager also makes sure the environment is sufficient for the lead tester and the testers to perform the testing in. (Levy and Novak, 2010, p.75)

The lead tester is responsible for a lot of the activities that make testing itself possible. The main tasks of the lead tester include providing the individual testers with daily schedules and items to test, reviewing all the bugs submitted by the team and making

sure the bug reports are up to the agreed upon standards, making regular reports of progress to their superiors and possibly taking part in the testing activity themselves. (Levy and Novak, 2010, p.76)

The floor lead role is not mandatory. This role is created in support of the lead tester, if the lead tester is in charge of many different departments / floors. The floor lead acts as an “unofficial” lead tester, having no real authority. A game project might have several floor leads or none at all. (Levy and Novak, 2010, p.76) Schultz and Bryant (2011, p.103) also mention this role, but instead use the term 'primary tester'. The term 'vice lead tester' is also used.

In addition, Chandler (2008, p.35) mentions the role of the regular QA tester. The testers are responsible for finding the defects in the game software. They use the test plan to test the game's functionality, new features, possible prototypes and confirm that the console manufacture's technical requirements are met. Most of the testers work revolves around playing the game under production, so they usually have informed opinions on the fun factor of the game. (Chandler, 2008, p.35)

5.3 Bug categories

A game can contain bugs in any of its features, from audio, to artificial intelligence to networking. Levy and Novak (2010, p.77) provide an extensive list of different kinds of bugs and their categories. The bug types are explained in more detail in the next section.

Visual

- Clipping
- Z-Fighting
- Screen Tearing
- Missing textures
- Visible Artifacts

Audio

- Audio drop
- Skipping
- Distortion
- Missing sound effects
- Volume level

Level design

- Stuck spot
- Sticky spot
- Invisible Wall
- Map Hole

- Missing geometry

Artificial Intelligence

- Pathfinding bugs
- NPC bugs

Physics

- Breakable objects related bugs
- Dynamic behavior (Faked vs Real physics)

Stability

- Freeze
- Crash

Performance

- Frame rate
- Loading times
- Minimum requirements
- Installation time

Compatibility

- Video Card
- Controller
- Operating System
- Other standards

Networking

- Failed, dropped connection, unaccepted invitation
- Lag
- Invisible player
- Scoring error

5.3.1 Visual

Visual bugs affect the game's graphics. Clipping occurs when a polygon overlays or penetrates another polygon. The end result is e.g., a character's hand going through a weapon model. Z-fighting is a texture related bug. As two textures are placed in the same depth, neither may have a priority over the other and the displayed texture seems to switch rapidly. Screen tearing occurs when the display device and the GPU (graphics processing unit) refresh rates are out of sync. The GPU cannot draw a frame fast enough, resulting in a picture where the edges of the objects fail to line up (in a

horizontal line). Missing textures are simply textures that are missing, typically replaced by placeholder textures. A visible artifact is an artifact on the screen which does not seem to be connected to any model or geometry in the game. They typically manifest as stray pixels. (Levy and Novak, 2010, p.78)

5.3.2 Audio

Audio bugs concern the games audio. In an audio drop, a part of an audio clip is simply not played. In skipping, a part of the audio clip is skipped entirely, jumping to a new location in the audio clip. Audio distortions occur when the sound is not being played as it should be (e.g., with the correct pitch). Missing sound effects mean nothing is played, where a sound effect should be played. If a single sound effect is played too loud or too quiet in regard to what is happening in the game world, this can be regarded as a sound volume level bug. (Levy and Novak, 2010, p.80)

5.3.3 Level design

Level design bugs are bugs related to the environment the player is interacting in. A stuck spot is a spot in the level, where the player gets stuck (cannot move) and cannot get out of. The sticky spot is a similar spot, however, with effort the player can get out of this spot. An invisible wall is geometry that is invisible to the player, that still prevents passage. When entering a map hole, the player will leave the intended playable area, and entering the area beyond the mapped geometry. (Levy and Novak, 2010, p.84) A typical example of a map hole is falling through the floor and being able to see the entire level from below.

Missing geometry means that the artwork for preventing player passage is present, but the preventing geometry is not. Thus the player seems to be able to walk e.g., through walls. (Levy and Novak, 2010, p.84)

5.3.4 Artificial Intelligence

Artificial intelligence related bugs affect the game characters behaving around the player. In pathfinding related bugs, the AI has trouble in navigating the map, e.g., running in circles or getting completely stuck. NPC behavior related bugs are a form of AI bugs. E.g., bad performance from a NPC teammate can be qualified as a bug. (Levy and Novak, 2010, p.86)

5.3.5 Physics

Games in modern systems typically use actual simulated physics, where as those with lesser resources have to “fake” the simulation. In breakable object related bugs, the objects that have been marked as “breakable” in the physics subsystem are malfunctioning in some way, e.g., they are not breaking at all. Other physics related bugs can occur when object don't act in a believable way to the player. (Levy and Novak, 2010, p.88) As an example of this situation, a player is stacking boxes and then takes one box away, which results in floating boxes.

5.3.6 Stability

Stability related bugs include freezes and crashes. In a freeze, the game stops completely, the sound may stop or go to a permanent loop and all input is completely ignored as the game remains in a frozen state. A crash on a console system differs from a freeze in that the screen goes dark. On a PC based system it is typically possible to recover from a freeze or from a crash, as only the application has crashed, not the operating system itself. (Levy and Novak, 2010, p.90) Levy and Novak (2010, p.90) have also separated a loading bug as a separate entity. If this occurs, the loading of a game stops, because an error happens in the loading process. The effect is similar to a freeze.

5.3.7 Performance

Performance related issues include frame rate issues, load time issues, minimum requirements issues and installation time issues. Frame rate defines how many frames per second (fps) the game is being rendered. (Levy and Novak, 2010, p.90) A PC typically renders to the display device's frame rate (frame rate depends on hardware and game settings), where a console's vary between game titles from 30 to 60.

If the fps drops below the agreed upon target fps for a significant amount of time, this is a frame rate related bug. If the loading time of a game goes over the agreed upon maximum value, this is a game loading related bug. If the minimum requirements machine cannot run the game or has trouble in providing the agreed upon fps, this is a min. requirements related bug. In an installation time related bug, the installation time of the game exceeds the norms, the installation size of the game must also stay reasonable or within the console hardware provider's limits. (Levy and Novak, 2010, p.90)

5.3.8 Compatibility

Compatibility related issues effect how the game runs on specific hardware. If the hardware is claimed to be supported by the game, it should function correctly with the game. The hardware itself can range from video cards, audio cards, controllers, operating systems to USB and bluetooth devices. (Levy and Novak, 2010, p.92)

5.3.9 Networking

Networking bugs relate to connectivity and used bandwidth. (Levy and Novak, 2010, p.94) Connections are typically either peer-to-peer connections or server-client connections.

In a failed connection the attempt to create a connection between the parties has for some reason failed. In a dropped connection an active connection between the parties has been achieved but the connection is severed for some reason. Lag is an issue, where the delay between input and the actual action on screen is too high. This is usually caused by dropped packets or excessive bandwidth usage. Other typical networking related bugs include malfunctioning invitations, invisible player models during a session, and game scoring related miscalculations. (Levy and Novak, 2010, p.94)

5.4 Testing as a process

The testing process can be seen as a loop between the tester and the developer. The testers first plan the tests, execute them on the code and report the found bugs to the developer. (Schultz and Bryant, 2011, p.131). The lead tester or the primary tester typically plan the tests themselves. (Levy and Novak, 2010, p.76) (Schultz and Bryant, 2011, p.132). The developers in turn inspect the reported bugs, debug to find the bugs in the code, fix the bugs in question and compile a new build of the game. The cycle continues as the testers plan the tests and execute them for this new build of the game. (Schultz and Bryant, 2011, p.131) The QA does not test each and every build the developers produce. Typically during an alpha new builds are available daily. Because of this the QA focuses on a build that seems initially stable and focuses the testing on this version of the build. The typical cycle can last from a few days up to a week. (Chandler, 2008, p.367)

Schultz and Bryant (2011, p.130) separate this process into six individual steps:

1. **Plan and design the test.** In each build the design specification could have changed, the game could support new configurations, old features might have been cut and bugs could have been addressed. Because of this, the planning and design of tests should be revisited in each build. The aim of testing is to make sure no news bugs were introduced when the aforementioned changes were made.
2. **Prepare for testing.** The code, tests, test related documents and the test environment should be updated and be aligned with one another. When the development team has marked the bugs as fixed for the build, the QA team can start running the tests.
3. **Perform the test.** Testers run the test suites against the new build. When a bug is found, it is examined, so sufficient information can be provided to the bug report. The more research is done in this step, the easier the bug report is to use and the more useful it is to the developers.
4. **Report the results.** The completed test suites are logged and all defects found are reported.
5. **Repair the bug.** The development team debugs the code to find the bug and repairs it. The test team can help in this phase by reproducing bugs, if the development team has trouble in making the bug manifest inside the game.
6. **Return to step 1 and re-test.** When the bugs are repaired, and any other additional features wanted for the build are done, the new build can be released to the team. This starts the cycle again, with new possible bugs to examine and new test results.

These steps apply to any type of QA testing. They can also apply to the entire game, a phase of development or an individual module or feature. (Schultz and Bryant, 2011, p.131)

Chandler (2008, p.367) mentions that it is possible to test different sections of the game on different builds. This is typically done when the development has progressed and the game has become more robust. E.g., when testers check a newly submitted level for

geometry and texture bugs, the level itself will not be tested again until the artist has fixed all the bugs and resubmitted the level. In the meanwhile the testers focus on other levels and features in subsequent builds. During this time the level itself may not undergo testing for several weeks. This schedule for testing the different sections of the game is maintained by a QA analyst. In addition to helping testers, it also helps the developers in planning their work as they know when certain parts of the game should be ready for testing.

5.5 Testing in different phases of game development

Schultz and Bryant (2011, p.99) present a basic structure for the phases in testing, that is not affected by the size of the game or the length of the production schedule. This structure divides testing to the following phases: pre-production, alpha, beta, gold, post-release.

Before the testing itself can begin, the team must decide how to track, report bugs and be aware of the different processes behind these actions. As an example, a bug tracking database can be used for this. The QA team must also know how to insert, comment, assign, change status of the bugs and also have a common understanding of the different bug fields, such as category, severity, and priority. (Chandler, 2008, p.366-371)

5.5.1 Pre-production

The pre-production phase involves planning tasks, assigning a lead tester, determining phase acceptance criteria, participation in game design reviews, setting up the defect tracking database, making drafts of test plans, designing tests and performing preliminary testing (Schultz and Bryant, 2011, p.101-109). During pre-production, almost as soon as the game development starts, the testing activity begins by the creation of plans for future tests. The test manager reviews the game design document, technical design document, the project schedule and starts to formulate a document that outlines how much testing resources, namely time and money, will be needed to test the game thoroughly for release. This document determines the scope of testing that the project will require. (Schultz and Bryant, 2011, p.100-103)

The term test manager used by Schultz and Bryant (2011) seems to match the definition of the QA lead / QA analyst used by Chandler (2008). Chandler (2008, p.361-363) mentions that the QA analyst should provide feedback on all the deliverables being generated, keep working on the test plan, test gameplay features, work with leads in managing the production pipeline and arrange testers to test possible prototypes or playable builds. The second activity involves selecting a lead tester and a primary tester / testers (vice lead tester). As the lead tester has a lot of influence over the entire testing cycle, this is an important decision. The lead tester should be able to keep the test team focused, productive and motivated, be able to recognize the role testing plays as part of the production process, able to gather and present information clearly and concisely and able to manage conflicts between people as they arise. After the selection of the lead tester, the primary tester / testers are appointed by either the test manager or the lead tester. (Schultz and Bryant, 2011, p.100-103)

The third activity during pre-production involves determining phase acceptance criteria.

In this phase, the lead tester establishes clear and unambiguous entry acceptance criteria for each phase of testing, which include the alpha, beta and gold versions of the game. Each test phase requires three elements: entry criteria, exit criteria and target date. Entry criteria define the set of tests that a build must pass before entering a given test phase. Exit criteria define the set of tests that a build must pass before completing a test phase. The target date defines the date a specific phase is planned to launch. With these criteria, conflicts can be avoided at a later phase, where different parties or parts of the organization might claim that the game is ready for e.g., beta, when this is not the case. After the criteria are defined, the test manager has to approve the criteria, after which they should be distributed to all senior members of the team. (Schultz and Bryant, 2011, p.104- 109)

The fourth activity involves participating in game design reviews. To have knowledge of the latest design changes and to advise the project manager of technical challenges and other testing complications, either the lead tester or the primary tester should participate regularly in design reviews. A change in design should be accommodated in the test plans. The fifth activity involves setting up the defect tracking database. The project manager and the lead tester should mutually agree on proper permission levels. These define which team member has the right to edit which field in the database. The sixth activity involves drafting test plans and defining tests. A test plan defines the team's goals along with the resources (people, time, money, tools, equipment) and methods used to achieve them. The types of tests to be performed, the individual test suites and matrices are defined in an overall test plan document. The final activity involves preliminary testing. In this testing, the test team is not testing a complete build of the game, but rather individual modules of the game. Until the development team submits the first alpha version candidate, defect testing remains as this so called modular testing. (Schultz and Bryant, 2011, p.104- 109)

According to Chandler (2008, p.361-363) only a few testers are typically needed for a few weeks at a time before the alpha phase, unless the game is very large and complex. However the QA lead / QA analyst should join the team as soon as possible, at least working part-time on the project, so that the work on the initial test plan can begin. In the later phases, the QA team follows the test plan to thoroughly check all areas of the game. As major milestones are met, more testers are added to the team and by the time the code base is frozen, the entire QA team is testing the game. (Chandler, 2008, p.361-363,444)

The test plan typically consists of pass/fail or checklist entries. E.g., a pass/fail test plan could contain entries whether the light switches turn on and off, or if the copy machine is functional in a certain level. Typical values include Pass/Fail/CNT(Can not test) and space for additional notes. A checklist test plan could contain e.g., possible character and weapon combinations. Typical values include the checklist and additional notes. Depending on how large the game is, the test plan could be hundreds of pages long. (Chandler, 2008, p.363-366,444-450)

5.5.2 Alpha

When reaching the alpha phase, the first alpha candidate has been created. The test team begins to to certify this version against the alpha criteria established in the planning

phase. The created test suites define the testing that the test team needs to perform. However during alpha testing the game design is still fine-tuned and features can be revised or scrapped. If this happens, the team needs to update the related tests accordingly. During alpha testing all modules of the game should be tested at least once and the team should establish performance (frame rate, load times, etc) related goals, thus defining performance related release targets for the remaining phases. (Schultz and Bryant, 2011, p.114-115). Typically, barring milestone builds, the QA department will not test every build against the entire test plan. Instead, certain sections of the test plan are rotated for each build they receive. The team may also receive specific instructions to test a certain portion of the game. In this case the team consults the test plan to decide how to test this portion of the game. The test plan is checked more thoroughly as the game gets closer to shipping. By the time the first gold master candidate version is ready the team should be running the entire test plan against the game. (Chandler, 2008, p.365-366)

5.5.3 Beta

When the first beta candidate has been reached, the team has mostly stopped creating new code, artwork and improving already existing features. Final game play testing may be performed with people from outside the design team, but only during the early stages of the beta phase. The focus is now on identifying and fixing the remaining bugs. Game play feedback and suggestions are now typically considered only for a possible patch, expansion or sequel. The team may recruit people from outside the team. (e.g., from the internet community) to perform bug reporting and load testing. The beta phase also includes an event called feature lock. When this is declared at some point during beta testing, all forms of issues related to game play and balance have been resolved and no new features or tweaking of old features is allowed. The test team continues to run the test suites against the new builds of the game as it is possible that a previously fixed defect creates new defects by mistake. (Schultz and Bryant, 2011, p.116-117)

At the end of beta the team will have to make a few critical decisions. Firstly, there might be an idea regarding a last minute feature. The team must decide whether it is worth implementing and testing a new feature, with the possibility of new defects and missing a possible deadline, or shipping the game as is. Secondly, the team may have to decide, whether to cut some content out of the game or not. This is an issue, when the team has noticed that some content in the game (e.g., a level, certain quests, a character) is not of similar quality as the rest of the game or not as fun. Cutting out the content may be a solution, but the game must still be tested to ensure it works properly around the missing content. This may also create the need to create new assets or rework on old assets to work around the missing content, thus creating the need to retest these also. Finally, the team must decide if some bugs are not going to be fixed for the release version or at all. The reason for this decision may be that there are too many technical risks related to fixing the defect, there is no more time to fix the defect or there is a workaround that the technical support team can supply to users who encounter the defect. Despite the reason, the senior members of the team should meet often to discuss the costs and benefits of fixing the bugs the development team wish to waver opposed to leaving them in the game. If the bug is left in the release version, it is still possible to decide to fix it in a future patch, dlc content or an expansion, if these are planned.

