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

Tomi Ratilainen

Guidelines for creating a testing process for a software -

case study of comparing testing of two different size of slot game projects

Master's Thesis
Espoo, May 13, 2019

Supervisor: Professor Antti Ylä-Jääski, Aalto University

Instructor: Tuomo Hakulinen, Quality team lead, Veikkaus

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

ABSTRACT OF
MASTER'S THESIS

Author: Tomi Ratilainen

Title:
Guidelines for creating a testing process for a software - case study of comparing testing of two different size of slot game projects

Date: May 13, 2019 **Pages:** 93

Professorship: Mobile Computing, Services and Security **Code:** T-110

Supervisor: Professor Antti Ylä-Jääski

Instructor: Tuomo Hakulinen, Quality team lead, Veikkaus

Nowadays an important part of software development life cycle is software testing. As software and systems become more complex and integrated with other software and systems the importance of software testing becomes critical part of a successful software development process.

Testing itself has a little value but for the software the testing is necessary. Well planned software testing can make mediocre software a great one whereas poorly executed testing can even harm the final product. Thus, it is important that there is well designed testing process for a software project. As every software project differ from each other the testing process can also vary from project to project. There are multiple levels, types and techniques of testing that can implemented to a testing process made for a certain software project. Creating an efficient and functional testing process can be a difficult task.

In the case study of the paper it is concluded that two similar software projects of different size have multiple differences but do not force of creating two different testing process as both use the same kind development life cycle and both software projects are the same type, slot games. The major difference for testing is that as a major software project, compared to a minor one, has more requirements it is important to consider them in testing strategy and planning but not in the testing process.

Keywords: Software testing, Testing types, Testing levels, Testing techniques, Testing process, Slot games, Software quality, Software development processes

Language: English

Acknowledgements

I wish to thank my instructor, Tuomo Hakulinen, for giving me time to complete my master thesis during work and to Professor Antti Ylä-Jääski for fast feedback when it was needed.

Espoo, May 13, 2019

Tomi Ratilainen

Abbreviations and Acronyms

UAT	User acceptance testing
OAT	Operational acceptance testing
QA	Quality assurance
ROI	Return of invest
DoD	Definition of Done
API	Application programming interface
TPI Next	Test Improvement Next model for improving test process
CTP	Critical Testing Process model for improving test process
STEP	Systematic Test and Evaluation Process model for improving test process
TMMi	Test Maturity Model Integration model for improving test process
Tmap	Test management process
COTS	Commercial off-the-shelf
BVA	Boundary Value Analysis

Contents

List of Figures

Chapter 1

Introduction

Software are a major part nowadays world. Nearly everything created by human has some software build in it. Every mechanic construct has a software even though few decades back we did not even have computers. Even simple little things like a clock can have software that is comparable to software of a computer. The increasing number of software products means that there must be more people programming software, meaning that most of software is not done by the senior programmers. Thus, probability of errors in coding, called bugs, must be increasing to a higher number. We live in a world of capitalism which leads to situations where software is just required, done quickly or ordered hastily from a third-party software company without a proper check leading to more and more poor-quality software with bugs in it.

When adding a software to a product it usually defines the product. How it operates or even changes appearance of the product. Thus, it is most important that the software itself operates as planned in every normal situation possible for the product. Software testing is one of the best tools against human errors made during creating the product. Software testing is an important part of nowadays cycle of creating a high-quality product. With testing it is assured that the product has no critical malfunctions and confirm quality of the product. When the competition in the market is done mostly by comparing the software, it is not surprising that software testing has a huge part in creating popular and successful outcome of the product.

Though the testing can be included from the start of creating a product it does not always conclude that there could not be malfunctions or that the outcome of the product would be a high-quality product. If the testing is done without explicit process understandable for every tester with easily followed guidelines and instructions, the result can be quite unsatisfactory. In worst case scenario it could lead to same result as no testing at all. One

possible way of this happening is that QA (quality assurance) team might be stuck in certain way of testing even though the team should adjust their testing processes, techniques, models and types in different software projects. On the other hand, a successful and well-planned testing process from the start of the design of the product might even add extra value to the product. The product becomes more than it was designed to be, in a good way.

This paper tries to answer why software testing is essential in nowadays world and what are different types, methods, levels and models of testing and how together they can create different testing processes creating functional QA environments for testers. The paper introduces few useful tools used in testing, talks about importance of communication in testing and tries to show that with successful testing process QA has return of invest (ROI) value [33]. The paper will also introduce precise testing process and results done to a two different slot games designed and created by Veikkaus.

1.1 Problem statement

In most of software projects it is quite uncertain at the start what would the best means to test the software and in which points. Thus, it is important from the start of the project to design practical testing process that does not leave any place to speculate. Desirable testing process is understandable and possible to execute efficiently. Whereas inadequate testing process might be complex and seeming to miss major testing points during the life cycle of the software project leaving an uncertain understanding of the quality of the software.

As every software differs from one another it becomes obvious that ready-made solutions for testing processes can only offer direction for suitable and functional testing process. There are common set of test activities that can be applied to achieve desirable objectives concerning the testing of the software. As all common tools for testing are properly known it is easier to create functional integrity for the testing process of the software. [25]

In many companies exists no QA team explicitly. Different teams have projects of their own and they might be hiring a consultant as responsible for testing of the product or have one person in the team to be responsible for testing. The responsible might have a way of working, a ready solution to be used as a testing process or few known practices used, which does not include total set of available toolkits for successful testing considering the product. As mentioned above all software differ and thus it is critical that for every different software there is enough knowledge behind decision made concerning testing process.

How can a single person with limited knowledge or QA team stuck in certain testing habits assemble best solution for a new software concerning the testing process or improve their current way of testing? What are some existing functional testing processes? How can be those process modified to support software under test? What are the testing types, levels, techniques and models used in those existing processes? Is fully agile model or strict waterfall model better for testing process concerning the product? Ultimately, how can one choose a corresponding testing process for a software?

The paper introduces basic theory of software testing through chapters 2 to 5. Different test levels, test types, test techniques are presented as well as main levels of testing process and what they contain. Different ways of improving the testing process are introduced as well as basic development process models to support understanding of creating the corresponding testing process for a software.

In the case study of this master's thesis project, we analyse, evaluate and compare two different slot game projects. Other slot game project is a single game project not concerned and not needing a cross platform modifications to slot machine itself and to internal system supporting the slot machine. Whereas the another slot game requires cross platform modification to the slot machine as developing of internal systems supporting the slot machine. Both slot game projects use the same testing process.

As a result of analysing, evaluating and comparing the two different slot game projects we show differences between the found defects in each of the project and the number of critical issues found during testing. As a result we also try to conclude should the test processes differ between a smaller slot game and cross platform changes demanding slot game.

1.2 Structure of the Thesis

In chapter 2, Fundamentals of testing, the focus is in the objectives of testing and why software testing can been seen as necessity in development process of a software. First some of the objectives of testing are being listed and then some other reasons other than failures in the code of the software why there can be defects and malfunctions in the software and thus, why software testing is essential. The end of the chapter introduces few principals of testing.

Different styles of testing, chapter 3, focuses on different types of testing, different testing levels and techniques. Every subchapter introduces basic types, levels and techniques of testing. First two basic ways of testing are presented, dynamic and static testing. Then four basic test types are ex-

plained. Thirdly, five major testing levels are introduced from unit testing to acceptance testing. Lastly, three different major testing techniques types are introduced, and examples of each type of testing techniques are presented.

Chapter 4, Testing process, focuses on introducing three different levels of testing process and what does every level contain. Also, the chapter focuses on how to improve the testing process and how to keep communication between developers and testers reasonable. Three main levels of testing process are introduced in order of organizational test process, test management process and dynamic test process. Every level of testing process presents what different tasks they contain. For improving testing process few improvement models of testing process are introduced. The last part consist of multiple tips how to make communication better between developers and testers and how a tester can increase the trust during a software project.

Chapter 5, How to create the right testing process for a software, considers how does the testing process take into account the development process of a software project and how to tackle major risks in a software project. Basic models of different software development process models are introduced and how to identify different kind of risks and how much can a defect cost in different time of the development lifecycle.

The case study, chapter 6, focuses on analysing, evaluating and comparing two size of different slot game projects. The development process of slot machines of Veikkaus is introduced as well as testing process used in testing slot games made for the slot machines. The testing of two different slot game projects are presented, and the results of testing are shown and analysed. Lastly, from the data of results, different conclusions are made between the two slot game projects.

Chapter 2

Fundamentals of testing

Most people have experienced different kind of software. Software that feels to operating always and software that from the start feels little clunky and hard to use even with a manual. Sometimes software does not even function properly or as expected even though it is brand new. As software appear nearly everywhere, a new refrigerator, a clock or a popular consumer product like a car, it has become more and more critical that the software functions as planned. Dysfunction of a software can lead from minor problems to major problems. Examples consequences of dysfunction software can be loss of money, time or business reputation. In extreme cases consequences are as severe as injuries and in worst case scenario death. The purpose of software testing is to access the quality of the software and give information about of the quality and thus reduce risk of malfunction after the software has been released.[27] [25]

Software testing has become a standard part of developing a software. Even though software testing as a concept is well known there are still quite a few common misperceptions about software testing. For many, software testing only consists of running tests, meaning executing software and examining appointed areas of the software and in the end checking the results of the tests. Running the tests and reporting the results is major part of testing process and in most projects also the most important part. Software testing is a process that includes many other activities than running the tests. The test process includes activities such as planning and creating of tests by means of requirements. Also reporting test progress, results, defects in software and evaluating the quality is part of software testing.

Although testing can mainly focus on verification of requirements and specifications it does not entirely focus on these specifications and requirements. It is common misunderstanding that only wanted specifications are tested. Testing also includes confirming that product meets the expectations

of users and stakeholders.

Testing usually focuses on probing the software. Dynamic testing is a testing where testing does involve execution of the software. It is also common misperception that all testing is dynamic. Non-dynamic testing is called Static testing. Static testing can include reviewing user stories or source code of the product.

Also, terms debugging and testing can be tangled together even though they are different. Debugging is development activity with a main goal of finding a fix for a defect. Whereas testing finds malfunctions and defects in software. Usually developers do debugging and fixing of a defect and tester makes final confirmation test for the defect. In highly agile development environment it is possible that also tester involved in debugging. [25]

2.1 Objectives of testing

Common objectives for any software might include:

- Evaluate requirements, user stories, design and code
- Verify specified requirements
- Confirm that software meets expectations of users and stakeholders
- Inform the level of quality of the software
- Prevent defects, find defects and malfunctions
- Providing enough information to owner of the project of quality and risks so that decisions are easier to make

Objectives of testing vary upon a context of the software and needs of testing concerning about the software. Objectives also vary in different states development life cycle of the software. For example, at mid stages of development it can be important to find as many defects as possible whereas before release date it is most important to relate information about the quality of the software and risks concerning the release. [25] [16]

2.2 Is testing necessary?

In simplest precise testing of the product usually helps to reduce the risk of malfunction of the product in production. As defects are found during testing and hopefully fixed, leads this towards higher quality of the product.

Furthermore, software testing is required due to legal requirements or by specific standards.

It has been proven by history that it is common for software to be in production and cause failures due to defects. Using suitable testing process, techniques and having appropriate level of testing experience usually results in fewer failures due to defects in production. To strengthen the effect of testing it is typically essential to keep tester close from the start of design part of the project to the delivery to production to ensure best possible outcome of testing.

Mistakes do happen and it is know that human errors happen time to time due to different reasons. Errors lead to defects in the software and defects can lead faults in production. Example path of to a defect is making an error in early stages of creating requirements or specifications. Error made in early stages in requirements leads quite easily to an error in programming which ultimately leads to defect in code. [25] [32]

Few reasons why errors may occur:

- Too strict schedule
- Human error
- Inexperienced team
- Miscommunication
- Complexity of used technologies or using of unfamiliar technologies
- Misunderstandings

As important it is to find defects and report them forward it is also important to understand what could be the root cause of the defect and what effect would a certain defect cause in production. For example, single line of incorrect code when using mathematical equation in finance. Root cause might be error made in specification of providing incorrect information for developer resulting in defect in code. Effect of the defect depends how critical part of the product the equation is. Example effect could be increased number of customer complaints. [25]

2.3 Seven principles of testing

1. Testing shows the presence of defects, not their absence

   (a) In testing it is quite likely to find defects, but it does not prove that there could not be more defects or no defects at all.

2. Exhaustive testing is impossible

   (a) Testing all combination of software is usually impossible. In corner cases it can plausible to test everything. Testing process and techniques are key to a sufficient coverage.

3. Early testing saves time and money

   (a) Defects found early are easier to fix and more cost efficient.

4. Defects cluster together

   (a) Usually it is small number of parts in software that are broken.

5. 5. Beware of the pesticide paradox

   (a) As same test is run again and again it no longer cannot find any defects. Changing test data or creating new test may need to created.

6. Testing is context dependent

   (a) Every project and software are different. Some can include life-saving safety features whereas other software can be done just for leisure.

7. Absence of errors is a mistaken belief

   (a) Finding and fixing a large number of defects does not conclude a software that would fill the needs of users and stakeholders

[25]

Chapter 3

Different styles of testing

There are multiple ways doing testing. There are different kind of testing types, techniques, levels, models and even static testing. In this chapter most known of those different styles of testing used in development lifecycle of software are introduced.

- In dynamic testing software is being executed while static testing is merely manual exploring of work products
- Types of testing aim to test specific matter of the software
- Test levels are part of testing process each appearing in different stages of development lifecycle of the software
- Test techniques help to discover test cases, test data and test conditions

3.1 Dynamic and static testing

Most of testing is linked in testing with operating software. Executing tests with the software is called dynamic testing. Usually most of the testing is done dynamically. Even though it is easy to misunderstand that dynamic testing would be the only way to test there is another way of testing called static testing.

Static testing entrusts on manual exploration of work products (i.e., reviews) or evaluation of code. Both types of static testing do not include executing the code or testing a work product in any way. It might feel that static testing is quite a minor part and thus not playing important role. In safety-critical systems static analysis plays especially important role and has become more important role of testing in other common systems for example, security testing.

There are quite a few work products that can reviewed or statically analysed, for example [25]:

- Specifications and requirements
- User story and Definition of done (DoD)
- Code review
- Test plans and test cases
- User guides
- Web pages
- Project related plans, contracts, timetables etc.

Benefits of static testing may include:

- Finding defects in early stages (before dynamic testing) which is also cheaper than finding defects during dynamic testing
- Finding defects that dynamic testing has difficulties of finding
- Reducing development and testing time and costs
- Reviews can improve communicating between team members

Main purpose of static testing is to improve the overall quality of the product and find defects. In addition to finding defects with static testing can have secondary purpose when executing technical reviews or doing walkthroughs. Technical reviews can improve consensus of the product and walkthroughs can be used for exchanging information between team members. [7]

3.2 Test types

Group of test activities intended to test certain qualities of software based on objectives of testing are called test types. Objectives of different testing types can include evaluating completeness of the system, performance efficiency of the system, architecture correctness of the system and confirming that future changes in software does not cause it malfunction. All these four examples are from four different kind of types test types:

- Functional testing

- Non-functional testing
- White-box testing
- Change-related testing

3.2.1 Functional testing

Functional testing at it's simplest tests that functions of system operate as requirements of work product describes. Such requirements can user stories, use cases, functional definition and in some cases undocumented specifications of the system.

Functional testing can be performed at all test levels even though the focus will be different at each level. Testing technique mainly used in functional testing is black-box testing.

Functional testing in prospective of design and execution can involve particular knowledge or skills. Such a skill could be understanding and creating mathematical equations concerning the product.

Functional testing can be measured through coverage of functional properties of the system. Functional coverage concludes functional elements covered by tests and is presented percentage of different types of elements covered. [25] [32]

3.2.2 Non-functional testing

Non-functional testing evaluates how well does the system behave in different states. Testing targets of non-functional testing are such as usability, scalability, performance efficiency and security.

As in functional testing non-functional testing may also require sufficient knowledge or skill. These are needed in certain area of technology or knowing common weakness of design. For example, understanding common scalability problems.

Even though it might seem that non-functional testing requires a software at end of development life cycle, it is a common misperception to think that non-functional testing could not be done in early levels of testing. Non-functional testing as functional testing should be performed at all levels of testing.

For understanding of satisfactory coverage non-functional testing same applies as for functional testing. Coverage can be based on non-functional elements being tested and percentage of these types of elements being covered. For example, in case of usability in different mobile devices the traceability between tests and supported mobile devices. [25] [26] [31]

Example list of non-functional testing targets [23]:

- Security
- Performance
- Reliability
- Data integrity
- Acceptance
- Stability

3.2.3 White-box testing

White-box testing focuses on internal structure or implementation of the system. Such examples of internal structures can be architecture, code, data flows and work flows within the system. Most of white-box testing is usually executed in at component level testing but can be used at all levels. As white-box testing is at the same time testing technique as a type of testing most of the information relates closely to technique. More about white-box in chapter about testing techniques. [25]

3.2.4 Change-related testing

To correct a defect or updating software with new functionalities changes to the system are needed to be done. Change-related testing confirms that changes made to the system have corrected the defect or that new functionalities of upgraded software operate as premeditated.

There are basically two types of change-related testing types; confirmation testing and regression testing. Both types of change-related testing can be performed at all test levels.

In confirmation test it is verified that the defect that caused the problem is fixed by at least verifying reproducing steps of the defect. If the defect is not reproducible by the steps it is fixed. If time allows it also a good idea to check all test cases failed due to the defect meaning retest of all failed test cases related to the defect. In some cases, fixing a defect may lead to new functionalities. As new functionalities are added new test cases can be needed. Thus, it is possible that in confirmation testing new test cases are run.