(Schultz and Bryant, 2011, p.117-119)

5.5.4 Gold

Once the game reaches the entry criteria for gold testing, a final version of the game called a release candidate / gold master candidate is built. When this is ready, all forms of development on the game is halted and the team starts to test the possibly final version of the game. This is performed by rerunning as much of the test suites as time permits and assigning some testers to break the game one final time. If this version of the game is accepted, it becomes eligible to be the gold master version sent to be manufactured. However, if any remaining bug is considered too severe to be allowed to stay in the game, the release candidate is rejected and a new release candidate version must be built that contains the required fixes. After this the release candidate testing starts from the beginning. This is repeated until a gold master version is finally produced. If the developed game is a PC game, a web based game or other open platform related game, after producing the gold master version, the game is ready to be manufactured and sold. In these cases the game's publisher or other financing entity can solely decide whether to release the product or not. However in the case of a console game (e.g., mobile phone, PS 3, XBOX 360) a phase called release certification must be passed. (Schultz and Bryant, 2011, p.120-121)

Each console manufacturer enforces a set of technical requirements, which the game must comply with. In order for the developers to fulfill these requirements, the manufacturer provides a checklist of each requirement. They may also provide tools that assist in checking the compliance. If these requirements are not met, the game may not receive an approval until these compliance errors are fixed. Because of this, these non-compliance bugs should be rated as high-priority bugs and be addressed quickly. (Chandler, 2008, p.370-371) In the release certification phase the team sends the release candidate version that passed the gold testing phase to the the platform manufacturer (e.g., Nintendo, Microsoft, Sony), who in turn will certify the game using certification testing. The testing consists of two phases that can happen sequentially or concurrently. In the standards phase the game is tested against a technical requirements checklist. In the functionality phase the game is tested for functionality and stability. When this testing is done, the platform manufacture's QA team issues a report of all the bugs found in the release candidate. These bugs are discussed between the representatives of the game's publisher and representatives of the platform manufacturer in order to reach a consensus on which bugs are fixed and which bugs can be wavered. At this point the team should consider to only fix the required bugs and leave the rest in. Every defect fixed contains a risk for more defects and causing further delays in the schedule. Once the platform manufacturer certifies the release candidate, the game is ready to be manufactured. However, it is possible that the entire development effort is not yet over. (Schultz and Bryant, 2011, p.121-122)

5.5.5 Post-production

The final phase of testing is called post-release testing. After the game has been released, testing will still have to be performed if the game receives post-release support by the developers. This phase allows the development team to review the list of waived

bugs and the previously abandoned design tweaks to add further polish to the game. However, even in this phase, each additional bug fix or a feature polish requires testing, so each improvement should still be planned accordingly. (Schultz and Bryant, 2011, p.122) Additional post-release support may include downloadable content (DLC) and expansion packs, which can also have an affect on the base game (even without the user purchasing them), creating the need for additional testing.

5.6 Methods of game testing

This chapter briefly describes game testing methods found in the literature. The focus is on describing actual testing techniques instead of testing disciplines, such as balance, compatibility, compliance and usability testing (Levy and Novak, 2010, p.58-69).

5.6.1 Ad hoc testing

Ad hoc testing is a form of free-form testing. The tester does not use a checklist to go over the game, but explores the game freely and looks for bugs. Gordon Walton, the Co-Director from Bioware Austin at the time, mentions that it is valuable for usability and gameplay validation, however it is not a substitute for methodical quality assurance practices. (Levy and Novak, 2010, p.144-145) Schultz and Bryant (2011, p.286) also mention that it is a natural complement to structured testing, but not a substitute for it.

Even doing thorough and careful test planning and design or using a very complex test suite, that has been carefully reviewed by test leads or the project manager, there can always be something the tester or the leads might have missed. (Schultz and Bryant, 2011, p.284) Ad hoc testing allows you to find bugs which could be missed using test plans and checklists, and allows the tester to test the game in the way the tester would normally play a game of that type. (Levy and Novak, 2010, p.144) (Schultz and Bryant, 2011, p.285). It can also allow a tester to explore an area of the game that the tester has not previously been working on, thus avoiding mistakes made due to testers becoming over familiar and desensitized to reoccurring defects (Schultz and Bryant, 2011, p.285).

Ad hoc testing has two main types, free testing and directed testing. In free testing the tester improvises tests on the fly. In directed testing the tester sets out to solve a specific problem or attempts to reproduce a bug. However, in both cases the testing itself should still be documented and be verifiable. The tester should also not act completely aimlessly, but have some sort of a test goal, e.g., whether the tester can get the player character stuck in the level or not, or figuring out if there is a limit to the number of units that can be built. The tester then attempts to reach this goal and records the defects that were found. Whether the actual goal of testing is achieved is less important. Directed testing expands on this by providing a specific question, e.g., if the tester can access all the characters, or attempts to find a reason for a hard to reproduce defect e.g., all the audio of the game drops out. (Schultz and Bryant, 2011, p.284 - 294)

5.6.2 Scripted manual testing

Manual testing can be planned to varying degree or be more ad hoc in nature. In scripted manual testing the testing is guided by scripts which were written in advance. Typically the scripts guide the tester in the input selection and define how the results of

the testing should be checked for correctness. The scripts can be very detailed or be more flexible in nature. As an example, Whittaker (2009, p.14-15) mentions that when Microsoft is testing games they often take a less-formal approach. In this case, a possible input could be "Interact with the mage". (Whittaker, 2009, p.14-15) In this situation the tester is free to decide what this testing might contain. The tester might talk to the mage, attack the mage, attempt to push the mage with the player character or do something else entirely. This means scripted testing can be either very rigid or a lot less formal (Whittaker, 2009, p.15).

This set of test inputs and the expected outputs of the software are typically bundled into a test case. In addition, information on the execution conditions required for running the test are typically found in a test case. (Burnstein, 2003, p.21-22) Schultz and Bryant (2011, p.132 – 134) define a test case as a single test that is performed to answer a single question. As an example, in the game *Minesweeper*, one test case could be to determine whether the value 999 is accepted as input in the Height box. In this case, the test case could consist of the following steps: “Choose Game > Options > Custom”, “Enter 999 in the Height box” and of the question “999 accepted as input?”. This test can either pass or fail, and in the case of failure, it would be a good idea to attach comments on how it failed. In its simplest form, a test suite is a series of such test cases. (Schultz and Bryant, 2011, p.132 – 134)

5.6.3 Exploratory testing

In exploratory testing, the testers can use the application they are testing in any way they want and generally explore the application without restraint. However, as the testing is being performed, the testers are producing documentation, which include test cases, test documentation and test results. The use of tools during testing is allowed. For example, Whittaker (2009, p.16) mentions that using screen capture and keystroke recording is ideal for recording the results of this type of testing. The main difference of this testing method compared to manual scripted testing is that the documentation of testing is not done beforehand in a test plan. It is also possible to combine exploratory testing with other forms of testing, e.g., by injecting variation to script-based manual testing using exploratory techniques. (Whittaker, 2009, p.14-20,180)

5.6.4 Automated testing

Automated testing means any form of testing which is performed automatically, typically by a computer, instead of a person performing this testing manually. Kaner and Pettichord (2001, p. 93) define it as writing software to test another piece of software. In practice it is possible to automate most of the performed testing, however this may end up only harming the development project. The choice on what to automate should be defined in an automation strategy. This strategy varies based on the testing requirements, the software product architecture and available staff skills. (Kaner and Pettichord, p.93-98) Levy and Novak (2010, p.146) specifically mention the ability of automated testing to go through a very large number of permutations in a game as useful, e.g., checking there are no conflicts between the multitude of player upgrade possibilities.

Schultz and Bryant (2011, p.355-380) mention the capture/playback automation

approach. In the capture phase of this method the player input is recorded and various points are designated in which screenshots are taken. Later, when the test is ran in the playback phase, the recorded inputs are given for the game and the captured screenshots are compared to the in-game situation. If the screenshot does not match the current in-game situation the test has failed. This means there likely exists a defect somewhere in the game. Another way to use this method is to use this recording in regular testing and thus being able to easily arrive to the spot which contains defects. However, if the game has random events, then the recorded input will not lead to the same outcome. This means, that in order for this testing to work the game must be able to be set to work in a deterministic and repeatable manner. (Schultz and Bryant, 2011, p.355-380)

5.6.5 Cleanroom testing

Cleanroom testing is mentioned by Schultz and Bryant (2011, p.229-260). This method was adapted from a software development practice called Cleanroom Software Engineering. The method attempts to produce tests in which the testers will test the game by emulating the way players typically play it. In order to be able to perform this type of testing information on usage probabilities must be collected. These include mode-based usage, player type usage and real-life usage. The probabilities can be based on data collected from players or on expectations on how the game will typically be played. (Schultz and Bryant, 2011, p.229-260)

Mode-based usage takes into account how the different modes in the game are played in different ways, e.g., single-player campaign or multiplayer. The player-type usage takes into account how people have different ways of playing a game. (Schultz and Bryant, 2011, p.229-260) For example, some players want to focus mostly on the story, others want to focus on combat, while others like to focus on role-playing their character. Real-life usage refers to capturing information on the in-game actions of players or NPCs (Schultz and Bryant, 2011, p.229-260).

After this type of information has been collected, the testers can emulate different player styles when testing the game. This type of testing can detect defects, where testing based on a balanced use of game features would not. (Schultz and Bryant, 2011, p.229-260)

5.6.6 Regression testing

The need for regression testing stems from the fact that when developers are fixing bugs, they can easily reintroduce old defects in to the code or generate completely new ones (Levy and Novak, 2010, p.148). If this happens, the code has regressed due to the changes that were introduced to the new build (Schultz and Bryant, 2011, p.331) Regression testing essentially looks for old bugs in the current code. (Levy and Novak, 2010, p.148) It is performed by deciding which tests are to be ran against each version of the game, but it can also involve the modification of existing tests or the creation of completely new tests. Efficient regression testing will minimize the numbers of tests ran, yet still test for newly introduced and remaining defects. (Schultz and Bryant, 2011, p.331,334) It is typically performed a few times in a week (Levy and Novak, 2010, p.148).

In order to properly perform regression testing, it is not enough to re-run tests that have failed in previous builds. This is because code that has not been changed in anyway could also have regressed, as it has been affected by changes made to other parts of code in the game. In other words, it is possible to have code that was unintentionally affected. A good way to perform regression testing for the new build, which claims to fix defects, is to re-run the tests that previously failed and any possible new tests that were created for those specific defects, regardless of the current test cycle. The remainder of the tests are then chosen on the basis of maintaining confidence in the quality of the remaining features. If no preexisting tests have been designed for new issues or tweaks that appear in the consequent releases, it is possible to design new tests that specifically target a specific bug or change. These new tests should also be ran in a cycle in order to confirm that the changes continue to work as intended. (Schultz and Bryant, 2011, p.331-334)

5.6.7 Focus group testing

Keith (2010, p.256-258) mentions focus group testing, but refers to it with the term play-testing. When using focus groups, potential consumers are recruited to test the game in an arranged session. It is typically organized and run by QA or usability specialists. The information from the sessions can be gathered by simply taking notes during the sessions and / or having discussions with the participants. However, a more scientific approach is also possible by recording progress metrics from the session, or by using surveys during the session. (Keith, 2010, p.256-258). In addition, it is possible to record these sessions. Typically the on-screen gameplay and faces of the participants are recorded and later this video material is analyzed and reports are created.

As benefits, Keith (2010, p.256-258) mentions that the participants can notice the flaws and shortcomings overlooked by the development team. The sessions also give an idea, what the current usability and challenge level of the game is. The participants can also give ideas on how the development team should prioritize the items in the development backlog. However, the participants should not be expected to provide the team new original ideas, this is the responsibility of the developers. (Keith, 2010, p.256-258)

As additional details Keith (2010, p.256 – 258) mentions that the recruited participants should represent the full scale of the player demographics and skill levels. He suggests to maintain a database of people and to ask the more valuable participants to participate in further focus group testing sessions. He also mentions, that it would be good for the developers to take part in these sessions, so they would see the ways the participants interact with their game. This is mentioned to have a chance to lead to improved interfaces and usability. Discussions between the developers and the participants is also encouraged. On publishing the results of the sessions, Keith mentions that a simple compilation of potential answers is enough and that the conclusions should be left either to the readers or for discussions. This is done so that the results are not 'overinterpreted' by QA. (Keith, 2010, p.256-258)

5.6.8 A/B testing

A/B testing involves dividing the users of a piece of software randomly between two different versions of the software. There is no limit to how different one version of the

software can be compared to the other version. It can be a completely overhauled version or a version with tiny improvements. The users taking part in this testing do not get to test both versions. They will only test the variant they received through the duration of this testing. After this testing is done, results can then be evaluated based on the participant feedback, through using statistical analysis or through other means. (Kohavi, et al., 2009)

When discussing mobile games Wagner (2012, p. 39,161) mentions that it is a good idea to give the (open) beta players slightly different versions of the game. This will test how these differences will affect player retention and engagement. For example, the variation could be a particular game play feature, such as a new level or gameplay mode. He also mentions that the mobile game company Wooga emphasized using A/B testing to test new game features on a select group of players to see how they improved the engagement, growth and monetization rates. (Wagner, 2012, p.39,161)

5.6.9 Psychophysiological measurements

Psychophysiological research in gaming focuses on measuring mostly the involuntary physiological processes that happen in the body. When using this method instead of observing the player or using a questionnaire, a lot of issues are avoided. The participant answering style, social desirability of an answer, different interpretations of wording, limit of participant memory or the bias of an observer do not affect the measurements made. The sensitivity of the devices is also an advantage, they can pick responses that would be impossible to detect by the human eye. (Kivikangas, et al., 2011)

The psychophysiological measurement methods include (Kivikangas, et al., 2011) :

Facial electromyography: The measurement of electrical activity in the facial muscles for assessing positive and negative emotional valence (negative emotions have negative valence, positive emotions have positive valence).

Skin conductance / **Galvanic skin response**: The measurement of the change in the ability of the skin to conduct electricity to measure emotional arousal (bodily activation).

Cardiac activity: The measurement of the heart and circulatory system to measure both valance, arousal and has also been used to measure attention, cognitive effort, stress and orientation reflex during media viewing.

EEG / MEG / fMRI: Electroencephalography (EEG) is the measurement of the brain's electrical activity to measure level of attention and amount of activity in the different regions of the brain. It has also been used to study the processing of visual emotional stimuli. Magnetoencephalography (MEG) maps brain activity by recording magnetic fields in the brain. Functional MRI (fMRI) maps brain activity by detecting associated changes in the blood flow of the brain.

Cortisol level: The measurement of the amount of cortisol from participant saliva to measure stress levels.

Respiration: The measurement of the respiration system to measure emotions or level of attention, can also provide additional control data when measuring cardiac activity.

Eye gaze tracking, pupil size: The measurement of the eyes to measure arousal,

cognitive effort, attention level and its direction.

Additional methods include (Kivikangas, et al., 2011):

Measurements via controller: Some emotional reactions can be detected by measuring the pressure applied by the player on the buttons of the controller. As game controllers evolve to provide more powerful input, so does the ability to model the player with the data provided by the controller.