Fixing a defect can change only one part of the code but might knowingly or unknowingly have affect to behaviour of other parts of the code. Also

changes made to fix the issue might also need changes to the environment such as upgrading operating system. In regression testing all unintended side-effects of fixing the defects are tried to be found by running set of tests. As regression tests can be run many times during project and usually has same test set as before it is strongly suggested that automation is used to help in regression testing.

Change-related testing is especially important agile development lifecycles and in systems where devices or system upgrades are constant. [25]

3.3 Test levels

Test levels are instances of a test process performed to relation of development level of a system. Every test level is suitable for different state of development from individual components to a system ready for production. Mostly used test levels are:

- Unit testing
- Integration testing
- System testing
- Acceptance testing

Each test levels have designated objectives, testing targets, common defects and certain responsibilities. Also, every test level requires environment fitting for the test level. For example, for unit testing developers own developing environment might be enough whereas in acceptance testing an environment should be comparable to production. [25] [19] [31]

3.3.1 Unit testing

Unit testing focuses on separately testable components of a software. As unit testing focuses on separate parts of the software testing is often performed in isolation from other components of the software. Although testing is mostly performed in isolation of the rest of system different other tools might be needed such service virtualization or mock object instead of real parts of the system.

Normal environment for unit testing is development environment and thus, as defects are found they are usually fixed as soon as possible and are not reported to formal channels. Even though it is not usual to report defects

formally at this stage it is possible to keep at least record of the defects or sometimes report them even formally for all developers to see.

Unit testing is usually performed by writer of the code i.e. developer. Unit tester can be another developer or even someone with access to the code. Writing tests for unit testing is often done after writing the code by developer. As development is rapid or agile it might be reasonable to write automated test cases.

Objectives of unit testing:

- Risk reducing and preventing defects affecting higher test levels
- Unit operates as designed and specified (functional and in non-functional)
- Increased reliability of unit s quality
- Finding defects

  Incorrect functionality

  Incorrect logic in the code

  Data flow

Typical test objects of unit testing include components, units or modules, code and data structures, classes and database modules. [25]

3.3.2 Integration testing

As unit testing focuses on separate parts of a software integration testing focuses on combining of components or systems and testing interactions between them. There are two typical levels of integration testing; Unit integration testing and system integration testing.

Unit integration testing focuses on the interactions between integrated units, modules or components. Unit integration testing can be performed after enough components are coded meaning that unit testing should be done before unit integration testing. It is common that automation is used in unit integration testing.

The focus of system integration testing is interactions between systems, packages and microservices. System integration testing might also cover interaction with for example web services of external organization. System integration testing can be challenging if one of the tested systems is from external organization. System integration testing can be executed properly after system testing. It is possible to have system testing and integrations testing of the system in parallel in the testing process is in ongoing state (development is sequential or iterative).

Integration testing should concentrate on testing the actual integration. As mentioned above both system testing and unit testing should be done before starting integration testing. For example, when testing integration between modules or systems A and B the focus should be on for example, communication between A and B not in individual properties of A or B.

Unit testing is usually performed by developers so is unit integration testing. As for system integration testing, it typically done by testers. System integration testing requires certain understanding of systems acchitecture.

Objectives of integration testing are mostly the same as in unit testing difference being that focus point is combination of components or systems. Thus, test objects are quite different [25]:

- Subsystems
- Databases
- Infrastructure
- Interfaces
- APIs
- Microservices

Typical defects of integration testing [25]:

- Incorrect data
- Interface mismatch
- Failures in communication between components or systems
- Communication failures between systems or components are poorly handled

3.3.3 System testing

As the name suggest system testing focuses on functioning of entire product or system. System testing often includes end-to-end tasks of the system and containing non-functional features while performing these tasks.

For lower test levels like unit testing or integration testing it is confirmed that a part of system is working as expected whereas in system testing focuses gathering information of the whole quality of the system. Thus, system testing produces information that can be used to inform stakeholders when

making decisions about releasing. System testing can also be a part of satisfying legal or regulatory standards or requirements. As system testing information is used to many extremely important decisions it is quite important that the test environment would try to be as close as possible to production environment.

System testing is usually performed by testers. It is quite normal that projects have defects in specifications, like missing user story, which leads disagreements or abnormal behaviour of the system. To minimize these kinds of misunderstandings it is recommendable to have tester be included to a project from the beginning and take part to such reviews affecting in planning of, for example, user stories.

Most unique objective of system testing is to give information of the quality of whole system and build trust to the quality and thus, the system. Other objectives quite same as in all other test levels.

Sample test objects of system testing [25]:

- Applications
- Hardware/software systems
- Operating systems
- System under test
- System configuration and configuration data

Typical defects of system testing [25]:

- Incorrect or abnormal system functional or non-functional behaviour
- Failing to carry out end-to-end tasks as defined
- System not operating as specification or user manuals describe
- System not operating as expected in production environment

3.3.4 Acceptance testing

As system testing, acceptance testing focuses on behaviour of an entirety of system of product. Acceptance testing, as system testing, gathers information quality of the system but instead of using the information for release decisions it is used for evaluating readiness of the system for deployment and for end-users. Acceptance testing can be also used to fulfil regulatory and legal requirements like system testing.

One major difference to other testing levels that finding defects is not one of the objectives of acceptance testing. It possible and even normal to find minor defects in acceptance testing but finding considerably number of defects in acceptance testing can be in some cases considered as a major risk to deployment. Of course, critical defects must be fixed before deployment but there should not be any at stage of acceptance testing.

Typical testing objects of acceptance testing:

- System under test
- System configuration and configuration data
- Business process for the whole system
- Recovery systems
- Operational and maintenance process
- Forms
- Reports
- Existing and converted production data

Typical failures of acceptance testing:

- System does not satisfy contractual or regulatory requirements
- Non-functional failures like performance efficiency
- Workflow of the system does not meet business or user requirements

Acceptance testing can have different forms depending on the system. Typical forms included in acceptance testing can be:

- User acceptance testing
- Operational acceptance testing
- Contractual and regulatory acceptance testing
- Alpha and beta testing

Acceptance testing is typically the highest level of the testing levels and the responsibilities of acceptance testing are often customers, product owner or operators of the system. [25]

3.3.4.1 User acceptance testing

User acceptance testing, also known as UAT, is form of acceptance testing where system tested is focusing mostly on user experience or that testing is done by users. Users can validate that the system meets requirements in real or in simulated operational environment. Objective of UAT is to build confidence that the actual user can use the system as intended and can operate so that there is a minimum difficulty, cost and risk when performing business process for the system is designed for. [25]

3.3.4.2 Operational acceptance testing

Operational acceptance testing (OAT) differ from UAT that instead the users are operational system administrators of the system. Testing is performed as close as possible to a production environment. Objective of OAT is to ensure administrators or operators can keep the system operating for the actual users even under exceptional circumstances.

It might be hard to follow the difference between OAT and UAT, but the typical test focus on operational aspect in OAT makes the difference [25]:

- Testing of backup and restore
- Installing, uninstalling and upgrading
- Disaster recovery
- User management
- Maintenance tasks
- Data load and migration task
- Check for security vulnerabilities
- Performance testing

3.3.4.3 Contractual and regulatory acceptance testing

Contractual acceptance testing is done comparing acceptance criteria of contract given by customer. Acceptance criteria should be obvious for both parties before agreeing to contract. Contractual testing can be performed individual testers or by the user.

Regulatory acceptance testing is performed against any regulations regarding to the system such as safety regulations or government regulations.

Regulatory acceptance testing can be also performed by individual testers or the users but can sometimes require testing results being witnessed by regulation agency.

The objective of both regulatory and contractual acceptance testing is to give information of regulatory and contractual compliance of the product and thus, build confidence that those goals have been achieved. [25]

3.3.4.4 Alpha and beta testing

Both alpha and beta testing are used by developers in a state where the product is not yet ready to be officially published. This can be called as commercial off-the-shelf (COTS) software from which developers want to get feedback before the product is actual released. The feedback is collected from potential or existing users, customers and operators.

Alpha testing is typically organized in order that testing is performed at the developing company s site by someone else than the development team itself. Whereas beta testing is performed by testers own location, home for example, where testers are potential or existing users of the product. Alpha testing is done before beta testing, but it is possible to skip alpha testing and go straight to beta testing.

The objective of alpha and beta testing is that the product works as intended under normal circumstances as expected among potential or existing users. Also, it is possible that there are not enough resources for testing, for example with all different environments, and thus, alpha and beta testing may find defects that would have been only found by chance of luck. [25] [32]

3.3.4.5 Testing levels in different testing types

Testing levels typically go from low test levels to high test levels meaning that first tests are done in a level of unit testing whereas last tests are done in acceptance testing. In a rapid development environment, it is possible that some levels overlap with each other.

Even though test levels are separated from each other by the timeline of development, it is not the same for testing types. It possible to execute all the four testing types, functional, non-functional, white-box and change-related, in each test level.