Measurements via game engine: It is possible to measure continuous data from the game engine to analyze player behavior.

Studies have shown that traditional media (e.g., video, television, radio) can be measured using psychophysiological measures to index emotional, motivational, and cognitive responses. For digital games, similar evidence is slowly growing. (Kivikangas, et al., 2011)

5.6.10 Evaluating player enjoyment

Flow & GameFlow

Sweetser and Wyeth (2005) present a model for evaluating player enjoyment in games. This model, GameFlow, refines and extends the flow model presented by Csikszentmihalyi (1991) by using heuristics presented in game usability and user-experience literature. The criteria presented in the model could be used as a guideline for an expert review or as the basis for constructing other types of evaluations (e.g., player testing) (Sweetser and Wyeth, 2005).

The flow model itself was presented by Csikszentmihalyi (1991). Flow is an experience in which a person moves to an optimal state of experience when they are fully immersed in the task they are performing. According to Holt (2000) & Chen (2007), gamers value video games based on whether they provide a flow experience or not. The flow consists of eight different parts (Csikszentmihalyi, 1991, p.49)

1. A task that has a chance of being completed.
2. The ability to concentrate on the task.
3. This concentration is possible because the task has clear goals.
4. This concentration is possible because the task provides immediate feedback.
5. A person acts with a deep but effortless involvement that removes from awareness the worries and frustrations of everyday life.
6. The ability to exercise a sense of control over actions.
7. Concern for the self disappears, but the sense of self emerges stronger after the flow experience is over.
8. The sense of the duration of time is altered, hours can seem to pass by in minutes or minutes can feel like hours.

Chen (2007) presents the original flow channel concept of Csikszentmihalyi (1991, p.74), but from a video game centric viewpoint. In order to keep the players in the flow, the game should provide the right amount of challenge and ability, thus keeping the players inside the flow zone. If the game provides little challenge for the player, the player will quickly lose interest and stop playing. If the game is too challenging it will

generate anxiety in the player. The goal is to reach a balance in which the game is not too easy or too challenging, to avoid negative player emotions, and reach a so called safe zone. However, not all players share the same flow zone, in fact, different players, from novice to hardcore, have different flow zones that depend on player ability and game challenge. To counter this, the game should offer adaptive choices inside the game and embed choices to the core mechanics of the game, thus allowing different players to reach their own flow zone.

According to Cowley, et al. (2006) flow is the best-fit experience description construct as it has remarkable similarity to the act of playing. The GameFlow model is based on the elements of flow and the evidence of flow experiences in games. The model consists of eight elements and the related criteria (Sweetser and Wyeth, 2005) :

Element	**Criteria**
Concentration. Games should require concentration and the player should be able to concentrate on the game.	- games should provide a lot of stimuli from different sources - games must provide stimuli that are worth attending to - games should quickly grab the players' attention and maintain their focus throughout the game - players shouldn't be burdened with tasks that don't feel important - games should have a high workload, while still being appropriate for the players' perceptual, cognitive, and memory limits - players should not be distracted from tasks that they want or need to concentrate on
Challenge. Games should be sufficiently challenging and match the player's skill level.	- challenges in games must match the players' skill levels - games should provide different levels of challenge for different players - the level of challenge should increase as the player progresses through the game and increases their skill level - games should provide new challenges at an appropriate pace
Player Skills. Games must support player skill development and mastery.	- players should be able to start playing the game without reading the manual - learning the game should not be boring, but be part of the fun - games should include online help so players don't need to exit the game - players should be taught to play the game through tutorials or initial levels that feel like playing the game - games should increase the players' skills at an appropriate pace as they progress through the game - players should be rewarded appropriately for their effort and skill development - game interfaces and mechanics should be easy to learn and use

Control. Players should feel a sense of control over their actions in the game.	- players should feel a sense of control over their characters or units and their movements and interactions in the game world - players should feel a sense of control over the game interface and input devices - players should feel a sense of control over the game shell (starting, stopping, saving, etc.) - players should not be able to make errors that are detrimental to the game and should be supported in recovering from errors - players should feel a sense of control and impact onto the game world (like their actions matter and they are shaping the game world) - players should feel a sense of control over the actions that they take and the strategies that they use and that they are free to play the game the way that they want (not simply discovering actions and strategies planned by the game developers)
Clear Goals. Games should provide the player with clear goals at appropriate times.	- overriding goals should be clear and presented early - intermediate goals should be clear and presented at appropriate times
Feedback. Players must receive appropriate feedback at appropriate times.	- players should receive feedback on progress toward their goals - players should receive immediate feedback on their actions - players should always know their status or score
Immersion. Players should experience deep but effortless involvement in the game.	- players should become less aware of their surroundings - players should become less self-aware and less worried about everyday life or self - players should experience an altered sense of time - players should feel emotionally involved in the game - players should feel viscerally involved in the game
Social Interaction. Games should support and create opportunities for social interaction.	- games should support competition and cooperation between players - games should support social interaction between players (chat, etc.) - games should support social communities inside and outside the game

Table 1. Game flow criteria for player enjoyment in games. (Sweetser and Wyeth, 2005)

For each of the elements the model includes criteria that can be used to design and evaluate games with respect to player enjoyment. Sweetser and Wyeth (2005) use the GameFlow model on two different games by using expert reviews to validate the GameFlow criteria and to expose weaknesses, ambiguities and issues found. As a result, it was discovered that some of the criteria are more suited to specific genres and some were not applicable, in this case, to the strategy (RTS) games under inspection. Also, some of the criteria were found to be difficult to evaluate by mere expert evaluation and would instead require player testing. Firstly, to determine whether a game suits players of different skill levels, players of different skill levels need to be used. Secondly, to measure the pacing of a game several players of different skill level need to be used. Finally, to measure immersion, there is a need to measure the players while they are playing the game. This is not something that can be evaluated through self-report. (Sweetser and Wyeth, 2005)

For each criterion a numeric value from zero to five was also assigned for both games that indicates the extent the game supports a criterion. The final values assigned resulted in a rating (by taking the average of the values) that seemed to be in line with the rating taken from professional reviews (48% vs 61% and 96% vs 94%). The criteria also helped to identify the reasons why the other game title under inspection was less enjoyable for the player, e.g., the challenge level of one of the titles was particularly low. It was also discovered that it is easier to identify what is wrong in a game compared to identifying what is done well. The evaluation of the lower scoring title revealed some of the well designed features of the other title. (Sweetser and Wyeth, 2005)

Jegers (2007a, 2007b) has explored the game flow in pervasive gaming. A technological-focused definition for pervasive games characterize these games by three properties. Firstly, they enable the player to play at any given time, in all possible settings. Secondly, they integrate the virtual and the physical world in a meaningful way and finally, they are driven by social interaction. Jegers (2007a) provides a Pervasive GameFlow model by making adaptations to the GameFlow model proposed by Sweetser and Wyeth (2005). A later study by Jegers (2009) concluded that seven of the eight elements should be considered important and one (social interaction) would require further study. In an assessment of the applicability of the model provided by Bleumers, et al. (2010), despite raising issues, the model allowed a systematic and elaborate review of both technology-based and non-technology based pervasive games.

Game Experience Questionnaire & Game Engagement Questionnaire

Both Ijsselsteijn, et al. (2007) and Brockmyer, et al. (2009) have presented the idea to develop a questionnaire that is related to the user's engagement in a game and the overall experience of the gamer. Ijsselsteijn, et al. (2007) present the Game Experience Questionnaire (GEQ). It attempts to produce an assessment of the player's perceived experience of a game by assessing the different aspects of the subjective experience of playing the game. This would be done by characterizing and measuring the user's experience with the game experience questionnaire. However, the paper itself doesn't describe the model in detail, instead mentioning that the model is currently in the process of being developed further.

Brockmyer, et al. (2009) present the Game Engagement Questionnaire (shares the same acronym GEQ), which attempts to measure the level of engagement an individual has while playing a game. This model is presented in more detail. However, something to note is that the main motivation of the paper is not to explore the gaming experience, but to identify players that are susceptible to video game violence and to study the impact of playing violent video games. (Norman, 2013) (Brockmyer, et al., 2009).

The questionnaire presented by Brockmyer, et al. (2009) contains 19 items, these items include e.g., statements such as “I feel scared”, “I can't tell I'm getting tired” or “The game feels real”. These statements are categorized to the following categories (and thus attempt to measure): absorption, flow, presence, immersion. Psychological absorption is a term that describes the total engagement in the present experience and just like flow, it induces an altered state of consciousness. Presence means having a normal state of consciousness, but having the person feel the experience of being inside a virtual environment. The statements were answered with a “No”, “Maybe”, and “Yes” answer.

From these answers, the GEQ scores are calculated for each person, where a person with a low score has likely experienced low engagement, and people with higher scores likely experienced deeper levels of engagement. (Brockmyer, et al., 2009).

6 RESEARCH METHOD AND PROCESS

The research presented in this thesis is qualitative. The first stage of the research process was to examine the field of game testing by collecting and reading literature of software development, game development, software testing and game testing. Using this literature, most of the theory section was written before the interviews were conducted. This was done in order to have enough knowledge of the subject area to be able to create the interview questions and to be able to gather as much relevant information as possible during the interviews.

Interviews were chosen as the data collection method. This method was chosen, as it seemed to be able provide the best answers for the research questions of this thesis. As an example, if surveys were chosen, the questions of the survey would have been very open ended, as the exact nature of the testing performed by the companies was not clear beforehand. As another example, participating in the testing sessions would have been difficult as the thesis covers testing during the entire production process resulting in having to take part in multiple sessions.

The interview questions were created by using an iterative process. The researcher made the first version of the questions, and they were refined together with the instructor. Minor alterations were also made during the first couple of interviews.

A total of seven Finnish gaming companies were interviewed and the results of these interviews were analyzed. The interviews were semi-structured so that a set of specific questions were built, however, the interviewer was free to ask additional questions during the interview. The questions were open-ended and they typically required an in-depth and a lengthy response. The thesis did not attempt to test or create a specific theory or hypothesis. Instead, it attempted to map the subject area by using the data collected from the interviews, to point out good practices in game testing, to point out possible problem areas of game testing and to point out possible areas for further research. (Metsämuuronen, 2001, p.9-10,13-15,38-48) The companies and the people involved were promised to be kept anonymous.

After the interviews were conducted the recorded interviews were first transcribed into text. After this, this text was divided into several main themes. A single piece of text could belong to a single or several themes. Finally, the themes that were seen as being the most important were analyzed and written into sub-chapters for chapter 7 .

7 RESULTS OF THE INTERVIEWS

The results of the interviews are reported in this chapter.

7.1 The goals and objectives of testing

Many of the interviewees stated that the goal of testing is to ensure that the game is good. A good game is described as a game, that is of high-level and good quality, the best possible for all potential users, as bug free as possible, is as playable as possible already during development, provides a seamless experience or a problem free experience of the game and is fun.

The responsibility of the testers for quality control and acceptance testing was also brought forth as a goal. The testers in this situation must at all times have an idea, what the current state of the game is, if it is possible to, e.g., publish some type of a demo version of the game or to release a live version of the product. In some cases this even leads to testing taking responsibility of the organization of the entire project. Testers were capable of saying what items were still needed, in order to move forward to the next phase in the project. This same goal was also described by the ability of testing to provide visibility to the overall situation of quality, to the nature of individual problems and their scope.

In addition the communication within the development team was mentioned as an objective of testing, which leads to being able to catch possible problems as early as possible. It was also seen as helpful, if the developers were able to catch as many problems as possible by themselves.

For a bit of a differing goal, one company mentioned supporting other departments as a goal of testing. This enables the game content creators, e.g., the coders to fully focus on production and fixes, so that they don't have to spend resources on the testing themselves.

Differing opinions were presented in regards to seeing fun factor as one of the goals in testing. Each of the interviewees saw fun factor as very important, if not even as the most important aspect of the game, but not everyone saw it as belonging to the responsibilities of testing. A few of the interviewees emphasized, that ensuring the fun factor of a game belongs more to the game designers and producers than testers. In this case the role of the testers was more in functional testing. However, in each of the interviewed companies the opinions of the testers were heard, in regards to how fun the game felt, even if this was not seen as being one of the goals of testing.

All of the quotes taken from the interviews were translated from Finnish to English.

“[...] Fun factor, is one. We have seen that it is absolutely the most important, one can say, that it is the most important thing in testing. To figure it (the fun factor) out and if we are on the right track, with the entire game [...]”

“[...] I would say more that, ensuring the usability, […] the viewpoint of the testers was always that the producer knows and has to know, if it (the game) is fun. And the designer, game designer, must take responsibility that the game is fun. The role of the testers was more analytical and more pedantic, they focused more on the details whereas the producer and the designer, they focused on the opinions of the users. Through polls they analyzed what the users currently liked in the world, what kind of music, fashion, etc, and looked for inspiration, from there, the fun factor was found. So I would see that it (the fun factor) was not in such a significant role, because of these roles.”

> **"Yes, the fun factor was not part of the testers repertoire. It did belong in the tools of the game designers and especially in the tools of the producers, this was the way it was divided".**

Many of the interviewees also mentioned, that the goals of testing change during the different phases of game development. Finding the fun factor as early as possible was generally seen as important, because the later problems are found in the fun factor aspect of the game, the more resources have been spent on the development of the game. If the development has already advanced much, it can be too late to make radical, big changes to the game. This leads to the fact, that the evaluation of the fun factor of the game should start already during the prototype or pre-production phase of game development.

Several of the interviewees mentioned using focus groups when testing the fun factor of the game, but some of them also mentioned, that the game should already be in the alpha phase at this point. This seems to lead to a contradiction, in which the fun factor of the game should be tested with outside personnel, but making drastic changes to the game based on this data in the alpha phase may not be possible. Some of the interviewees also mentioned that if the game is not in a ready enough state, the testers might mention, that it is too difficult to evaluate the game at such an early phase. Related to this, several interviewees mentioned, that in the early phases of game development, the internal members of the development team evaluate the fun factor of the game, and the outside testers were typically not brought in until during the alpha or beta phases of game development.

> **"[...] different phases, pre-production, production, alpha. Already in the pre-production the fun factor should already emerge, and the testing of it […] so that it is fun, because again, if we go, for example to the beta and start testing the fun factor for the first time […] and then notice that no, the game is not fun, we have already spent a lot of money for nothing at that point [...]"**

> **"[...] The focus changes a little bit at that point, at no point is the fun factor forgotten about, but, if it (the fun factor) has been present in the pre-production, production phase, and the alpha has mostly maintained this, then the fun factor will not suddenly break or disappear in these phases [...]"**

> **"[...] the opinions were kind of at that point too late already, because the components were ready and the process had gone far, so I would see, that we never thought to seriously reverse something solely for the reason that the fun factor is not good enough on some new component. Instead these were perhaps moved to future development ideas, in the way that, okay, we will take this along […], because it is fun for some people."**

In the beginning and early phases of development keeping the game's prototype and the pre-production version in a playable state was also seen as a goal of testing. This allows the development team to constantly experiment with and play the game, so that the received feedback from the game can be maximized. One interviewee also emphasized, that by using this feedback and by planning together one attempts to prevent certain mistakes and problems from occurring in the first place.

For the pre-production and production phase, a few of the interviewees also emphasized finding all problems, that would possibly disrupt the production, and helping in fixing these problems as goals of testing.

Even though the game itself can be more difficult to test during the early phases of development, one of the interviewees mentioned, that if the game engine is ready enough, it is possible for the testing to focus more on testing the game engine itself, to

breaking it on purpose and to testing the extremities of the engine. In this way, during the final phases of production, testing can focus more on the technical operation of the game mechanics and the technical functionality of the game features.

The further the development of the game advances the more testing moves to observing concrete problems. The tightened quality criteria lead to a situation where smaller and smaller problems are intervened with. When the development reaches alpha and beta phase, the focus of testing moves from testing new features to polishing the game, bug removal and getting the game to a stable state. In the alpha phase the focus was typically to find all the critical bugs, where as in the beta phase, the focus was more on polishing, however also ensuring, that everything is still working as intended. Many of the interviewees mentioned, that only after reaching the alpha phase most of the testers started their work, finding bugs, filling out the bug database and ensuring, that the bugs that were marked as being fixed were actually fixed. On an even more detailed level, one of the interviewees mentioned, that it is important to know what is being tested at a given time. E.g., if in the build that is undergoing testing, a map is still unfinished, the tester wastes time by reporting e.g., possible texture problems of that map. This emphasizes the importance of communication between the testers and the game developers.

In the release candidate phase, the main focus was to ensure that in addition to all the fixed problems, no more new issues were created, and to ensure, that the game is ready to be moved forward to the next phase.

The main goal of the gold phase was typically to get through the different requirements the platforms themselves require of a game, that are monitored by the manufacturer of a device. It was preferable to pass these in the first attempt, as the time spent for these checks delayed the release, and in the case of not passing successfully the first time, the consequent rounds of checking could also require a fee.

7.2 Evaluation and testing of the fun factor in a game

The majority of the interviewees saw the evaluation and testing of the fun of the game as one of the goals of testing, but a few saw it as being more of the responsibility of the designers and producers. In addition a few of the interviewees also emphasized a shared responsibility of the fun factor of the game between all team members. However, everyone agreed that the fun factor of the game is one of the more important, if not even the most important feature of the game.

Many of the interviewees mentioned how the fun factor is built from all the different elements of the game and is a sum of all of them. Regardless of this, the interviewees mentioned game features, which have an especially big influence on the fun factor of the game. Firstly, the progression in the game should be good and consistent, so that that the game experience doesn't contain so called bumps. The player should feel, that she continuously advances in the game. The player should never get stuck in the game for too long. Secondly, the game should be fair towards the player. Generally this means, that the player only suffers from the mistakes the player makes during gameplay, not from issues that the player is unable to control. As an example, if the game requires too fast reaction speed, or the game balance has problems because of certain overpowered abilities, the player might start to get frustrated with the game. Thirdly, the learning

curve of the game should not be too steep. With this, the game should take into account both more experienced players and novice players, and it cannot give too much information or new gameplay to the player in a short amount of time. The quality of the tutorials provided by the game also play an important part in this. It was also mentioned, that it is important to follow already established practices, which the players have gotten used to by now, e.g., moving units with the right mouse button in an RTS or control groups in an RTS. Fourthly the experience provided by the game should be as seamless as possible. This means that the player should not have big problems with any parts of the game during the game experience. This contains the possible bugs in the game, as well as relative quality, e.g., the frame rate of the game should be high enough or the game should not have too long loading times.

A few of the interviewees mentioned the importance to understand that fun is not objective in all cases. Therefore it is completely possible, that a feature in a game is fun for a certain player group, and not so fun for others. A good example of this could be the game difficulty levels, where the different viewpoints can vary greatly. Regarding this, it was also mentioned, how different target audiences can play the same game and enjoy the experience, but for different reasons. As an example, in a strategy game, one group of players can think of min maximizing the production as the most fun, where as another group of players focuses on being the largest production company in the game. Because of this, it is important to be aware of the large already established player groups, that can have certain expectations for the game in advance.

Hamari and Tuunanen (2014) have investigated the subject of different player types by analyzing the previous works done of player topologies, and then synthesizing them according to common and repeating factors. The end result is seven dimensions, that are related to the motivations of play or orientation of the player. The dimensions are: Gaming intensity and skill, Achievement, Exploration, Sociability, Domination, Immersion and In-game demographics. The in-game demographics dimension refers to things such as player class or player progression. All of the seven dimensions can be used to typify players by using behavioural measurement. (Hamari and Tuunanen, 2014)

Many of the interviewees mentioned, how the ability to evaluate the fun factor of the game disappears quickly, when the team is constantly working on the game. This leads to a situation where someone who works on the game a lot can have a difficult time to evaluate, how fun the game is. Despite this, evaluating the fun factor of the game using internal development team members was the second most used method in testing the fun factor of a game. Related to this, a few of the interviewees mentioned, that the game is made primarily for the players, and not for the development team itself. In other words, the development team and the target group for which the game is made can have a different taste regarding games, and can have differing opinions of what makes a good game. On the other hand, one of the interviewees mentioned, that most members of the development team are gamers themselves. If the team itself doesn't find the game fun at all, then they know, that they are going in the wrong direction.

Evaluating the fun factor of a game had two methods that were used above all others, the use of so called focus groups and the feedback received from the development team itself. Each of the companies used focus group testing at least to some degree. Typically the focus group participants were acquired from different places, which included the

friend circle, the neighborhood and also outside organizations, e.g., the focus groups provided by a game publisher. The focus groups themselves were not divided to different subgroups, but during the recruitment phase, the person might have been asked a few questions related to the background of the person, and to check, whether the person was fit to be a part of the focus group.

A typical focus group session was a type of a blind test, where the participants were mostly free to test and try the game. The sessions typically had at least one member from the development team, that took notes during the session. The session itself could also have been recorded on a video, in which case, both the footage from the game and the face of the tester was recorded simultaneously. But it was also possible that this was not done, as going through this material was seen as laborious. A few of the interviewees also mentioned the use of questionnaires in addition to an interview, that was used to map the game experience. In addition to this, one of the companies mentioned the ability to present questionnaires inside the game, at which point the game would stop, and the tester could evaluate the experience at the given moment. The interviewees did not specify the structure of the questionnaires, and it falls a bit outside the scope of this thesis. For this reason it cannot be said, if they used e.g., game flow criteria (Sweetser and Wyeth, 2005) or the game engagement questionnaire (Brockmyer, et al., 2009) to aid in their design.

> **"[...] perhaps testing this (the fun factor) is more emphasized on the focus group tests, for example, where we bring outsiders. Also we have a closed alpha test group, in which we still have active players, we collect feedback from them [...]"**
>
> **" […] We have usability testing, which is under design, our focus group testing department, and their job is to figure out the fun factor, and also to examine the usability with personnel from outside the company."**
>
> **"[...] the focus group and user research testing is part of the game testing package. Perhaps it is the other part of the equation, if the other side is the functional testing. It is an important part nowadays (testing the fun factor), and almost all game companies are performing it on some level."**

Six out of the seven interviewees mentioned also the importance of the development team itself in evaluating and testing the fun factor of the game. Several of the interviewees mentioned, that no specific sessions or time was reserved for this, so that the approach was not as systematic. It also did not have any specific approach, the team members tested the game in an ad hoc way. One of the interviewees mentioned, that when a member from the development team needed to get a general picture on how far the development currently is, then at the same time, this was a good opportunity to evaluate how fun the game is. Another mentioned, that as the developers have to constantly look at and try out the game, it is possible to evaluate the fun factor in this way as part of the normal development cycle.

7.3 Game quality factors

The interviewees mentioned the following game quality factors as being the most important. One interviewee mentioned that these quality factors differ per project, and didn't expand on them further.

Game quality factor	Number of interviewees mentioning the factor
Responsiveness	6
Fun factor	5
Game stability / Player will not get permanently stuck	4
Usability	3
Controllability	2
Server uptime	2
Non monotonousness	1
Latency	1
Game fairness	1
Ease of use of payments	1

Table 2. Game quality factors seen as being the most important.

Responsiveness, fun factor, and the game stability were most often mentioned. Responsiveness means how responsive the game feels and it is related to game optimization, input lag and frame rate. In addition to good game responsiveness, one interview also mentioned the load times of a game and how they need to be kept short as well. This was done not only for the user experience, but some console manufacturers have guidelines regarding the load times. Of the fun factor, a few of the interviewees mentioned again that every aspect of the game can have an effect on the fun factor, and as such, the fun factor includes the other game quality factors inside it as well. Of game stability the interviewees mentioned that the game should not under any circumstances crash. In addition, a few of the interviewees also mentioned, that the player should never get permanently stuck in the game. This was seen as a similar event, where the end result is that the player must either restart the game or gaming machine.

Usability was mentioned in relation to an online store, where the user journey, the path a user goes from registering to the service and making purchases, must be made simple, and that all separate components of the UI should support this. It was also mentioned in the context of usability testing, which included e.g., inspecting responsiveness, checking whether players liked a certain feature or not. Controllability means how good the controls in the game feel and does controlling the character feel good and natural for the player. Having a responsive game is prerequisite for this. Server uptime is an online only issue, however, it was mentioned that if the game servers are down, it might have a direct effect on the cash flow in the case of not receiving payments from the game, and it can also have an effect later on, if a player decides to quit because of the server

downtime.

Finally, individual mentions were made which stated, that the game should not be monotonous, that is, it should not repeat itself too much, that the online lag in the game should not be too high, that the game should not be unfair towards the player, and that payments related to the game should be easy to use.

The game quality factors presented here are a bit different than the ones presented in the examined literature. A likely cause for this is that the ones presented here are game quality factors that are more related to testing. The most overlap seems to be on the ones presented by Schultz and Bryant (2011, p.70,128) with the mentioning of the quality of the game interface (UI and controller input), the quality of game performance and with the mentioning that the feedback from games should be just random enough to be unpredictable (thus avoiding monotony).

7.4 The organization of testing

The companies that were interviewed ranged from small to large, from a company with roughly 20 people, with a few regular testers, working on a single project, to large teams with roughly 20 testers, and having multiple projects at the same time. The length of the projects ranged from a few months to several years, and to projects that were constantly being worked on.

Testing was organized differently in the firms. Firm A had a game producer, which supervised the rest of the development team. As the size of the firm was small, the few testers themselves did not answer to any specific test lead, but to the producer herself. However, it was also mentioned that testing is the responsibility of each team member, even though at the same time acknowledging, that it is common to be blind to the bugs that oneself has created.

Firm B had a separate test team consisting of roughly ten people with a test lead. The main task of the test lead was to design test cases that could contain from 50 up to 100 steps. These were then handed out to the testers. However, over time, when the testers learned to understand the game better, they started writing test plans and test cases themselves, and thus the importance of the test lead role started to diminish. Depending on the development process used, the testers were either together in a separate team by themselves, or in the case of agile development, situated so that one tester in a single development team was responsible of the testing for that team. The reasoning for this was that there was just enough work for a single tester in such a team, but not enough for any additional testers.

Company C was similar in structure to A. It had a single test lead, also working as a tester. The lead worked closely with the rest of the development team and in addition organized all additional testing. The lead also worked with a few outsourced testers that were testing the game regularly. Depending on which parts of the game were changed, the testers were told to focus on specific parts, e.g., multiplayer or the way items react to physics.

Company D was a larger company, that was developing multiple projects simultaneously. The organization was such that a QA director was at the top of the organization, and was responsible for all of the products. One step below, a QA manager

was the supervisor of the testing units and the test leads. The main responsibilities of the QA manager included things such as the management of resources, test plans and maintaining the overall vision of quality. The test leads were spread out over individual game projects, in such a way, that a single test lead was responsible of a single product. Individual testers were then taken from the testing department and placed to work under a test lead on a specific product. In addition, there was also a QA coordinator role that was placed between the lead testers and the manager, acting as a supervisor of multiple test leads. The QA coordinator was responsible of a range of products. In addition to the aforementioned teams, there were also teams that were not tied to a specific game project, but were specialized to test a specific part of a game or a more technical component.

Company E is a case, where majority of the testing has been outsourced to a publisher. The company had only a few testers and even these people had additional work responsibilities. In addition, outside firms were used for testing. It was not possible to get information of the organization of these outsourcing companies.

Company F had a testing department of its own consisting of roughly ten people and it was ran by a test lead. The responsibilities were divided in such a way, that the testing department was responsible of technical game testing, and issues related to usability and the fun factor of the game were mainly the responsibility of the designers, even tough the testers did provide some input on these matters as well. The testing department finds and reports problems to other departments, e.g., design and art department, and checks if the reported problems were actually fixed. If there were no incoming requests on what to test, the department maintained a list of game features that needed testing. Testers were also encouraged for independent action and to find new areas of the game to test by themselves.

Company G had similar testing organization to E. At the time, the company had one QA lead, whose job was to keep in touch with the test organizations under the publisher and other outside testing organizations. It was not possible to get information of the organization of these outsourcing companies.

Each of the interviewed companies also confirmed, that the developers themselves did testing as well. The repeating theme in these answers was that everyone is responsible for the work they output, to a certain degree. One interviewee specified, that it is almost crucial that this is done, so that the development can stay smooth and obvious blunders can be avoided.

> **“Yes, we all are testing and playing the game, testing and reporting problems, and just like I said, […] everyone should be responsible for the quality of the work they create, to a certain level. This has been the company culture.”**
>
> **“[...] But, mainly it is the responsibility of everyone to also view the quality of the work they create, and the correctness, the functionality.”**
>
> **“[...] absolutely yes, it is, perhaps even in a larger role, the goal is absolutely that, the developers test the basic case themselves, and then the testing can support with the regression-problems, the exception cases or the case where something else breaks which is not directly related to the feature under development. Absolutely this is pretty much necessary, so that the development would be smooth, also it has the unit tests to support this, the unit tests which are automatically ran in a build. These also support in the same way, so that these kind of mistakes can be removed.”**

7.4.1 Outsourcing of testing

Companies A and F used outsourcing in testing only to cover specific demands the company that owns the gaming platform expects from developers, in other words, platform compliance testing. These lists can be quite large and demanding, and thus, using an outside company to ensure that they are correct, can save the time and money that would be spent if the actual submission process would return a non-passing result.

Company B used outsourcing on a large scale in localization testing. This mostly involved checking that the translations were correct and that the text was positioned correctly using the new language. The reason to do this was that the company itself did not have the language skills that were needed. Some mobile projects had their testing also fully outsourced. It was mentioned that generally outsourcing mobile game projects is cheaper than using the company testers to do it.

For Company C, the reason for having outsourced testers was mainly a cost related issue. In addition, the ease of scaling the testing up or down on demand was mentioned.

Company D mentioned that outsourcing has been used for localization and compliance testing. These were seen as testing that requires special skills, however, at the same time, they are not needed all the time. The ease of scaling the testing was mentioned again.

For company E, the reason for outsourcing most of the testing, was an organizational issue. Less testers are needed during the early phases of the project and more are needed during the later phases. Thus keeping the correct amount of staff was seen difficult. In other words, it is another mentioning of the possibility to scale outsourced testing more easily. However, as a downside, it was also mentioned that all the information that can be gained by being in a testing room yourself, e.g., seeing the reaction of the testers can be missing in this case. Company G provided almost exactly the same reasoning.

> **"[...] but, in any case, we nevertheless must […] have more testers in the company before […] than in the beginning. So it is difficult to time the project, so that we could maintain exactly the right amount of personnel to test the game full-time, so that they would have enough meaningful tasks to perform. So, it is easier to do this in a way where we can on-demand use the outside parties when performing the testing, so that, it frees resources for us. [...]"**
>
> **"Well, for us it has been this way because, we have long game development processes, and testing requires so much manpower in the end phases, that we would have no way to keep that amount of people on a payroll the entire time. They wouldn't have work for 80% of the time. So, in away, because of this, we have to make sure that we have the (test) leads, that are in charge, and the bulk of the work is, for the most part, done outside."**

7.5 Required knowledge, skills and training in testing

7.5.1 Required knowledge and skills

Five out of the seven interviewed companies stated that no special skills are required in the testing. However, a lot of skills and attributes that are good to possess when being involved with testing were mentioned. One company mentioned having an eye for details, being systematic, analytical and having a desire to confirm things. They described it as a mindset in which some persons fit, and others not as well. Another mentioned good communication skills, so that a tester is able to give a detailed

description of the problems, and reasons why something is either good or bad. Programming skills were also seen as a bonus, and as a requirement for top of the line testers, as this gives the ability to give an estimation, on what could possibly be causing the problem. Patience was seen as a virtue, when performing repeated tests daily. Another company stated a requirement for being enthusiastic for the gaming industry, as everything else can be trained and special skills are seen as a bonus.