For example, in acceptance testing for a banking application all the four test types can be used [25]:

- Tests based on how the banker declines or approves a credit application (functional)

- Usability tests for credit processing interface concerning people with disabilities (non-functional)
- Tests designed to cover all supported data file structures and value ranges from bank to another bank transfers (white-box)
- After a defect is found and fixed all failed test cases are re-run (change-related)

3.3.5 Maintenance testings

After a successful development process software and systems are deployed to production. Both, software and systems, need maintenance in production in form of modifications, migrations or lastly retirement. Perhaps simplest needs for maintenance findings in operational environment such as critical defect, need of a new functionality or removing of an old functionality. Also, maintenance is needed for non-functional properties such as performance and security.

As maintenance is needed to be done as well as maintenance testing is. Purpose of maintenance testing is to evaluate that the changes are done successfully and run regression tests for the software or systems checking for possible side-effect of the maintenance fixes or modifications. Thus, maintenance testing focuses on testing changes made to the system and running regression test to rest of the system unaffected by the changes. Maintenance can involve unplanned release, for example due to critical defect. Unplanned releases are typically called hot fixes.

Maintenance releases can require considerable amount of testing and thus, multiple test levels and test types. The use of test levels and test types depends on the scope of maintenance testing. Scopes depends on [25]:

- How large of entirety the system is?
- Does the change made regard a large part of the system?
- Degree of a risk the change makes

3.4 Testing (design) techniques

The idea behind test techniques is to assist in identifying test cases, test data and test conditions. Each technique might have some benefits compared to other in different situations and test levels even though some test techniques

are applicable to all test levels. Combinations of test techniques is typically used when creating test cases, to achieve best possible outcome of the testing.

As test techniques can be used to in different activities, like test design, implementation and analysis, those activities can vary from informal to formal depending context of the project. Major parts influencing on formality can be development process, schedule, regulatory and safety requirements. [25]

Testing techniques can be divided in three different segments. These are specification-based testing, structure-based testing and experience-based testing. Test requirements, models, user needs and specifications are used in specification-based testing as primary source of information to create and design test cases, data and conditions. The source code or the structure of a model is used in structure-based testing as main source of information to create and design test cases, data and conditions. In experience-based testing, the experience and knowledge of the tester is used as main source of information for the same gain as specification-based testing and structure-based testing. These three testing techniques are complimentary and using them combined in testing will typically result in most effective results in testing.

Specification-based testing and structure-based testing can be also known as black-box testing and white-box testing. White-box testing can be also known as clear-box testing. Both white-box and black-box refer visibility if the internal structure of the testing target. Internal structure is not visible in Black-box testing whereas in white-box testing the internal structure is visible to the tester. There basically one more major testing technique called grey-box testing. In grey-box testing both specifications and structure of the testing item are used to create and design test cases, data and conditions. [9] [22] [17]

As seen in Figure 3.1 there are multiple different testing design techniques beneath each three major testing techniques. the figure shows only certain testing design techniques and there more of them in each part especially when all techniques are taken into count (not only the design part). In next subsections few of the techniques are explained to get better understanding of each testing technique.

3.4.1 Specification-based test techniques

As mentioned before specification-based test techniques are based on test basis meaning user stories, requirement documents and regulatory requirements for example. Typical distinctive features of specification-based test techniques can include [9]:

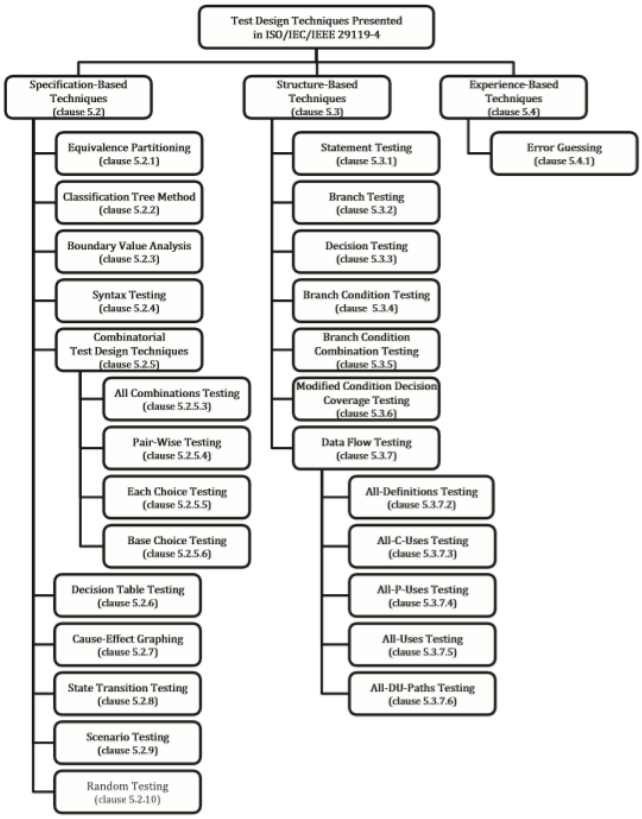

Figure 3.1: Testing (design) techniques [9].

- Test cases, data and conditions are mostly from test basis including software requirements, use
- cases and stories and specifications
- Coverage is measured based testing targets in the test basis
- Created test cases can be used to detect missing information between requirements and implementation of the requirements

3.4.1.1 Equivalence partitioning

In equivalence partitioning data is divided into partitions in a such way that all data members in one partition are processed in the same way. There are checks to valid and invalid values in equivalence partitioning.

To achieve best possible coverage with equivalence partitioning, test cases must try to cover all possible partitions including invalid partitions. At least

one value must be used in each partition. Coverage is measured by the number of tested partitions divided by the number of identified partitions. [9]

3.4.1.2 Boundary value analysis

Boundary value analysis, also known as BVA, is basically an extension of equivalence partitioning where comparable data used are minimum and maximum values of a partition. BVA can be only used if the partition consisting of numeric or sequential data.

A simple example of BVA technique is an input field that accepts a single integer as a value. Suppose the valid range is from 1 to 3. Thus, there are invalid value of too low value under 1, valid values between 1 and 3, and invalid value of too high above 3. Boundary values for too high are 4 and maximum value of the input field and for the low part 0 as the only value. Values 1 and 3 are valid boundary values.

It is more likely to find the incorrect behaviours from boundaries of partition than inside of the partition. Boundary value analysis can be used in all test levels. This technique is typically used to test requirements such as dates and times. [9]

3.4.1.3 Decision table testing

Decision table testing is a method where implementations of system can be tested by using combinatorial test techniques specifying how different combinations of conditions can result in different outcomes. Decision tables are a good tool for recording different complex rules that system must implement. Creating of decision table is quite simple. Tester uses certain inputs of the system resulting to corresponding outputs. These inputs and outputs form rows in the table starting with inputs leading to an output. The values of inputs can be shown for example, with Boolean values (true or false) leading to a certain state in the system.

The idea and strength of decision table to find all important combinations of conditions that might be otherwise go ignored. It is also a fine tool identifying shortcomings of requirements. A full decision table cover all combination of conditions excluding impossible combinations. As behaviour of the system depends on combination of conditions, decision table can be used in every test level. [25]

A simple example of decision table is one made of simple login screen. A simple login screen has two inputs, username and password. With two

conditions the number of combinations is four. Thus, decision table has four rows with different combinations (FF, FT, TF, TT).

Conditions	Combinations1	Combinations2	Combinations3	Combinations4
Username	False	False	True	True
Password	False	True	False	True
Output	False	False	False	Login Successful

From the decision table it is easy to see all four combinations. Also, it quite simple to create a test case for login due to decision table. Ignoring some of the combination can happen and on complex systems they probably will without a decision table.

3.4.1.4 State transition testing

Whenever an event occurs in a system it may respond correspondingly depending on current state of the system. The current state might be result of events happened after the system was initialized. A state transition diagram tries to present all possible states of the software. A state transition diagram also tries to cover transitions between states as well as exit and enter to or from the software. An event can lead to transition and the event is usually provided by user. Such an event can be a button pressed by the user or value input into a field. As the state changes in the program it is possible that that the event can lead more than one transitions and thus there might be guard condition qualifying the event. As the event can change state of the software the change of the state may result in action like an error message.

Simple example of transition testing could be opening of a door. There are to possible states for the door, open and closed. Starting state of the door is closed and to change the state of the door it needs open event. The open event may need a guard condition like checking if the key is corresponding to the lock of the door.

State transition diagram usually contains only valid transitions, but a state transition table shows all valid and invalid transitions between states as well as events and guard conditions. Tests can design to cover normal sequences of states, all states, every transition, every corner case transition and even test invalid transitions. [25]

3.4.1.5 Use case testing

It is possible to derive tests from use cases. Use cases are associated with human users, external hardware, systems or other components and subjects.

Every use case defines some behaviour that subject performs with one or more actors. Tests are designed to test known operations. These kinds of behaviours can be distributed to normal, exceptional, alternative and error handling. [25]

An example of use case testing could be a simple login. Actor is human user and subject is the system. Simple test includes input from actor (name and password) and answer from the subject valid or invalid name or password combination. Also, this simple test could be extended by testing error handling of the system by intentionally giving incorrect password as an input and repeating for as many times as systems closes the application or locks the user.