> **"Our demands for a basic tester, is an enthusiasm towards the gaming industry, and everything else is trained. It is a plus, if the tester has special skills [...]"**

> **" […] It requires a specific eye for details, and a type of systematic approach and a desire to make sure of things. It is a kind of a mindset, in which, certain kinds of people are well suited for and others are not suited at all. And a kind of, fast tempo creative work style, this is in my opinion not well suited for straight forward, manual black box testing, instead, one must be very systematic and have an analytical way of examining things. [...]"**

One interviewee specifically mentioned, that a person acting in the role of a QA lead, will require technical understanding and good social skills. For those acting purely in the role of a tester, much less training is needed. The testers are taught, e.g, what to test, what should specifically be taken into account daily with the new builds, however no specific training sessions were arranged.

One interviewee mentioned that they see a person who is focused on usability as more useful than a person that has gone through testing courses, traditional testing certificates or learned different process models of testing. The reasoning for this was that, a game producer can't define everything that makes a game good, as even the producer isn't fully aware of this. If in this case the tester can say e.g., that the steps that are required for payment are too long, or that the selected colors aren't working, then this is seen as more valuable than the aforementioned more literature based knowledge.

Out of the two interviewees that mentioned special skill requirements, the first one mentioned that there are several specialized areas which are more technical in nature, and then there is the so called, traditional game testing. These skills included having knowledge of different gaming platforms, of different tools related to testing, having the experience of being involved in alpha and beta testing beforehand, having knowledge of server side technology, e.g., transfer speeds and the requirement for an international software testing qualification certification (ISTQB). This certificate is attempted to be provided for all testers. The certificate involves studying, an end test and gives out a formal certificate when completed. The second interviewee emphasized the need to understand how the game works, and how the platform under which the game had been built works. The reasoning was that this helps in doing more informed testing, so that the person has a hunch what typically generates bugs, where the bugs may be, how to make a bug reappear and where additional bugs are typically located. This was stated to be helping as it is never possible to test all possible combinations of the different elements of the game.

7.5.2 Training and development of testers

Five out of the seven companies stated that they have no special training related to testing. However, most of the interviewees also stated that people learn the job by doing, that the job will train the person. A few also mentioned that the testers must know

the bug testing tools and bug databases, but this was seen as not needing any special training.

> **"[...] I would not say that it requires any form of special training, instead black box testing requires a correct personality and a kind of an unyielding systematic approach, also a bit of pedantry. And when you have these kind of qualities, the job will teach (the tester). [...]"**

> **"So far we have not used any training. […], but so far we have not needed (training), it has mostly come through work experience. Of course, it is good to understand the game development process, because it is important that the right things are tested at the right time. [...]"**

Out of the two companies which mentioned special training, the first one mentioned the testing certificate, which was already presented in the previous section. The second company mentions that it takes roughly two weeks of training in which one learns the basics of testing, knows the testing pipeline of the company, and after this time the person can start being somewhat productive. The entire training is stated as taking roughly 6 months, and happens along with the person performing game testing. However, the interviewee, like many others, also emphasize that regarding the subject matter, one learns by doing. The training is stated to involve knowing how to use the in-house tools (e.g., game editor, how to open maps, how to place enemies), how to write the reports, how to identify different problems and what the correct place to forward a specific bug (e.g., not sending an artist a defect related to game code) is. It also involves learning to understand how the game works internally, some inspection of game components, understanding how the components are made and what problems they might cause.

7.6 Planning and documentation of testing

Three of the interviewed companies A, C and F stated that they plan testing minimally. Company A mentioned that when outside testers were used, it needed to be clear on what they should be testing. However this sort of outside testing was done in low amounts. As the most important test situation was a sprint review organized between the team each two weeks, where the team members presented the features they had implemented, not a lot of planning was needed. Specific documentation was not used, expect when a build needed to be delivered somewhere, in which case a list of things to check was used. In addition, when console platform related requirements were being tested, there was a list of requirements to be fulfilled, in which case testing could be planned according to the implemented requirements of the list. Company C used agile development, did practically no documentation of testing and even attempted to keep feature documentation as brief as possible. The interviewee mentioned that if a document gets too big for a feature, then perhaps the feature itself is a little too complicated. The testing happened in daily meets and ad hoc meetings. Company F mentions that if there are no requests from other departments, the testing department maintains a list of features to be tested. For patches similar lists are maintained, but no other documentation of test planning is used. Some reasons were provided for this such as that the game features themselves aren't documented in such detail, the high amount of game features make documentation difficult, and generally writing these kind of documents was seen as bothersome.

> **"We are very much an agile development company all around, we practically don't produce any kind of documents regarding testing and we attempt to keep the feature documentation as very compact. Perhaps already, some kind of an alarm bell is ringing, if we must produce very complicated**

> **documentation on some feature. Perhaps in this case, the functionality is a little bit too complicated [...]"**
>
> **"The planning of testing is actually one of our current areas to develop. And, game testing is so far, based more on the gut feeling of the test lead. Also the coding, design or art department can take part in the test planning, e.g., asking for specific things to be tested. And in practice, the test team answers these request responsively. When there is a request we immediately go to observe the issue. Otherwise we have a list, or we attempt to maintain a list of game features, which is constantly looked through. If there is a patch for the games, we have a list of things to check. But, more detailed plans than this are not done. We attempt to strive for testers to have initiative and self-directedness, in a way that the testers are encouraged to find testable areas of the game on their own. This is purely for the reason that one person cannot remember all the different game features or come up with all the different test scenarios. And for this reason, the more initiative the tester has, the better [...]"**

In Company B the testing was planned as the product was being built. Then when the test planning was ready, the product typically started to be ready for testing. Then in a quick pace the product was tested so that it could be determined whether it goes back to development or if it is accepted. The planning started as the producer first thinks of new components to add to the game. From this, the lead and the testers started to think how the new feature could be tested and what the steps might be. The documents for this were the design document, test plans, and test cases. The design documentation also contained the requirement analysis for the new component. The test cases were written step-by-step, and the test steps and the design document were linked together. They were written based on the requirements analysis. The test plans themselves contained e.g., 10 test cases. A single test case contained the individual steps to be taken, e.g., press a button, a shopping window should open. A single test case contained typically from 50 up to 100 of these individual steps. In the beginning, these were mostly things that would be tested manually, and black box testing was used. Later on these included automated testing and load testing, splitting roughly 50/50 between automatic and manual testing. The test lead wrote the test cases, but over time the developers started writing them as well.

As the company moved to agile development, a lot of documentation was dropped, and only the test cases with their individual steps were kept as documentation. Additional documentation included a quality handbook and so called result documentation. The quality handbook contained information such as how to build a test plan from a game design document and in what part of the development cycle should the test cases be built. The goal was that by reading this handbook, one could work smoothly in the test team, and it also gave the other employees an idea what the testers are doing and what input they require. The handbook became redundant over time as well. The result documentation contained information on what worked well in testing at the time, what needed improvements, and an evaluation whether the testing is sufficient enough to move on. Over time, this documentation also changed to mail exchange, and finally to verbal discussions in the agile spirit.

Company D stated that testing was being planned continuously. On a higher level this included things such as resourcing, the structure of teams and the roles of different people, the selection of testing methods and whether to adapt testing to a new system or change the system itself. On a more detailed level, it included test plans and test cases. The test plans contained testing actions divided to different milestones, which explained what testing activities are needed, the size of teams, a resourcing overview which included information such as how many people are needed and required special skills.

The test cases contained comprehensive instructions e.g., on what is the expected functionality of a feature on a step-by-step basis. The resourcing was done on a management level and the test plan was done in collaboration with the test lead. The plan is then approved higher in the organization e.g., by a QA manager. The process itself is iterative, so that it is possible to observe whether something needs to be changed or new things added, as the development itself progresses.

Company E mentioned the use of master test plans and test cases. The plan contains information such as the schedule, the different milestones, evaluations on how many people are needed for testing on different times, list of possible collaboration partners for testing, and when specific testing methods, e.g., blind testing are to be used. It also contains estimations about the time when certain modules of the game are ready enough so that testing can begin on them. Test cases with test steps are mentioned to be used in small amounts, but only typically when regression testing is used, and for features that are seen as critical.

Company G had a similarity to E. The interviewee mentioned that the product defines what kind of testing is performed, what could be the possible collaboration partner, and what kind of expertise is needed. A master test plan was mentioned, which contains information on when to test, how much time is used for testing, and similar issues. Test cases are used in differing amounts depending on the project. In addition, when the console platform certification requirements are being built, specific test cases were present, which defined what was to be tested, what the response should be, and for what reason. E.g., for a console title such a list could contain roughly up to a few hundred instances.

Of the firms that gave a rough estimate on how much of testing is based on plans and the relative amount of time used for test planning compared to testing overall, company D mentions that most of the testing is based on plans. It typically requires a surprise, a change in the current situation, when 'ad hoc planning' may be needed. However, the interviewee further specified that this does not mean that each work day is fully planned out. Company E mentions a rough estimate of 40 – 50 %, and also mentions that not a lot of time is used on the planning itself. Company F gives an estimate of 50%. The time spent on the planning was stated as being roughly two to three hours a week. Company G mentioned roughly 30%, but did not have an answer for the amount of time spent on planning. Some of the interviewees saw giving such estimations as difficult, and did not give out any specific estimates or numbers.

7.7 Testing in different phases of game development

This chapter gives a brief synopsis on how the interviewed companies approached testing in different phases of the development process, and the methods used in testing.

7.7.1 Company A

Company A developed games using an agile game development approach. During the pre-production phase, the company used the internal team in figuring out a concept, evaluating this concept with the help of the team, and then starting to work on the concept when the team was satisfied with it.

During the production phase, testing was mostly performed by the internal team. This internal testing was mostly performed during sprint reviews, roughly once in two weeks, in which the team observed what had been produced so far and tested these features. Between these sprint reviews, the game designer and the producer were observing and testing the game. E.g., if a developer mentioned during a daily scrum that a feature was ready, the game designer tested whether it matched the vision of the game. However, if during the production phase the team was unsure of a new feature introduced to the game, focus testing was performed by bringing people from outside the company to demo and test these features. These sessions were also recorded, and the fun factor / player enjoyment could be evaluated from observing the faces and reactions of the participants. In addition, automated testing was performed in the form of unit tests and by running predefined input sequences on the game. It was also possible to follow the current state of the game engine and the game itself, informing the team on the current state of the build, and giving an indication whether a feature was broken or not.

In the alpha phase, the focus from developing new features switched to polishing existing features and bug removal. During this phase and onward, the company hired full-time testers to perform testing. These testers kept the bug database up to date, by populating the database with newly found bugs, and confirming that bugs reported as being fixed were actually fixed. The gold phase consisted of compliance testing, with the goal of passing the requirements demanded by the console manufacturers. When this internal compliance testing was complete, the game was passed on to the console manufacturer to be tested for compliance.

7.7.2 Company B

Company B used to use the waterfall model and the V-model, and during this time, the team were producing test plans in the early phases in accordance to the V-model. The different phases were seen as partially overlapping one another. However, later on the team started using agile software development methods. With this switch, the more strict processes disappeared and were replaced by the goal of the testers to support the rest of the team in some form throughout the development.

The methods used in testing consisted of test cases performed by testers, automated testing, load testing, focus testing with possible video recording, ad hoc testing, A/B testing and localization testing. The test cases were initially planned by the test lead, but later this was the responsibility of single testers. They were initially based on requirement analysis documents, but later on the tester also had to independently figure out what the component should look like, and produce the test plans accordingly. This black box testing in the form of test cases later moved to the end phases of development, when the end product was possible to be tested as a whole, much like in a system testing phase in traditional testing. In automated testing, the tests were attached to the build system, which fetched the latest version of the code from the version control system, built it, and then ran the tests, signaling if something was broken. Due to this taking up to half an hour or more, the testers on occasion manually chose tests to be ran from this set. These tests were implemented in early phases of development. Already before the first prototype of a product was available, the automation tester was asking how it would generally operate, and started to prepare coding the tests for it. When the first prototype was released, the automation tester would build automation test scripts

for it. These scripts were worked on to the end of the products release, and were maintained even after the release. The load testing was performed in a specific load testing environment, with a custom load testing tool, which was testing a single instance of the product. The goal was to make the game show signs of malfunction or failing under the traffic, thus figuring its ability to cope with network traffic. This was performed in the very end phases of development, as it was seen that, the entire product and its features should be placed under this load. In a tdd spirit, the developers also used unit testing when writing code, however, it was up to the developer to decide how they handled this. This wasn't seen as a tool for the testers, but for the developers instead. Localization testing was performed both in-house, when the language talent was available, and outsourced.

Usability testing did not have a specific cycle or time when it was performed. Instead, it was performed when the team felt that the product had changed enough to undergo more testing. This was performed mostly in the form of focus group testing and in the less systematic form of one team member showing the game to other members of the team and discussing about it. In a common focus group session, a tester with knowledge of usability testing directed the group, and occasionally took notes, e.g., recording that the progress in the game was halted to a specific spot. Sometimes the sessions were taped, however due to time requirements, they were seldom analyzed. Occasionally this testing was also outsourced. A/B testing was also used on occasion in a form, where, a small portion (5-10%) of the user base received e.g., a new feature. The goal of the usability testing was to get confirmation whether the direction the product was being developed was the correct one, e.g., if the chosen themes were correct, if the color space was correct, or whether the characters were nice. The fun factor was seen to be more the responsibility of the designers and the producers, who would examine this by observing the behavior of the users. For a rough estimate on the amount of time reserved between the different methods, 50% was reserved for manual black box testing, 35% for automated testing, 10% for load testing and 5% for usability testing.

7.7.3 Company C

Company C developed the game using agile development methods. The interviewee had not been present in the current project during the pre-production phase, however, it was emphasized that it is very important to have a working build as fast as possible, even in this phase. This build was attempted to be kept in a playable form, and to quickly react if there was something that blocks testing the functionality of this build. This approach was mentioned as avoiding the end crunch situation when using the waterfall model, where there can be tons of bugs and issues to fix in late phases of the project. As the project approaches the release phase, smaller and smaller problems are being caught and worked on, to improve the quality of the end product.

In the day to day development, the team in charge of developing the game tested the basic functionality in a local environment by themselves and with the help of unit tests. Testing was then supporting the development with regression testing, automated testing, focus testing and load testing. Regression testing involved looking for exception cases and ensuring that the newly developed functionality did not break other parts of the product that was not directly linked to this new functionality. An important goal of this testing was to keep the project in a playable form, and functional at all times. Once a

day, a QA build was built, which contained the commits for the entire day in a single package. The test lead spent most of the time testing the former build, and the latter build was sent to the rest of the testers. A mostly ad-hoc approach was seen as best for this type of testing, as different kinds of checklists were seen as funneling people into only focusing at the contents of the list, and not e.g., whether the list omits important features or whether the game actually even works properly. Sometimes directions on what the testing should focus on was inserted in the build notes of the daily QA build. E.g., if the physics engine for the game was modified, the focus could be on all object related physics in the game. Additional reasons for this approach was that as the testers were outsourced, this solution was seen as being easily scalable, as more people can be introduced when needed, and it was seen as more affordable.

For automated testing, unit tests were ran on the server side for each commit, and if these passed, the build was deployed to a development server. However, for the game client, there was no automated testing. Thus functional testing, testing on a user interface level, was performed manually. The interviewee mentioned that increasing automation for functional testing would be a good idea for a game that is meant to be maintained online for years, but is undesirable at this stage in the project. Load testing was also under development for the project, having hundreds to up to tens of thousands of game clients sending network traffic to the server, by the way of clients using the in-game functionality. E.g., visiting different places in a world or completing tasks in a location.

At times, focus group sessions were arranged, which involved bringing people from outside the company to play the game. The participants were handpicked beforehand, and as the game targets the casual market, people of a more casual background were targeted. This testing began during the production phase, but increased in the alpha and beta phases. In a basic session, the participants were given tasks to complete, e.g., to play specific missions, or play a specific amount of the opening tutorial. The session was observed, and feedback was also collected via questionnaires. Questionnaires were preferred over recording video, as processing the video was seen as a laborious task. The sessions were at a time arranged even once per week, but at a minimum, there were several sessions in a single milestone. This was seen as a rather heavy process internally, so a few times these sessions have also been arranged by third-parties, where the focus had been on verifying the concept of the game or the fun factor of the game and to consider the ways to monetize the game. From these sessions, video was also recorded, and on one occasion, eye tracking data from users was also collected. As the game reached the alpha phase, a closed alpha test group was also formed. This originally consisted of roughly hundred handpicked individuals, but has expanded to roughly two hundred people in the form of invitations. The group receives a weekly build of the game to test, and the feedback coming from these users is collected.

At the time of the interview, the game was also nearing the soft-launch phase, in which, the game is released on a single market. Some typical countries for this type of a release can be e.g., Canada or Australia. The plan is the to collect and analyze data, and to decide, whether the game should still be iterated upon on this market, or whether the results are good enough to launch the game on a global level. As the game had yet to be soft-launched, the post-release testing was not yet fully planned. This was seen as being dependent on how successful the game is after the global launch, thus affecting the

amounts of resources and prioritization testing is allocated.

7.7.4 Company D

Company D saw testing before the alpha phase being more about giving feedback, and overall getting oneself acquainted with the product. Then when the alpha stage is reached, the testing attempts to ascertain that the game core and its features work as a whole, and attempts to provide visibility to their current status. In the beta phase, when the game features and content have been locked, actions are taken to ensure that everything is still working as intended, to ensure that the game balance is correct, and that the end product is polished. The main methods used in testing were exploratory testing, automated testing (unit testing, load testing, smoke testing, soak testing, stress testing), compatibility testing, performance testing, usability testing and balance testing.

Exploratory testing is seen as a very essential part of testing and is used throughout a project. The testers are not given specific test cases or steps to follow, but instead a general instruction to focus on a specific area, and otherwise, the tester is given free hands to search for possible problems. For automated testing, smoke testing consisted of running automated short length test runs. It attempted to target the wide range of devices able to run the game, thus these test covered many different devices. In contrast, soak testing was targeting devices on a smaller scale, targeting a smaller amount of devises. This involved running tests over a long time period, e.g., in the game menu, or in gameplay, possibly over night, to test that e.g., the game does not use too much memory, or that it does not contain a memory leak. In this example, this could be observed via the actual effect this soak test has on the game, or by measuring the current memory usage. In stress testing, the game was placed under heavy load using different methods, so that the stability of the game could be verified. The goal of load testing was to ensure that the environment stays stable in all situations which can realistically be assumed to happen. Unit testing was also performed by the developers, however, there were no specific demands or guidelines for it, e.g., on how much code the unit tests should cover.

The goal of compatibility testing was to ensure that all the devices that were within the spectrum of devices planned to be supported, were working as intended. This included different devices, the version of the devices and the different operating systems of the devices. This was automated to some degree, while some platforms required more manual work. The goal of performance testing was to give visibility to the current status of performance on the different devices, and also to see how this performance varies in relation to time played, e.g., does the performance degrade over time. This was mostly done by observing the fps counter, memory usage, and also, the general responsiveness of the game.

Usability was tested by the way of a dedicated team that was constantly observing it, by the feedback coming from focus groups, and through regular testing, where the experience and skills of testers gave insight, on what is good usability and what is not. When the focus groups were selected, the goal was that the people chosen would produce a comprehensive sample, which would represent the different demographics of players. Typically the participants of the session were given the same goals to achieve. The views on game balance were gathered from both developers and testers. The goal

even on balance related issues was to bring as much visibility to the current situation as possible, by collecting as many of the subjective views from the team as possible. E.g., A person who has designed a level may have a difficult time saying whether it is easy or hard to complete, but when you have the opinion of 20 different people, you have a lot more visibility to the current difficulty level.

Some of the testing was also at times outsourced. This was often related to testing which needed some special skills, such as localization or compliance testing. If the testing could have been performed in-house, the typical reason for the outsourcing was the scaling benefits it provided.

7.7.5 Company E

Company E developed their games using agile development methods. The general approach to development was that instead of producing very large design documents, the team had an idea of what could make a good game, and went on from there, producing fast prototypes. The design and code were modified along the way, as the fun factor and the core of the game were being searched and built, thus bringing visibility to these in the early phases of development. However, the development phases presented in the literature were seen as fitting the actual development process quite well. During the pre-production and production phases, testing was typically focused on testing specific modules in the game, as the game was still not in a state, where it can be played from start to finish. So the testing is performed on individual modules and features of the game, focusing on what is seen as most important for the game at the time. When the alpha phase is reached, testing is ramped up and expands from testing individual modules to testing larger parts of the game, e.g., testing how the plot is carried in the game. Additional testing methods are taken into use, including methods which involve using people from outside the company. For the most part, the testing has been dependent on outside parties, e.g., as in one instance, a publisher. In practice, the publisher performed testing, and then sent the reports to the company, where the QA lead analyzed the results. The QA lead then informed the rest of the team on the bugs and problems that were found, and assigned them priorities, defining which of the issues should be worked on at the time. The methods used in testing consisted of automated testing (unit testing), ad hoc testing, focus groups, blind testing, forum testing, regression testing and outsourcing.

The developers writing code and developing assets performed testing, but only in small amounts. This was done by testing the code during production, producing unit tests and ad hoc testing. Unit testing was seen as laborious and was performed in small amounts, and it did not have specific guidelines on e.g., how much code should be covered by them. In ad hoc testing, a developer played the game freely, and made observations on the fun factor of the game. As the team consisted of active players that enjoyed playing games, a situation where the team would start to have a general consensus that the game is not that fun, would imply that the game has issues to be fixed. In general, not a lot of time was used for test planning. Test cases planned and executed by testers were used to some degree, mostly for regression testing, and for game features that were seen as critical.

Starting from the alpha phase, blind testing and focus group testing were performed. In

blind testing, a person who had no prior knowledge of the game was given the game to play, without being given any instructions on how the game works, or other similar advice. The player was then observed, taking note, on how well the player was able to learn the basic game concepts, whether the player learned the things the game was meant to teach and how much fun the player had. This type of testing revealed whether the tutorial actually teaches the knowledge needed to survive in the game, how playable the game generally is and how intuitive the game is to learn and play. It also gives insight of whether a section in a game should have a tutorial if it is currently lacking one. In comparison, in the focus group tests, the testers were given more specific instructions, e.g., to test a specific level, or with a specific difficulty level. Sometimes these groups were also kept together for a longer time, e.g., working with the same group for a month. In this way the group saw how the game was developing, and it was possible to get information on whether the group thought the game was going in the right direction or not. However, both in blind testing and focus testing, the participants were otherwise free to test the game as they liked. When the people for the focus groups were selected, the aim was to get players from a wide spectrum, e.g., those who had played a lot and those who had played little to none, those who had prior knowledge of game development and those who did not, and so on. This was done for the differing viewpoints these people provide on the game. However, in general, the people participating in the testing were given the same tasks to complete in a session.

The publisher arranged forum based testing, where the users were gathered from a forum. This entailed opening up specific beta forums for the game, enlisting the users of the forums to this beta forum, and sharing a beta version of the game between these members. In turn, the users report the bugs they found. The feedback in general was seen as being valuable, and typically it was given by players already familiar with the genre. This type of testing was generally performed in the beta phase, as it was discovered that sharing the game in a too early phase leads to diminished feedback, as the players view the game as being too unfinished.

Parts of the testing were also often outsourced to outside testing companies. These companies were used for receiving a wide range of feedback, depending on the current need, from gathering user experiences, receiving information on the usability of the UI, to performing focus group testing. Typically, these testing companies were given instructions on what to test, and what kind of feedback was expected, e.g., an extensive report or simple user polls. The main reason for this outsourcing was the scalability of this approach, which enables keeping a desired amount of testers at hand at all times, and freeing resources and personnel for the company.

7.7.6 Company F

Company F developed games using agile development methods. The general development process followed the one presented in literature by Keith (2010, p.131) for the most part, however there was a specific prototype phase before pre-production. In the prototype phase, a small prototype was made for the game using the in-house engine, where the main goal was to create and find the fun factor of the game. When this was found, and the team was satisfied with the prototype, the game moved on to the pre-production phase, and more people were brought to work on the project, including artists, designers and coders. The designers started drafting the features which would

support the core fun factor of the game, the artists started to design the graphical outlook of the game and the implementation of it, and coders started to schedule and plan, e.g., whether the game engine needed modifications, and what kind of modifications. In the production phase, production continued based on the plans created in the pre-production phase. In an ideal scenario, the planned features of the game were locked in this phase, but in practice, it was typical that the game still received new features, even new engine features, in this phase.

The alpha phase was defined as beginning when the game was for the first time in a playable form. At a minimum, this included having a game character that could progress a small amount in the game. Testing was planned to be performed throughout a game's production, and this alpha phase was when the testers began working on the game, with the testing lasting all the way to the gold master. For the most part, as the game was still not in such a playable form, the testing typically focused on the features of the game engine itself, e.g., the networking functionality of the game engine.

A game was defined to be in beta, when the game was for the first time playable from start to finish. In this phase, the testing started to focus more on the game features and mechanics. At this stage, for the most part, the game engine should already be confirmed to be working as intended. Typically at some time during the beta phase of production, work on the next project was also started, from the prototype phase. As the game starts approaching the release phase, the priority list of bugs was observed, and it was decided, what items should still be fixed. Meanwhile, testing was also focused on testing compatibility with different hardware combinations. When the game reaches gold, it has been feature complete a long time, and has been extensively tested and polished. After release, the games typically also received heavy after release support, which included bug fixes and adding additional content to the games. The test department followed the customer feedback, observed the problems the customers were having, examined these problems, and reported the found, confirmed bugs. Typically, the testing department did not split personnel between different projects, thus the testers worked together, focusing on one project at a time. The methods used in testing included ad hoc testing, automated testing, focus groups, performance testing, network related testing, compliance testing and a small amount of outsourcing of testing.

The other members of the team besides testers performed testing as well. The testers were mostly responsible for the technical side of the game, and the designers were more responsible for the fun factor and the usability of the game. The testers did give input to these matters as well, but in general, the design team was mostly responsible, on what they chose to include in the game.

The performed testing was mostly ad hoc, black box type of testing in nature. In the typical scenario, a list of game features was maintained, from which the test team selected features to be tested. The testing department aimed for the testers to have their own initiative, and self-directedness, in the sense that the testers were encouraged to find areas from the game to test on their own. Thus the testers were given a considerable amount of freedom, and responsibility of their own actions. The QA lead could then intervene, and direct the testing, if for some reason, this self-direction failed. The stated reason for this approach was that the QA lead as a single person can't keep track of all the game features, or to come up with test scenarios for all of them, and thus having this

kind of initiative and self-directedness was seen a strong point in a tester. At times, the code, art or design departments may have made direct requests to test some parts or features in the game, and these requests were given priority. These requests were quickly checked, after which the regular testing continued. For the most part, the list of features to be tested was the extent of documentation related to test planning. More detailed documentation had been attempted in the past but this was seen as difficult, laborious and taking time away from testing itself, as a game had numerous amount of features, and in some instances, the game features themselves may have not been documented on a very detailed level. As example scenarios, the testers could be told in the morning that platform specific versions of the game have been reported having input related problems, and the testers are told to focus on that or that the focus for that day is on the multiplayer features of the game. The QA lead observes the situation, and takes action, if problems emerge.

Automated testing was performed by the coding team by writing unit tests, having a test machine that opened game levels and checked whether they loaded successfully, and automated checks that confirmed that the latest build can be ran successfully. However, the testing department did not work on test automation, and thus the interviewee was not at the time of the interview aware how the coding team handled this type of testing on a more detailed level. The engine also had in-built logging capabilities, which the testers could activate on demand. Thus when there were issues with the game, the testers could record data about the state of the game, when the problem was occurring. The coders could then later examine this data for possible clues on what could be causing the problem.

Focus group testing was performed in a specific department, but this was more the responsibility of the design team, rather than the test team. The design team had the most responsibility for finding the fun factor and usability of the game. In a typical focus group session, there were a few participants per session, who received the game to test. These sessions were recorded on video, and an observer also participated in the session, helping the participants advance if they got completely stuck. The participants were not specifically selected, but their information, e.g., how much the participant generally played, was used when the results were analyzed. From this analysis, the focus testers wrote a report, which they sent on to the designers.

The performance of the game was also at times specifically tested. The main methods for this was observing the amount of draw calls the engine made, the amount of polygons on screen, the fps counter and evaluating whether the camera was too far from the game area, which would create slowdown. The fps was also evaluated in regular testing by simply observing how the game performed. The team also had PC's of variable performance, to test the game performance on.

The networking code of the engine and game was also at times specifically focused on. This included testing the game on a poor internet connection, compatibility testing with different software and hardware and observing the pings of the testers in game. When testing the game on a poor connection, the game was hosted with a subpar internet connection, and other testers joined this game, in a sense, creating the worst possible scenario. The testers then observed, what effects this had on the game, and whether the game was still in a playable form. Devices with different hardware and software

combinations were also tested, to ensure they can communicate properly with each other. In general when testing the networking, the focus is on whether the game is synced between players (objects are displayed in the same location between players), and that the ping of the testers in the game did not go over desired levels.

Testing was outsourced only for compliance testing, to meet the device manufacturers requirements for releasing a title to their console. This involved working with a publisher, so that the game could be released on specific console platforms.

7.7.7 Company G

Company G developed games using agile development methods. The phases of game development presented in the literature were mostly seen as conforming to the current development, however the interviewee mentioned an additional prototyping and conceptualization phase before the pre-production phase. During this additional phase, different kind of versions of the game are conceptualized and then prototyped. Thus when the development reaches the pre-production phase, the team has already chosen, by evaluating the prototypes, the version of the game they wish to develop further. Depending on the size of the project, the pre-production and production phases can also be divided to several phases, and the larger the project becomes, the more difficult it is to distinguish the boundary between the pre-production and production phases. This happens because usually not all areas of development go from pre-production to production at the same day, e.g., the content creation systems can go to the production phase earlier than other systems, or the code itself can already switch to production phase, but the story, including environments and levels, could still be in pre-production.

In the prototype / conceptualization phase, different ideas and gameplay mechanics are thought of and tried out to figure out, on a rather high conceptual level, the type of game that the team will take to further development. In the pre-production phase, the game has already received a general direction from the previous phase, but now, this goes under further refinement, and the entire game package is defined. With this, the team is aware of the tasks that need to be done to create the game, so that the game can be built based upon the original concept idea and prototypes. The production phase typically entails creating the different code based systems, graphics and sound that the game requires, and also includes producing vertical and horizontal slices of the game. The vertical slice contains a cross-section of all the game's systems in a small segment of the game, demonstrating the current progress across all of the different components of the game. In a horizontal slice, the game is playable from start to the end, in a very rough form, and this can be used as a framework for production. During the pre-production and production phases, the main focus of testing was to find and solve the problems that were causing disturbances with the production process itself. In other words, the testing supported the production process, so that it flowed smoothly. E.g., this could include testing the in-house production tools, finding crash bugs in the game and paying more attention to the production processes in general.

In the alpha phase, all the important systems for the game should already exist in-game, and only some of the content should be in an unfinished state or missing. When this stage is reached, the testing switches focus more to the end product, and to the end user experience of the product. Thus usability related testing, such as user research, focus

group testing, and fun factor evaluation starts in the alpha phase, and they are performed more and more, the closer the project gets to being finished.

In the beta phase, in practice all of the systems and content should already be in place, and the testing focuses more on functional testing. This includes finding bugs, and ensuring old ones are fixed, thus improving and polishing existing features. As the game starts getting closer to the release, it is tested until the related measurements are within acceptable margin, and the team's general confidence for the game is high enough. Depending on the devices the game is released on, it may still undergo the device manufacturer's certification testing. The company also supported some of the titles after release, producing patches, adding additional content to games, and producing DLC for games.

The testing methods used included ad hoc testing, automated testing (unit, continuous integration tool, static code analysis), focus group testing, user research, the use of questionnaires, performance testing, and outsourcing of testing to various degrees depending on the project. In addition to the testers, the development team also tested the game and reported the defects found, as part of regular development. It was part of the company culture, that everyone was responsible for the quality of their own work. This type of testing was done mostly by using an ad hoc approach, and if needed, the people responsible for the different departments coordinated a team's internal feedback of the game. In general, test cases were planned to some degree, and depending on the project, were also created by an outsourced partner. For a rough estimate, 30% of the testing was based on planned testing, and the rest was mostly ad hoc type of testing.