3.4.2 Structure-based test techniques

Structure-based test techniques (also known as white-box test techniques) is based on internal structure of the test target. Structure-based test technique focuses on detailed design, analysis of the architecture, internal structure or the code of the test target. Structure-based test technique can be used in all test levels.

Typical distinctive features of structure-based test techniques can include [25]:

- Test cases, conditions and data are derived from any source of information regarding the
- structure of the software
- Coverage is measured based on item tested within the target structure of the software like code
- Specifications are often used as a source to determine results of test cases

3.4.2.1 Statement testing and coverage

Exercising the executable statements in the code is the main purpose of the statement testing. Coverage of statement testing can be calculated by number of statement executed divided by total number of executable statements in the test object.[25]

3.4.2.2 Decision testing and coverage

Decision testing ensures that the decisions in the code are executed based on the decision outcomes. Control flows that occur from decision point (e.g. IF statement or CASE statement) are one way to create test cases for decision testing.

Achieving 100 percent decision coverage (decision outcomes executed divided by total number of decision outcomes in the test object) will guarantee 100 percent statement coverage whereas 100 percent coverage in statement testing does not guarantee 100 percent decision coverage. [25]

3.4.3 Experience-based test techniques

Experience-based test techniques relies mostly on the experience of the developers, testers and users to design and execute tests. Experience-based testing techniques are usually combined with both structure-based and specification-based test techniques.

Experience-based test techniques are most useful when other systematic techniques as structure-based and specification-based do not easily identify tests for the target. Test cases are generated from the skill, intuition and experience of the tester with similar projects, software, applications and technologies. As test cases are not systematic and rely heavily on the tester, experience-based test techniques may achieve varying degrees of effectiveness and coverage. It is even possible that it is impossible to measure any kind of coverage with experience-based test techniques. [25]

3.4.3.1 Error guessing

In Error guessing technique occurrence of defects, failures and mistakes is anticipated by tester relying on knowledge of the tester. The knowledge of the tester might consist of:

- How similar applications have operated or how the application has operated in the past
- What are most common mistakes made by developers concerning the application or similar application
- Failures occurred in other similar applications or in the application

Even though error guessing is not systematic testing technique it can be made more systematic by creating a list of possible defects, mistakes, failures and thus design tests for those defects, failures and mistakes in the list. [25]

3.4.3.2 Exploratory testing

When tests are design, executed, logged and evaluated dynamically during testing informally, the technique is called exploratory testing. Result of testing are used to learn about the system and possibly to create more tests for areas that require more testing.

Exploratory testing is mostly used when there are no specifications or insufficient specifications or remarkably little time for testing. Exploratory testing is also useful to reinforce other formal testing techniques.

As exploratory testing is quite vague in the part of scale and time it is sometimes useful to use session-based testing as a guideline for exploratory testing border lining testing target and a time window for testing and having some simple objectives for the testing session. [25]

3.4.3.3 Checklist-based testing

A checklist is created by the tester based on experience and knowledge of importance to the user or understanding of common software failures. Testers design, implement and execute test covering conditions found in the created checklist. That is called checklist-based test testing. As testing moves forward the checklist can be modified and expanded.

Checklist can support most test types including functional and non-functional testing. Checklist-based testing provides a guideline in absence of detailed test cases. [25]

Chapter 4

Testing process

A common set of test activities together basically form a test process. Every software may need test process of its own and thus there is no universal test process that could fit perfectly for all software. Even a well though multi-function test process in any given situation depends on multiple factors. Thus, it could be even said that there as many test processes as there are different software. How, when and which test activities are involved to the test process may be settled in an organization s test strategy. [25] [28] [30]

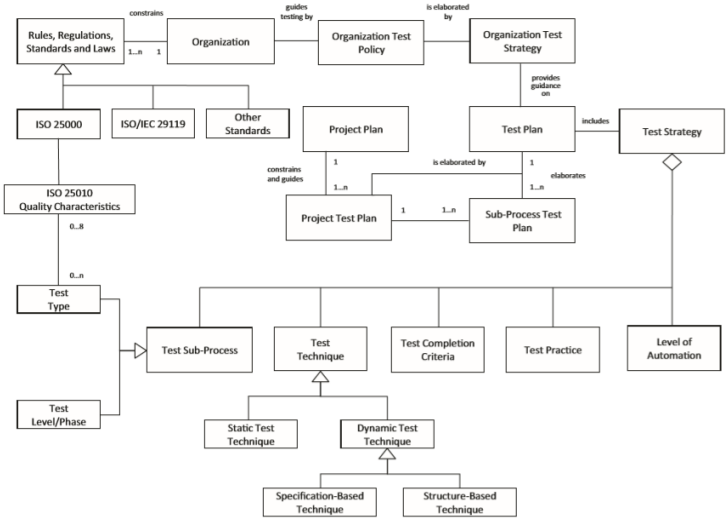

Figure 4.1: Multi-layered test context diagram [7].

Contextual factors that influence composition of test process may include:

- Development lifecycle model and project methodologies used

- Test levels and test types considered
- Risks of product or project
- Business domain
- Operational constraints including:
- Time
- Resources
- Complexity
- Contractual and regulatory requirements
- Organizational policies and practices
- Required standards

Common set of test activities (often implemented iteratively) in test process:

- Test planning
- Test monitoring and control
- Test analysis
- Test design
- Test implementation
- Test execution
- Test completion

It can be most useful if measurable coverage criteria are defined in test basis for any level or type of testing. From the coverage it can be effortless to follow progress of testing. For a mobile application such coverage criteria may include list of supported mobile devices that the application must function on. The coverage criteria may require something from test cases, for example, a simple test case for every different supported mobile device. The number of supported devices tested gives the coverage to product owner. [25]

The life cycle of a software can be divided into three different test process groups in which common set of test activities are performed. These three test processes are Organizational Test Process, Test Management process (known

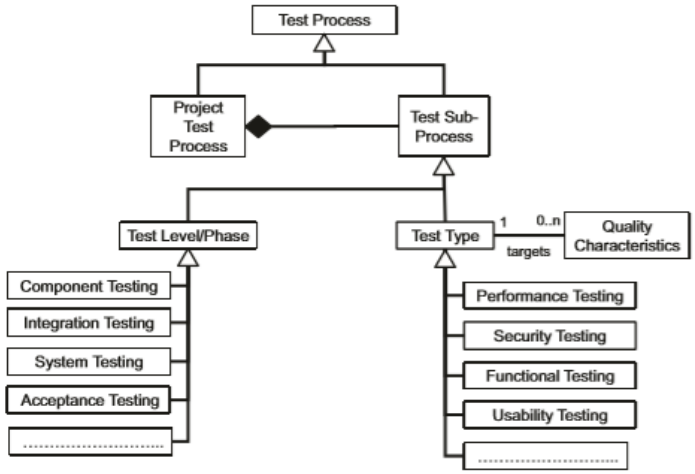

Figure 4.2: The relationship between the generic test sub-process, test levels and test types [7].

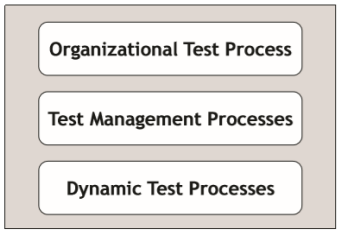

Figure 4.3: The multi-layer relationship between test processes [8].

also as TMap) and Dynamic Test Process and together forming Multi-Layer Test Process model. [8]

Organizational Test Process defines creation and maintenance of organizational test specification like organizational test policies strategies, policies, processes and procedures. Test Management Process defines processes concerning management of testing for a project, test level or test type within a project. Examples of these could be project test management, acceptance test management and security test management. Dynamic Test process focuses on processes for executing dynamic testing. [8]

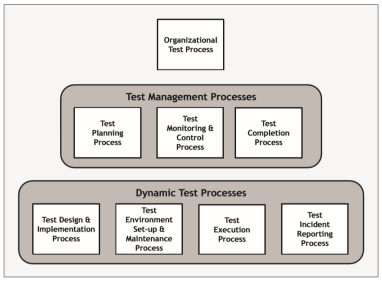

Figure 4.4: The multi-layer model showing all test processes [8].

4.1 Organizational test process

Developing and managing organizational test specifications is the main purpose of organizational test process. Organizational test specifications are not project-based and apply across testing in the whole organization. The organizational test process is generic and thus can be used to manage and develop non-project specific test documents.

Examples of organizational test specifications are organizational test policy and test strategy. The organizational test policy describes the goals, purpose and scope of testing inside the organization. The organizational test strategy is technical document that defines how testing is done inside the organization. [8] [6]

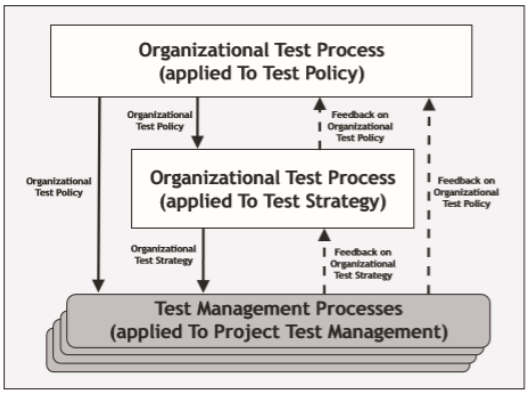

Figure 4.5: Example of Organizational Test Process implementation [8].

Successful implementation of the organizational test process includes [8]:

- The requirements for organizational test specifications are identified

- The test specifications are developed
- The test specifications are agreed to by all stakeholders
- The test specifications are made visible and accessible
- The test specifications are conformed and the conformance is kept eye on
- Updates to the test specifications are made and agreed by stakeholders

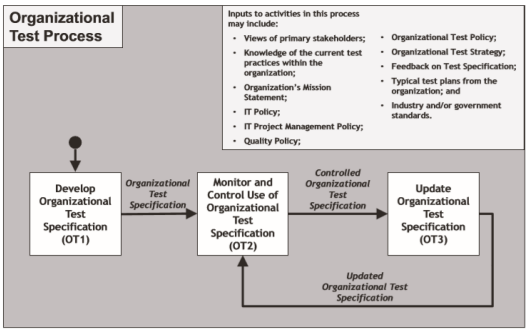

Figure 4.6: Organizational Test Process [8].

4.2 TMap (test management process)

Test management process consist of different parts [8]:

- Test planning
- Test monitoring and control
- Test completion

Generic test management process can be applied at project level (project test management) as well as in different test levels (acceptance test management) and test types (security test management). The project management processes are used to manage testing of the projects as whole based on project test plan. Test levels and test types require test management processes to be applied to their management separately, typically based on separate test plans such as acceptance test plan or security test plan. [6] [8]

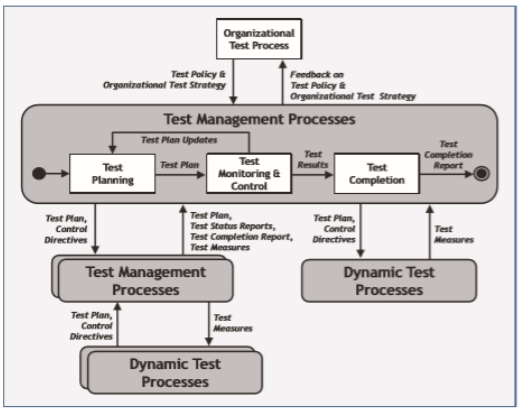

Figure 4.7: Example of test management process relationships [8].

4.2.1 Test planning process

Test plan is developed by using the test planning process. Test planning processes can be used in multiple stages in a project if needed. It can try to cover whole project as project test plan but can occur more specified test level or test type such as acceptance test plan or security test plan. Creating a test plan according to test planning process is shown in Figure 4.8. [8]

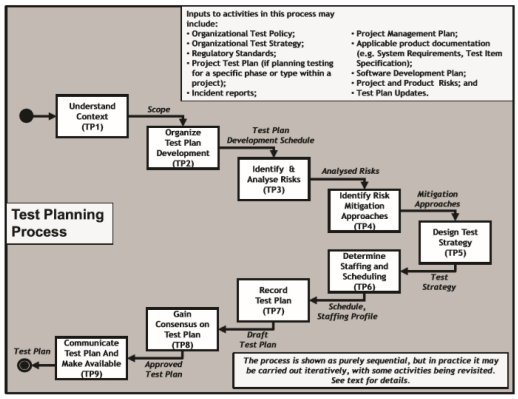

Figure 4.8: Test Planning Process [8].

As more information becomes available during the project such as new threatening risk, test plan is required to adapt. As result during testing it is possible that test plan may need to be modified. Thus, test planning process can be iterative. [8]

The main goal of the test planning process is to develop, record and communicate to stakeholders the scope of testing and approach in testing towards the testing target. Other requirements are mentioned such as resources and environment needed for sufficient testing.

Successful implementation of test planning process includes [8]:

- The scope of project is understood and analysed
- Stakeholders taking part in test planning are identified and informed
- Risk that can be avoided by testing are identified, analysed and prioritised by the level of risk exposure
- Requirements of sufficient testing are identified such as environment, tools and data
- Possible training or staffing needs are identified
- Scheduled activity
- Estimates such as cost, staff and timeline are calculated, and recorded evidence of the estimates is collected to justify the estimates
- Test plan is agreed to and distributed to all stakeholders

4.2.2 Test monitoring and control process

The test monitoring and control process observes if testing progress according to the test plan and other test specifications such as organizational test policy and strategy. If the testing does not progress as planned by, for example, test plan, activities will be started to correct the current situation. The test monitoring and control process can be applied to management of whole test project but also management of single test level or test type.

The purpose of the test monitoring and control process is to determine if testing progresses according to project test plan, test plan, organizational test specifications and manage testing performed at certain test level. Result of successful implementation of the testing monitoring and control process includes [8]:

- The means of collecting suitable measures to monitor test progress and changing risk are in order
- Progress according to test plan is monitored

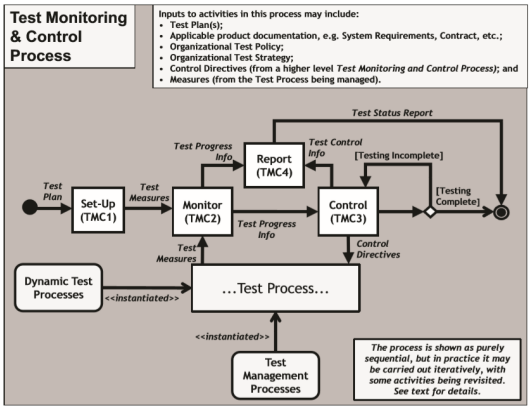

Figure 4.9: Test monitoring and control process [8].

- New and changed test-related risk are identified, analysed and needed activities are invoked
- Necessary control measures are identified and are communicated to relevant stakeholders
- The decision to end testing is approved
- Test progress and changes to the risks are reported to stakeholders

4.2.3 Test completion process

The test completion process is performed when it is agreed on that testing activities are complete. The test completion process can be performed to complete the testing of certain test level, test type or the project.

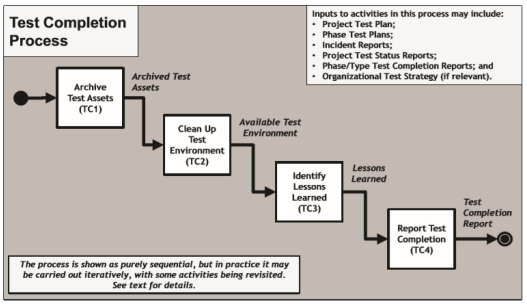

Figure 4.10: Test completion process [8].

The purpose of the test completion process is to leave the test environment in usable condition, make useful test assets available for later use, and record and communicate results of testing to relevant stakeholders. Result of successful test completion process includes [8]:

- Test assets are either achieved or passed to the relevant stakeholders
- The test environment is left in agreed condition
- All test requirements are satisfied and verified
- Recording of the test completion report and approval of the report
- The report is communicated to the relevant stakeholders

4.3 Dynamic test process

The Dynamic test process compared to the management test process and organizational test process focuses only on dynamic testing done in different test levels such as unit, integration, system and acceptance testing and test types like performance, security and usability testing. The main dynamic test processes are [8]:

- Test design and implementation
- Test environment set-up and maintenance
- Test execution
- Test incident reporting

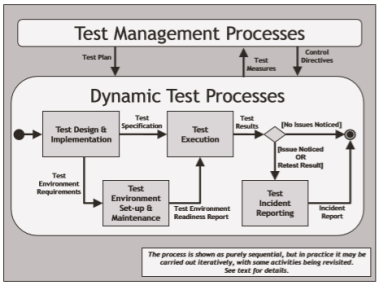

Figure 4.11: Dynamic Test Process [8].

4.3.1 Test Design and Implementation Process

The purpose of the test design and implementation process is to derive test policies and test cases that will be used later during the test execution process. In some cases, such as regression tests, it is possible to use previously designed test assets as a help for the process. Even though test design and implementation process is done before others it is possible to re-enter to back to test design and implementation process for example in case of realising the requirement of new test cases. This process also requires tester to use multiple test design techniques.

A successful implementation of the test design and implementation process includes [8]:

- The test basis for each test target is analysed
- The features to be tested are combined into featured sets
- The test conditions are derived
- The test coverage targets are derived
- Test sets are assembled
- Test procedures are derived

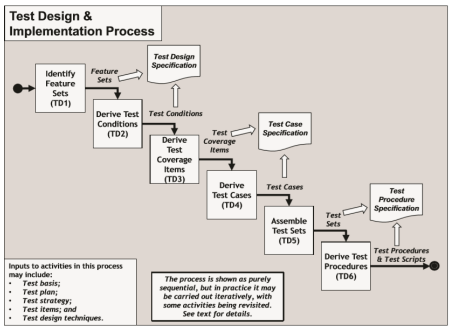

Figure 4.12: Test Design and Implementation Process [8].