Automated testing was performed in the form of unit tests, using a continuous integration related tool to perform simple scripting, and using static code analysis. However, the scope of automated testing in general wasn't seen as that large, as this was seen easily becoming too laborious. The coders had implemented unit tests in some specific tools, and some of the code modules. However, the creation of unit tests did not have specific guidelines, e.g., on code coverage. The interviewee mentioned, that writing unit tests in game development can be harder to do than in more traditional software development, as a lot of the code is related to interactive, visual content. In practice, the amount of unit tests was attempted to be increased for new code, but unit tests were not retroactively created for older code. In static code analysis, the code was checked by using automated tools, which attempted to find possible memory leaks, buffer overflows and similar defects and point them out in the code itself.

Performance testing was performed with the help of an inbuilt logging system in the game. When this was performed, the testers tested the game, while having the game logging this data. After the testing was finished, this data could then be uploaded to a database and visualized. This logging system was used to test other possible issues with the game as well, e.g., in figuring out the common spots for the player to die, helping in figuring out possible difficulty spikes in the game. For memory related issues, there was also a system that attempted to load all the game's content, with the aim of passing this test. This will confirm that the memory in the console or PC doesn't run out, and thus, that the code frees the memory properly.

Usability testing was performed in the form of focus group testing, questionnaires, user research and arranging open betas. In general, the design team of the company was

more in charge of the usability testing, whereas the testers were in charge of the functional testing. In a typical focus group session, the participants were given the game to play, capturing the sessions also on video and observing the reactions of the participants. The participants were at times also asked to speak as they were playing the game, to describe what the participant is currently feeling, or how they currently feel about the game experience. At times, questionnaires were also used, after the session, asking how the participants felt or how they would rate the game currently, asking this on different areas of the game. In-game questionnaires had also been created, in which a question popped up during gameplay, asking whether the participant liked a certain feature of the game or what their thoughts were on it. When the participants were selected, some division was made along the lines of what type of games the participants typically played and what kind of experiences they wanted or what they wanted to play. The difference to user research was that user research was more focused in nature, attempting to test specific features or functionality in the game. It was trying to validate, whether the game is good, how it could be improved, what the players currently think of it and whether the players understand what they need to do in the game. On some limited occasions, the user research also involved the collection and analysis of eye tracking data from the participants. On some projects, typically related to mobile devices, open betas were also arranged. When these were arranged, the feedback came from the end users, and most of it came via automatic analytics. This involved saving data to a database and then visualizing it, including information such as what the players are doing in the game, how well the monetization of the game is working, where players are often quitting the game and how often players return back to the game. Depending on the project, varying degrees of the usability testing was also outsourced, e.g., to be arranged by the publisher of the game.

7.7.8 A summary of methods used in testing

The following table gives a brief summary of the methods that were used when testing was performed.

Used method	Number of companies mentioning the use of
Ad hoc testing	7
Automated testing	7
Unit testing	7
Load testing	3
Smoke testing	1
Soak testing	1
Focus group testing	7
Recording video of user behaviour, and analyzing the material	6
Collecting eye-tracking data from users	2
Using questionnaires	2
Planned manual testing, testers executing test cases	4
Compliance testing, testing the requirements made by the console manufacturer. E.g., TCR	3
Balance testing	2
Compatibility testing	2
Localization testing	2
A/B testing	1
Blind testing	1
Exploratory testing	1
Forum testing	1
Soft launching a product	1

Table 3. Methods used in testing.

7.8 Testing sufficiency and criteria for ending testing

Most of the interviewees mentioned that testing does not end until the project reaches the gold master state and is released. The companies that had online aspects in the game mentioned monitoring the game after a release, e.g., following the comments coming from players. One interviewee specifically mentioned supporting the older titles in the

form of receiving and checking support tickets for the older games. The testers could then quickly check the possible issues related to the tickets, however the games were no longer under active testing. A few companies also mentioned game updates, which instead of just fixing and tweaking existing issues could also add more content to the game. These patches were seen as projects of their own, which generally were done with a smaller team and fewer amount of testers.

The method for monitoring the progress of testing was the same for each of the interviewees, i.e., the bug tracker and specifically, the list of bugs and the severity of these bugs. When looking at the rate at which new bugs are found and the fix rate, evaluations can be made on the advancement and schedule of the project. The typical approach was such that for each milestone a certain amount of bugs were chosen to be worked on, with the overall goal of reducing the total amount of bugs. The typical end goal was to have zero bugs of very high and high priority classes. Bugs preventing player progress and other bugs of a critical level were not allowed in any situation. In practice, a compromise was made on what was deemed to be acceptable for the release, which may have even included high priority bugs.

> **"In practice we use […] named tracker software, in both for controlling the backlog and the bugs, and I would see that that is closest to measuring in what state the testing currently is. So in a way it tells us the current problems we have and their level, in the bug tracker […], but perhaps as an indicator it tells us how many open problems we are aware of at the moment, informing us of the current status of testing [...]"**

> **"It was, the opinion of the tester, that now we are at the level, where there are no more overly critical bugs in the current release, so we (the development team) can go forward. So part of the testers role changed to being an 'acceptance' tester. The tester operated as the right hand of the product, informing when the product was good enough, so that the team can move on (in the development)."**

A few interviewees mentioned how the planned release date and time related pressures in general have an effect on when the game is seen as ready enough for a release, and why it isn't solely the decision of those responsible for testing. One interviewee specifically mentions, how the state of the game on release is affected by how the team has handled the bugs during development, whether they have acquired 'bug debt' during development or fixed them at a good pace, how much time was left for testing overall in the project, when the release date was locked, and of other similar matters.

7.9 Evaluation of the success of testing

The approach of evaluating the success of testing was similar between the interviewees. The common theme was observing the aftermath of what happened when the game was actually released or updated. Company A was the only differing one from this trend and the interviewee stated that this so called 'testing of the testing' was not performed.

> **"[...] in practice, the way we evaluate the testers performance, we look at our customer feedback. If our customers bad mouth our game as a buggy mess, then at that point the test team will also hear their piece. […] This is the only reliable measure we have used to examine this with."**

> **"[...] But for the most part, the success of testing is evaluated by the feedback coming from the end users, observing whether the game works properly or not. It is also observed, how the game fares on the market and if the consumers have problems with the game. […]"**

Some of the interviewees provided more detail for this approach. Company B mentioned the effects of the release on the amount of users, what happens to the purchase behavior of the users, how the users react to the new features in the release,

and general user satisfaction. These were mentioned to provide a general view of whether the testing was successful or not. In order to measure the general user satisfaction, the method of using user polls (e.g., to ask what users think of the new features) was mentioned. Taking specific measurements and finding the holes in testing from which bugs can get through was not the aim, instead it was more preferable that the testers themselves learned from this experience.

Company C mentioned using the feedback coming from the alpha testers for evaluating the success of testing. Testing was always seen as having failed on some level, if serious enough problems reached the alpha version. In contrast to company B, the question why a specific problem reached a specific state in the testing process was seen as a starting point, and thus an important matter. Company D mentioned that when the product is released, it is then possible to compare the observations that were made throughout testing to the actual situation when the game went live. Thus the comparison of this 'vision' of the current state of the game to the actual state of the game, is a good indicator of how successful the testing was.

8 DISCUSSION AND CONCLUSIONS

8.1 The game development process

When the general game development process and the phases it consisted of was discussed, the in-house processes the interviewees described seemed to be for the most part aligned with the phases presented in the examined literature (Manninen, et al., 2006; Keith, 2010, p.131). Both the literature (Manninen, et al., 2006; Keith, 2010, p.132) and some of the interviewees also mentioned the fact, that the phases are not completely sequential, that they can overlap with each other, e.g., where some activities are still in the pre-production phase, while the majority is in the production phase, and that most of the design and development is iterative in nature. To slightly expand the main phases presented in the literature (Manninen, et al., 2006; Keith, 2010, p.131), the addition of alpha, beta and gold phases felt relevant, as typically major testing is performed during these sub-phases. This division also seemed to lead to a good flow of discussion, and generally seemed to work well, during the interviews. So, when the subject area to be discussed also includes testing, a good division for these development phases could perhaps be: Conceptualization & Prototyping, Pre-production, Production, Alpha, Beta, Gold and Post-release or Post-production phases. A thing to note is that the alpha, beta and gold phases can be seen as being part of the production phase. To make this more clear, it could be possible to rename the Production phase to Main-production phase or the Alpha, Beta, Gold phases could be placed under a Late-production phase.

8.2 Testing in different phases of development

When comparing the interviewed companies with the theory presented on testing in different phases of game development (Schultz and Bryant, 2011; Chandler, 2008), the texts seem to be mostly aligned with the interviewed companies, however some differences can be observed.

In the pre-production phase, these differences include the phase acceptance criteria, the participation to the game design reviews, and the handling of preliminary testing. In the presented literature, Schultz and Bryant (2011, p.104-109) mentioned that the phase acceptance criteria would define the entry criteria, exit criteria, and target date of each test phase. However, such strict definitions, on e.g., when a game reaches the alpha phase and then later moves on to the beta phase, did not come up during the interviews. Most of the interviewees defined reaching a new phase in development on a more broader level, describing briefly, in what state the game contents were. E.g., in a beta, most of the features were content locked, and existing features were being polished. However, as an example, Schultz and Bryant (2011, p.114-115) mentions 11 different alpha phase entry criteria, ranging from confirming that the game logic and AI is final, that all the controllers work, to confirming that the game can be navigated on all paths. As these strictly defined phase acceptance criteria did not come up during the interviews, the overall impression is that most of the companies did not define these phase transitions on such a detailed level. For the second difference, Schultz and Bryant (2011, p.104-109) mentioned that the lead tester should participate regularly in design reviews. None of the interviewees directly reported if this happened or not. However, one of the interviewees mentioned that the dialogue between the design team and the

testers should be improved, and one interviewee mentioned that the features of the game were not always so well documented. This raises doubts, that such a thing would have been the case in all of the interviewed companies. Finally, Schultz and Bryant (2011, p.108-109) mentions the act of preliminary testing. However, for some of the interviewed companies, this specific type of module testing was not mentioned, and testing was practically seen as starting from the alpha phase. This shows that this type of modular, preliminary testing is not mandatory for game testing.

For the beta phase, this was the first phase in the presented literature (Schultz and Bryant, 2011, p.104-109), where bringing people from outside the company to test the game was mentioned. However, among the interviewed companies, there were mentions of closed alphas, and bringing people to check the alpha version of the game, thus showing that this type of testing can already be performed in the alpha phase of development.

8.3 Outsourcing of testing

One big aspect in general that the presented literature seems to be missing is how the outsourcing of testing is planned and executed in different phases of game development. This raises interesting questions on what would be typically good areas to outsource in testing, when this outsourcing should be performed, and what are the typical benefits and possible drawbacks of this outsourcing. The results of the interviews regarding outsourcing of testing provides some answers to these questions. One testing area to be typically outsourced seems to be testing that requires special skills, especially those the team itself may lack e.g., localization testing or platform specific compliance testing. A second area could be testing which is only needed at certain times during development and not full-time. Some of the interviewees also mentioned that if the entire game project is small enough, such as a mobile game, it may be a good idea to send the entire game as a full package to an outside testing organization to be tested. This could yield detailed reports on a rather fast schedule, and thus is a way to outsource most of the testing. The cases of three of the interviewed companies also showed that it is feasible to outsource a large part of testing. A good time to start to outsource testing seems to be when the game reaches the alpha stage, typically when the game is for the first time playable from start to finish. This seemed to be a typical time when the work hours on testing largely increased, and remained high all the way to producing the gold master version of the game. However, as shown by e.g., the case of company C, there doesn't seem to be a specific boundary when the outsourcing of testing can begin. It seems feasible to outsource parts of testing, as long as something which can be tested exist.

The typically mentioned benefits of outsourcing were the cost savings and the scalability it provided. The cost savings come from being able to scale the amount of testers on demand, and being able to decrease the required amount of personnel knowledgeable on certain testing areas. The scalability makes it possible to avoid situations where the testing personnel may not have anything to work on during the production. For the downsides of outsourcing, one could be the lack of control for the performed testing. This could raise issues such as whether the testers are actually performing the testing in the way it was originally planned, and whether the reports actually come in a form that was originally planned. On this subject, one of the

interviewees stated that on one occasion, the received reports on compliance testing from a publisher did not meet the quality standards of the company, leading to a situation where most of this data was not used. Another possible downside of outsourcing could also be the amount of trust that can be given to these outsourced testers. If the feature that was covered by this testing contains a lot more undiscovered defects, and these stay in the end product, the customer will blame the company responsible for creating the game. For example, in a case of testing a mobile game fully by outsourcing the testing, and relying mostly on just the received reports, a lot of trust is placed on the outsourcing company. Another downside raised by one of the interviewees was related to the fact that it may not be feasible to physically participate in the outsourced testing. In these cases, unless the sessions are recorded, all the benefits that come from being present in the testing session are lost, e.g., seeing how the participants of focus groups behaved.

8.4 The testing methods

The testing methods that were used by the interviewed companies were summarized in table 3. This table gives a picture on what were the most used and less used methods of game testing in these companies. This raises a question on why some methods were more preferred over others. The most used methods were ad hoc testing, test case based manual testing, automated testing and focus group testing.

Ad hoc testing was used by all of the companies. The impression taken from the interviews was, that this is pretty much an essential method to be always used, when a game is being developed. One of the major reasons for this could be that as a game is typically very complex and it can be in so many different states, it is just not feasible to cover all the different states the game can be in. Playing the game in an ad hoc fashion attempts to give some coverage to this practically unlimited amount of states the game can be in. As opposed to the use of test cases, the general impression was that if the test cases were too extensive and were too descriptive, they were simply too laborious to construct and used too much time of testing. As an example of this, one of the interviewed companies mentioned that they had built more detailed test cases in the past. However, at the time of the interview, the company found it best to use only very brief descriptions when testing different features of the game, such as to test the game with a specific aspect ratio. Exploratory testing was specifically mentioned by only one of the interviewees. However, if we look at how this testing method was defined, and compare it to the other interviewees mentioning ad hoc testing, we notice that ad hoc testing did not always mean the testers could not be given subject areas to test, and possibly even more instructions, e.g., as in the case of company F. Giving some kind of instructions on what the testing should be focused on, even though the testers were otherwise free to choose their approach, seemed like a rather commonly used approach. So this seems more of a case where some of the interviewees did not make a distinction between ad hoc testing and this type of exploratory testing.

Automated testing was used by all of the companies, mostly because in every company, the developers built unit tests for the code to some degree, and had some form of build automation, which was categorized under automated testing. In comparison, load testing, smoke testing and soak testing were mentioned fewer times. Only one of the

interviewed companies did extensive automated testing. This interviewee, among with two others, mentioned that building extensive automated testing (e.g., tests, which involve automatic use of the games UIs) is only really worth it when the plan is to support the game for a longer time, such as an online game . If you think of a typical console title release, the automated tests will become unusable as the game is released, barring using them for further updates and dlc. However, with a game that is planned to be supported for a long time, the automated tests themselves are usable, and create value for a much longer time. It should also be noted that the more extensive the automated testing is, the more work is required to constantly maintain the entire automation suite. Load testing was typically mentioned to be used for testing the game's ability to handle network congestion. This type of testing should be performed if the game has network related features. Smoke testing and soak testing were mentioned by only one of the interviewees.