4.3.2 Test Environment Set-Up and Maintenance Process

The purpose of the test environment set-up and maintenance process is to establish and maintain the required test environment in which the tests are

performed and communicate status of the environment to the relevant stakeholders. Requirements for the test environment are usually described in the test plan and detailed composition of the test environment usually comes more explicit after the test design and implementation process has begun.

A successful implementation of the test environment set-up and maintenance process includes [8]:

- The test environment is set-up and ready for use of testing
- The status of the test environment is communicated to the relevant stakeholders
- The test environment is maintained

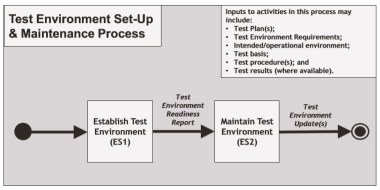

Figure 4.13: Test Environment Set-Up and Maintenance process [8].

4.3.3 Test Execution Process

The purpose of the test execution process is to perform test procedures created in the test design and implementation process in the prepared test environment established by the test environment set-up and maintenance process. The test execution process may be required to be performed number of times because it is normal that test procedures may not be performed in a single iteration (e.g. fixed issue requires a re-entering to the test execution process).

A successful implementation of the test execution process includes [8]:

- All planned test procedures are performed
- Recording of results
- Comparison of actual and expected results
- The test results are determined

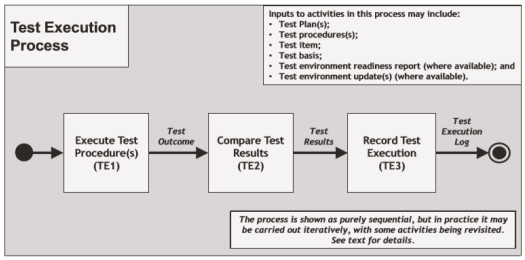

Figure 4.14: Test Execution process [8].

4.3.4 Test Incident Reporting Process

The test incident reporting process is used for reporting of test incidents such as failures and defects. The process in entered whenever a there are failures in tests that require reporting, something unexpected happens during testing or in the case of retest passing.

The purpose of the test incident reporting process is to get information of relevant incidents to the relevant stakeholders. In case of new-found incident, an incident report is required and created. In the case of retest, the incident report is updated. [16]

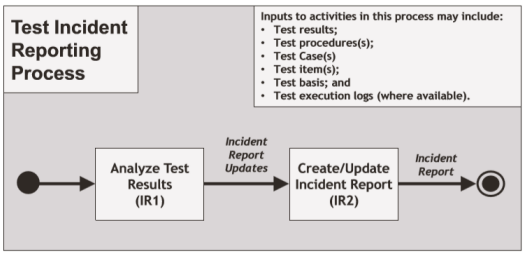

Figure 4.15: Test Incident Reporting process [8].

A successful implementation of the test incident process includes [8]:

- Test results are analysed
- New incidents are confirmed, and report details are created
- The status and details of previously-raised incidents are determined
- Report details of re-tested incidents are updated
- New and updated incident reports are communicated to all the relevant stakeholders

4.4 Improving testing process

After a testing process is established it does not mean it would stay as it is. It is highly recommended that the test process should be under continuous improvement. Just as in improving software process models same applies to testing. Thus, process improvement can be also applied to testing. There are different ways and methods to improve testing of software. These ways and methods aim to explicitly at improving testing process.

The attention of process improvement models in projects are easily focused on the software. Luckily there are test improvement models such as Test Improvement Next (TPI Next), Critical Testing Process (CTP), Systematic Test and Evaluation Process (STEP) and Test Maturity Model Integration (TMMi) which try to solve the problem of lack of attention in process model improvement in projects. [18]

Process improvements are important to the software development and to the testing process. A simple ongoing improvement cycle called the Deming improvement cycle has been used for decades and is relevant for testers to improve testing process (or ant process) even today. The cycle of the Deming improvement cycle goes: Plan, Do, Check Act. [24]

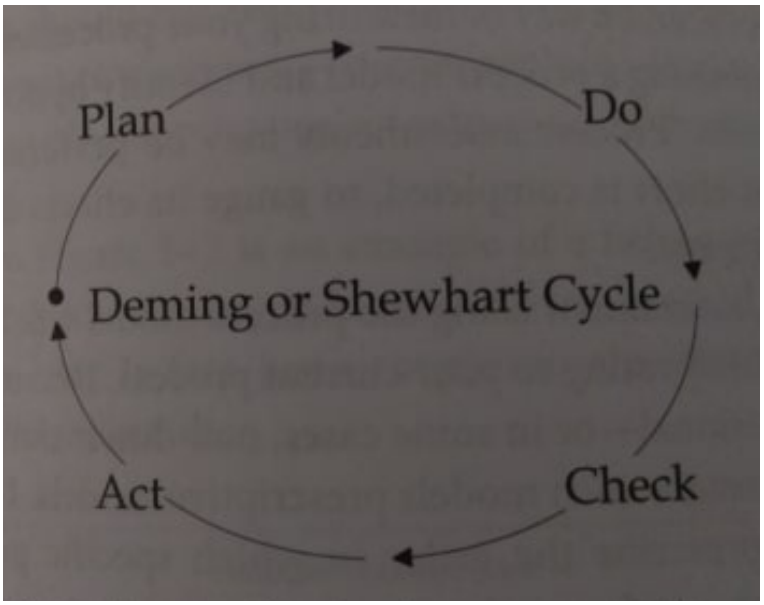

Figure 4.16: The Deming cycle or the Shewhart improvement cycle [11].

The test process improvement models are used to reach higher level of professionalism and maturity in the IT industry. For the need of process

improvement in the testing industry these four (TPI Next, TMMi, CTP and STEP) are some of the recommended processes for improvement. All of these four models give tools to organizations determine where it stands in the terms of the test process of the organization.

After it has been decided that improvement to the test process should be done, the process implementation steps can be defined in the IDEAL model [24]:

- Initiating the improvement process
- Diagnosing the current situation
- Establishing a test process improvement plan
- Acting to implement improvement
- Learning from the improvement program

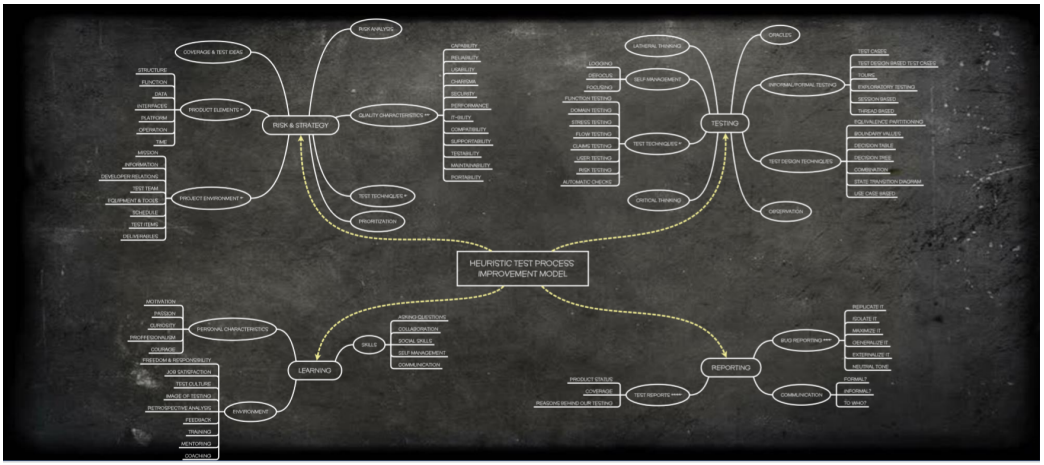

Figure 4.17: Heuristic test process improvement model [29].

4.4.1 Test Maturity Model integration (TMMi)

The test maturity model integration consists of five maturity levels. Each maturity level contains a certain process area and to move to the next level in TMMi 85 percent of the level's goals must be achieved.

The TMMi maturity levels include five levels:

- Level 1: initial

- Level 2: Managed
- Level 3: Defined
- Level 4: Measured
- Level 5: Optimized

In initial level there is no formally structured or documented testing process. Test are usually developed in an ad hoc way during coding and testing is often seen same as debugging. The goal of testing is understood to be demonstrate that software functions as intended.

In second level testing process is clearly separated from debugging. Separation can be done by creating test policies and implementing testing techniques and methods.

In third maturity level testing process is integrated into development lifecycle of the software and documentation is done by formal standards.

In fourth level testing process can be effectively measured and managed at an organizational level to benefit of certain projects.

In final level of maturity levels focus in on optimizing established process. Data from the testing process can be used to prevent defects. [24]

4.4.2 Critical Testing Processes (CTP)

The fundament of the Critical Testing Process (CTP) assessment model is that particular testing processes are critical. The critical testing processes support test teams when critical testing processes are well executed. The CTP model consists of twelve critical testing process. [21]

These twelve critical testing processes of CTP are [11]:

1. Testing
2. Establishing content
3. Quality risk analysis
4. Test estimation
5. Test planning
6. Test team development
7. Test system development
8. Test release management

9. Test execution
10. Bug reporting
11. Result reporting
12. Change management

The idea of CTP is to identify which of these twelve critical testing processes are strong or weak. If it is noticed that some processes are weaker than other some recommendations of improvement for those processes are provided. [21]

4.4.3 Systematic Test and Evaluation Process (STEP)

Systematic Test and Evaluation Process (STEP) is a content reference model. The STEP methodology is based on the idea that testing is ever going lifecycle that begins at planning of the system and continues until the system is retired from use. STEP emphasises testing before coding by using requirement-based testing strategy ensuring that created test cases validate the requirement specification before coding or design of the system is done.