Focus group sessions were arranged by all of the interviewed companies. Recording the sessions in a video, where both the game footage and the participants' faces were recorded, was a common practice. In some cases, the hands of the participants were also recorded for the input they make. However, some of the interviewees mentioned that analyzing this video is a time-consuming process, and at times it was skipped. Arranging focus groups sessions seems very important for game testing, as this is one way the developers of the game get feedback that is very close to the possible thoughts of the end user. As the developers and testers work on the game daily, they become very familiar with the game. This can lead to situations, e.g., where it is difficult to tell if something is difficult to use, is obscure, is too difficult, if something is fun or not. As the participants are not as familiar with the game, these focus group sessions can give a fresh look on the game, by people similar in profile to the actual end users or customers. Using questionnaires, typically in some way during or after these session, was less common. One possible reason for this could be that it adds even further workload, in the form of time spent on focus tests, designing the questionnaires, and analyzing the results. One of the interviewees mentioned that when the focus group testing was performed, they typically used questionnaires during the sessions. The reason for this was that the questionnaires were seen as less laborious compared to the video recording and analysis. However, when this testing was outsourced, videos were typically recorded and analyzed. Switching to using questionnaires only seems like a good idea if processing the video is considered to be too laborious and time-consuming.

Compliance testing was mentioned by three of the interviewed companies. The reason for this is rather simple, as there are platforms where the device manufacturer has no compliance requirements, such as the PC. These device manufacturer requirements must be fulfilled, and if they exist for the game project, it is important to have them thoroughly tested before the actual submission. If the submission fails, it is typical that further submissions get more expensive, and naturally this extends the development time. In other words, using only the device manufacturer's testing for this is not a good approach.

Collecting and analyzing eye-tracking data was mentioned by two interviewees. The reason for this seems rather simple, as performing such testing requires special hardware, special software and special skills. It would seem likely that this typically needs some kind of an outside organization. Soft launching a product was mentioned by

a single interviewee. The likely reason for this is that only specific types of games are typically soft-launched, such as free-to-play titles, mobile games. A/B testing was only mentioned by one interviewee. The interviewee mentioned that the limiting factor of this method was that it gave away the surprise when new features to the game were introduced. For this reason, the method was used sparingly. When A/B testing is used, it should be acknowledged that the entire player base can become aware of the feature that is being tested, and for this reason, the targets for this testing should be chosen carefully.

Forum testing was mentioned by a single interviewee, and this was mostly because the publisher had a history of performing it. The interviewed companies however did mention using, e.g., closed alphas, open betas. This form of forum testing could be seen as open beta testing. From this viewpoint, it was a lot more typical for the interviewed companies to showcase the game for a larger audience at some point in development. The forums can be seen as simply a method to facilitate the communication between the developers and the outside testers, and this kind of communication could be facilitated through other mediums as well, e.g., in an open beta through an in-game survey or through in-game bug reports.

Blind testing was mentioned by only a single interviewee. However, this may be another case where the definition of focus group testing and blind testing were not that well distinguished by the other interviewees. Some of the interviewees did mention giving specific advice on what the goals of the participants should be during the focus group session, but it was also possible that at times, the participants were simply free to roam about in the game as they pleased.

Balance testing was specifically mentioned by two interviewees. One reason for this is that not all games have a specific balance testing need, as they don't contain things that need to be balanced. E.g., a game based around a community, focusing mostly on community interaction, such as chatting, has few things to balance. A second reason could be that this type of testing is taken for granted when the testers are testing the game. Gathering opinions on what seems difficult, how powerful the equipment feels, and similar matters comes along the way, when performing testing. No special testing methods are thus required for this. A similar testing method or area would be performance testing. Some of the interviewees specially mentioned performing it, and the methods that were explicitly used for it. However, it is natural to assume that those who did not explicitly mention performance testing, were at a bare minimum observing the overall performance of the game when they were performing testing on it. In this case, they would have simply estimated the frame-rate through observing the game and seeing how fluid the controls felt. Compatibility testing was mentioned by two interviewees. One reason for this is that some games are not released on a large number of different platforms. However, another reason could be that this seems quite an obvious requirement. If the game would be released, e.g., for the Xbox One, PS 4 and PC, all the versions would have to be specifically tested. The interviewees did not give specific descriptions on what differences there would be between testing these type of different versions of the game.

One possibility to keep in mind from the interviews is that some of the less used testing methods were simply not mentioned during the time of the interview. This was attempted to be avoided by trying to confirm that most of the methods that were used in

testing were covered in at least some form during the interview.

The interviewees were also asked to name what they considered the most important practice of the currently performed testing. Most of the interviewees found the question difficult to answer. The interviewee from company A saw most of the testing that was performed as critical, and being necessary. Company B mentioned the manual, black box style testing, which involved the use of test cases. Company C mentioned that if the performed regression testing could keep the game in a playable form, it would be this regression testing. The regression testing was based mostly on an ad hoc approach. Company D mentioned maintaining the focus on the end user, and keeping testing as part of the production process throughout the development. Company E mentioned blind testing. Company F mentioned ad hoc testing, and specifically, the recordings or logs that were taken during the testing. Company G mentioned, that as all the testing complemented one another, nothing should or would be feasible to be dropped. By examining the answers which mentioned a specific practice, no specific practice is highlighted over the others.

8.5 Further research topics

As the thesis focused on game testing on a rather broad scope, there are plenty of opportunities to further examine the topic. To give some examples, one possible topic could be choosing one of the presented methods, and researching it further. E.g., in the case of focus group testing, figuring out how they are currently being arranged by participating in these sessions, taking note of the differences, perhaps listing the best found practices for arranging these type of sessions. Another one, could examine the scope of the documentation, going into more detail, on what information e.g., the test cases themselves should contain, what is a typically good level of documentation, and defining the level of documentation which will typically start to hinder testing too much. A third possible subject could be, in a case of a game being released on multiple platforms, how these different versions of the game are tested. Is a baseline version of the game picked, and the other versions receive less testing compared to this baseline version, or is this solved another way. Also what are the other differences in testing between the different platforms.

REFERENCES

Au, W.J. 2012, *Game design secrets*, Indianapolis, IN: Wiley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin,R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas,D. 2001, *Manifesto for Agile Software Development*, http://www.agilemanifesto.org/, Accessed 29.8.2012.

Bleumers, L., Jacobs, A. and Van Lier, T. 2010, "Criminal cities and enchanted forests: a user-centred assessment of the applicability of the Pervasive GameFlow model", *Fun and Games: Proceedings of the 3rd International Conference, (Fun and Games '10), 2010, pp.38-47*, pp. 38-47.

Boehm, B.W. 1984, "Software Engineering Economics", *Software Engineering, IEEE Transactions on,* vol. SE-10, no. 1, pp. 4-21.

Brockmyer, J.H., Fox, C. M., Curtiss, K. A., McBroom, E., Burkhart, K. M., Pidruzny, J.N. 2009, "The development of the Game Engagement Questionnaire: A measure of engagement in video game-playing", *Journal of experimental social psychology,* vol. 45, no. 4, pp. 624-634.

Burnstein, I. 2003, *Practical software testing a process-oriented approach*, New York: Springer.

Chandler, H.M. 2008, *The game production handbook, second edition*, Hingham, Mass.: Infinity Siene Press.

Chen, J. 2007, "Flow in games (and everything else)", *Communications of the ACM,* vol. 50, no. 4, pp. 31-34.

Cowley, B., Charles, D., Black, M. and Hickey, R. 2006, "User-system-experience model for user centered design in computer games", *Adaptive Hypermedia And Adaptive Web-based Systems, Proceedings, 2006, Vol.4018, pp.419-424,* vol. 4018, pp. 419-based.

Crossley, R. 2010, *Study: Average dev cost as high as $28m*, http://www.develop-online.net/news/33625/Study-Average-dev-cost-as-high-as-28m, Accessed 13.5.2013.

Csikszentmihalyi, M. 1991, *Flow: The Psychology of Optimal Experience,* New York: Harper Perennial.

Fullerton, T., Swain, C. and Hoffman, S. 2004, *Game design workshop designing, prototyping and playtesting games*, San Francisco, Calif.: CMP.

Hamari, J. and Tuunanen, J. 2014, "Player Types: A Meta-synthesis". *Transactions of the Digital Games Research Association*, 1 (2), 29-53.

Hetzel, W.C. 1988, *The complete guide to software testing*, Wellesley, Mass.: QED Information Sciences.

Hight, J. and Novak, J. 2007, *Game Development Essentials. Game Project Management,* New York: Delmar.

Holt, R. 2000, *Examining Video Game Immersion as a Flow State,* B.A. Thesis, Department of Psychology, Brock University, St. Catharines, Ontario, Canada.

Humphrey, W. 2000A, "The Personal Software Process (PSP)", *The Personal Software Process* (PSP), Carnegie Mellon University.

Humphrey, W. 2000B, "The Team Software Process (TSP)", *The Team Software Process (TSP)*, Carnegie Mellon University.

IEEE. Institute of Electrical and Electronics Engineers. *1*990, "IEEE Standard Glossary of Software Engineering Terminology", *IEEE Std 610.12-1990*, pp. 1-84.

IJsselsteijn, W.A., Kort, d.Y., Poels, K., Jurgelionis, A. and Bellotti, F. 2007, *Characterising and measuring user experiences in digital games*.

Jegers, K. 2007A, "Pervasive game flow: Understanding player enjoyment in pervasive gaming", *Computers in Entertainment,* vol. 5, no. 1.

Jegers, K. 2007B, "Pervasive GameFlow.: A Validated Model of Player Enjoyment in Pervasive Gaming", *Concepts and Technologies for Pervasive Games: A Reader for Pervasive Gaming Research vol 1*.

Jegers, K. 2009, "Elaborating eight elements of fun: Supporting design of pervasive player enjoyment", *Computers in Entertainment (CIE),* vol. 7, no. 2.

Kaner, C., Bach, J. and Pettichord, B. 2001, "Lessons Learned in Software Testing: A Context-Driven Approach", *Lessons Learned in Software Testing: A Context-Driven Approach,* New York: Wiley Computer Publishing.

Keith, C. 2010, *Agile Game Development with Scrum,* Addison-Wesley Professional.

Kivikangas, J., Chanel, G., Cowley, B., Ekman, I., Salminen, M., Järvelä, S. and Ravaja, N. 2011, "A review of the use of psychophysiological methods in game research", *Journal of Gaming & Virtual Worlds, Sep 2011, Vol.3(3), pp.181-199,* vol. 3, no. 3, pp. 181-199.

Kohavi, R., Longbotham, R., Sommerfield, D. and Henne, R.M. 2009, "Controlled experiments on the web: survey and practical guide", *DATA MINING AND KNOWLEDGE DISCOVERY,* vol. 18, no. 1, pp. 140-181.

Koutonen, J. 2011, *Ketterät menetelmät peliteollisuudessa: Suomalaisia pelistudioita koskeva kyselytutkimus,* Jyväskylän yliopisto, Tietojenkäsittelytieteiden laitos.

Levy L. and Novak J., 2010, *Game QA & Testing*, New York: Delmar.

Manninen, T. 2007, *Pelisuunnittelijan käsikirja. Ideasta Eteenpäin,* Tallinna: Kustannus Oy Rajalla.

Manninen, T., Kujanpää, T., Vallius, L., Korva, T., Koskinen, P. 2006, *Game production process. A Preliminary study.*,http://www.danielparente.net/es/wp-content/uploads/sites/2/2013/04/game_production_process.pdf. Accessed 13.6.2012.

Metsämuuronen, J. 2001, *Laadullisen tutkimuksen perusteet,* Helsinki: International Methelp.

Morgan, A. 2012, *Production Methodologies, Techniques and Guidelines for Modern Computer Game Testing*, Ashley Morgan.

Norman, K.L. 2013, "GEQ (Game Engagement/Experience Questionnaire): A Review of Two Papers", *Interacting with Computers, 2013, Vol.25(4), pp.278-283,* vol. 25, no. 4, pp. 278-283.

Perry, D. 2008, *29 business models for games,* http://lsvp.wordpress.com/2008/07/02/29-business-models-for-games/, Accessed 2.7.2012.

Pressman, R.S. 2005, *Software engineering a practitioner's approach*, New York: McGraw-Hill.

Ravago, M. 2009, *Game Development Outsourcing.* www.gameshastra.com/docs/Game_Development_Outsourcing.pdf, Accessed 13.5.2013.

Schultz, C.P. and Bryant, R.D. 2011, *Game testing all in one, second edition*, Dulles, Va: Mercury Learning and Information.

Sweetser, P. and Wyeth, P. 2005, "GameFlow: a model for evaluating player enjoyment in games", *Computers in Entertainment (CIE), July 2005, Vol.3(3), pp.3-3,* vol. 3, no. 3, pp. 3-3.

Wallace, D.R. and Fujii, R.U. 1989, "Software verification and validation: an overview", *IEEE Software, May 1989, Vol.6(3), pp.10-17,* vol. 6, no. 3, pp. 10-17.

Whittaker, J.A. 2009, *Exploratory software testing tips, tricks, tours, and techniques to guide test design,* Upper Saddle River, NJ: Addison Wesley Professional.

Williams, L. 2006, *Testing Overview and Black-box Testing Techniques*, *http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf,* Accessed 13.5.2013.

APPENDIX A – The interview questions

Appendix A contains the interview questions that were used in the interviews. They were translated to English from Finnish.

1. What kind of games is the company producing? (e.g., mobile, PC, console)
   - 1.1. What kind of game projects have there been?
   - 1.2. What was the typical length of the project and the amount of people in it?
   - 1.3. How did the size of the projects vary?
   - 1.4. Has there been many projects simultaneously, how many?
   - 1.5. How did the game development proceed and what were the phases?
2. How was the testing organized?
   - 2.1. How many people were involved and what was the organization? How many people of the company were involved in testing?
   - 2.2. Do the developers of the game (not only testers) perform testing as well?
   - 2.3. Did the testing require any special skills? If it did, what kind? Has there been a need to train personnel? If yes, what has this training included and how was it implemented?
   - 2.4. Is any kind of testing done outside the company / using personnel outside the company? If yes, who and what? Why are the outsiders used?
   - 2.5. If not, has there been any consideration to outsource some parts of testing, has any kind of evaluation been done related to the costs of outsourcing? Have the possible benefits / disadvantages been considered?
3. How much of the game's budget is spent on testing? How many work hours is used on testing compared to other tasks?
4. What kind of meaning did the game quality factors have on the game development and game testing?
   - 4.1. What are considered to be the most important game quality factors?
   - 4.2. Do these quality factors have priorities?
5. How is testing planned? What kind of documents are created related to this? How much of testing is based on plans? How much time is used for test planning?
6. What are the objectives of testing?
   - 6.1. What are the most important objectives?
   - 6.2. What kind of objectives does testing have in the different phases of

game development?

6.3. Is ensuring the fun factor of the game one of the objectives of testing? How important is this in testing?

6.4. What methods are used to test the 'fun' of the game?

6.5. Are focus groups used? Are the groups divided (e.g., beginner players), are different things tested on different subgroups?

7. How is testing arranged in the company? What kind of a process model is used in testing? As what kind of a process can testing be described?
8. How is testing arranged in different phases of game development? What is tested in these phases? What kind of testing methods are used in the different phases of development?

8.1. Pre-production

8.2. Production

8.3. Alpha

8.4. Beta

8.5. Gold

8.6. After-release (different support actions / possible future development)

8.7. Are these good phases? If not, what would fit better?

9. Is automated testing used? (e.g., unit tests)

9.1. In what ways?

9.2. What are the current experiences on it?

10. When is testing stopped, are there specific measurements for this? How does one know when to move to the next phase in development?
11. What methods are seen as being most useful in testing, what would be discarded last?
12. How is testing and the results of testing documented? How is the information on found bugs recorded / how is the information on performed testing recorded?

12.1. If there is a bug database, what kind?

12.2. Are pictures or video used for this? Where are they specifically used?

12.3. How is the data for the tests reported?

13. What programs are used in testing?
14. How is the functionality, success and the results of testing evaluated? What data is collected on the performance of testing? Are the testing methods and processes improved based on this data? What kind of improvements have been made?

15. What is especially difficult in testing? What makes it hard?
16. How should the current testing practices be improved and changed? What might be stopping this?
    16.1. What kind of changes have been done before?
    16.2. What kind of problems has there been?
17. What has been learned of testing? What knowledge and know-how of testing should be improved?
18. Has something essential for game testing been omitted or not discussed enough? What else could you tell about the implementation of testing in the company?