Basics of the STEP methodology include [24]:

- A requirement-based testing strategy
- Testing starts at the beginning of the lifecycle
- Tests are used as requirements and usage models
- Testware design leads software design
- Defects are detected earlier or prevented
- Defects are systematically analysed
- Testers and developers work together

4.4.4 Test Process Improvement Next (TPI Next)

As in TMMi, Test Process Improvement Next (TPI Next) has maturity levels. But instead of focusing on maturity levels of TPI Next, the process is based on 16 key process areas that are evaluated by the maturity levels.

The 16 key process include [11]:

1. Stakeholder commitment

2. Degree of involvement
3. Test strategy
4. Test organization
5. Communication
6. Reporting
7. Test process management
8. Estimating and planning
9. Metrics
10. Defect management
11. Testware management
12. Methodology practise
13. Tester professionalism
14. Test case design
15. Test tools
16. Test environment

TPI Next maturity levels for these key process areas are [11]:

1. Initial
2. Controlled
3. Efficient
4. Optimising

4.5 Communication in testing

Communication in testing can be mostly shown through by bug reports created by the tester. Interacting with programmers as a tester is an important aspect of the role of tester. Without proper interaction with programmers testing is not as efficient as it could be. Defect reports can be dismissed, testers are not informed timely enough, testers work is not appreciated by programmers, testers are not trusted, and many more problems can occur if the communication between testers and programmers is not reasonable.

Programmers are experts of the machine. As an expert of something does not make some same as the target of expertise, in this case programmer seen as a rational machine-like person. Programmer is a normal person with feelings who can deeply care for his work. Tester is the critic of the software or system and thus critic of the work of the programmer. Being honest, sensitive and diplomatic in communicating as a tester is an important skill to master. [15]

4.5.1 Understand how programmers think

Programmer and tester operate in two totally different condition. Both have a different role in software development. Thus, both have different perspective towards the software. To understand more about world of a programmer it is important to understand these perspective differences between tester and programmer. Perhaps the best way to really understand the perspective of a programmer is to be become one for a while in a career of software tester.

Five points are made to understand programmer better [15]:

1. Usually programmers are specialized in some certain subsystem or module
   (a) This often means that the tester might have better understanding of the integrity of the operating system
2. Programmers have their theory of the operating system
   (a) Tester reports a bug that can not happen in the theory of the programmer even though it is an obvious that is reproducible bug
3. Programming is complicated activity
   (a) Programming is on going solving of different problems and requires deep concentration. Thus, programmers might be impatient with interruptions.

4. Programmers often struggle with difficult situations

   (a) It is common that a tool used for software development is buggy or an update of a component brakes something that must be fixed. Work environment of programmer can be full of interactions

5. Programmers dislike routine work and create automation and script for help of their work

   (a) As a tester automation is great thing, but manual testing is always needed and is usually more effecting in finding defects

4.5.2 Develop trust of the programmer to testing

Creating a hostile relationship as a tester with programmers developing the system from the start is not a good idea. Being an effective tester requires information and one of most important source of information can be the programmer. Having a healthy and trustworthy relationship with programmers leads to a possible information of early plans, drafts of the design document and early prototypes.

Earlier the tester has the information the better. Engaging early to development project as a tester requires the tester to be sensitive and helpful. Tester will be dealing with draft products and must know that these products are not complete and thus require different kind of feedback than normally required from a tester. Programmer still want information if their draft has critical error that have gone unnoticed. As the development process develops giving worthwhile feedback all the way till the end will give the tester a trust they deserve. [15]

4.5.3 Provide service to the programmers

Offer programmers service as a tester. This builds trust and gives impression that programmer can cooperate with the tester. Basically, everything a tester does should be a service made for the programmer. For example, some services that tester could offer can be testing private build and protypes created by the programmers, setting up testing environment that the developer can also use or testing third-party components.[15]

4.5.4 Integrity and competence will demand respect

In the simplest the job of a tester Is to report problems of the developed system. Programmer might have difficult times understanding some of the

problems especially when the report is about user experience. When the programmer does not quite get the problem, the tester will be delivering an unpleasant message. As a tester finds these unpleasant messages it is important that they are delivered. Finding credible problems and reporting them accurately will lead to respect from the programmers.

When reporting problems there few tips for the testers [15]:

1. Report problems crisply
    (a) Report a defect step by step leaving all unnecessary steps and comments from the report
2. Base the findings on the behaviour of the product
    (a) Tester is not the expert of the internals of the system but can be the expert of behaviour of the system
3. All defects are not reproducible, show the steps tried to reproducible it
    (a) It is important when something irreproducible happens to investigate. On critical happening sometimes reporting the irreproducible defect can be helpful
4. Deliver critical findings directly
    (a) It is good give notice to a programmer before uploading a critical report for all to see
5. As a tester do not pretend to know about things a tester does not know
    (a) Tester can guess what causes the problem but, in most cases, does not know the exact cause of the problem
6. Exaggerating bug reports is not necessary
    (a) If a tester sees a problem the tester should report it and escalate only if necessary
7. Integrity and competence of tester are important
    (a) Tester without integrity is not taken seriously and without integrity there can be no competence

4.5.5 Focusing on target, not the person

When reporting problems tester should focus on the problem, not the programmer. As in every job someone is better than other in what they do, and everybody has different strengths. Same applies to programmers. Tester might know that a certain programmer does more mistakes than another but there is no reason to pinpoint the problem to the programmer. Making reports more about the programmers will leave the tester in positions of distrust and thus, less information and making the work of the tester less effective.

Also, it is possible that there is so called problem programmer. The programmer has always a lot of defects in the code and does not care so much of the work as others. Still pinpointing the programmer as a problem is not a solution. Sometimes the tester must trust that the management sees these kinds of problems and acts on them. [15]

4.5.6 Asking question about the work of programmer

Most programmers are invested in their work and many testers seems to have problems of getting information out of the programmers. As many might not expect programmers are quite willing show and talk about their work. Tester must show a genuine interest to the work of the programmers to perhaps get to the information the tester is after. Asking questions about technical parts the tester does not understand will increase the competence of the tester and increase the trust of the programmer to the tester that tester understand the developed product also technically. As tester ask question and get answer from the programmers a trust bond between the tester and programmers is formed. After the trust has been achieved tester might have easier way of collecting any information. Tester might be able to have conversation from the starts which starts with question about error handling. Without a proper question and conversation before it, it might feel tester just wants something and is not really interested about the work of the programmer. [15]

4.5.7 Programmers want to help with testability

Programmers want their product to be good. One part of the product being good is how good is the testability of the product. Programmers want assurance of the quality of their product which mostly comes through testing. Programmers do not always know how to make the testability of the product better. Tester should be able to tell how to improve testability for the product.

Tips for the tester to get better testability [15]:

- Speak the same language as the programmer

  Understanding the code and the design documents helps a lot
- Ask early enough
- Be realistic, do not hope moon from the sky

Chapter 5

How to create the right testing process for a software

Tools for creating a testing process are introduced through out chapters 2, 3 and 4. Testing process alone is not enough and can fell sort on it purpose if it is not well thought and targeted. Thus, it is quite important to understand that the testing process must complement the lifecycle of the development process.

Software testing alone has no actual value. Software testing has only value if it delivers value to different parts of software development lifecycle. Thus, it is utmost important that testing fits the software development lifecycle and supports it. It does not matter what kind of lifecycle the development process uses, testing must try to support it. As a fact each different software development lifecycle model effects significantly how the testing is done. [12]

5.1 Testing in the software development lifecycle

There are multiple different known software development lifecycles. Most known models are V-model and Agile model. Newer models are spiral model, rational unified process and rapid application development model. Thus, a testing must at least be aware of the different types software development lifecycles and be ready to adapt into those different styles of development.

In order the testing to be successful in any project, the tester must understand what will be needed, when it is needed and for whom it is done for. Also, projects participants and stakeholders might have their own expectations for the testers. Some might want to know only the critical issues discovered, others might want to know every minor issue found and others

might want weekly reporting of the quality of the product. As every part of these can shape the form of testing process it is also important to understand that an agile project in different organizations can be totally different agile project. For a simple example an agile project in an organization of under 50 personnel is probably enormously different from an agile project in multinational organization with over 1000 personnel. [12] [27]

5.1.1 Sequential model

Sequential models are called sequential for the reason of the entire system is being build only once, meaning that the building happens in sequences of design, implement and testing. Building the product in sequences means also that there is basically no overlapping between sequences. Each phase of the project is completed before the next phase begins. The most common Sequential models are waterfall and V-model. [11]

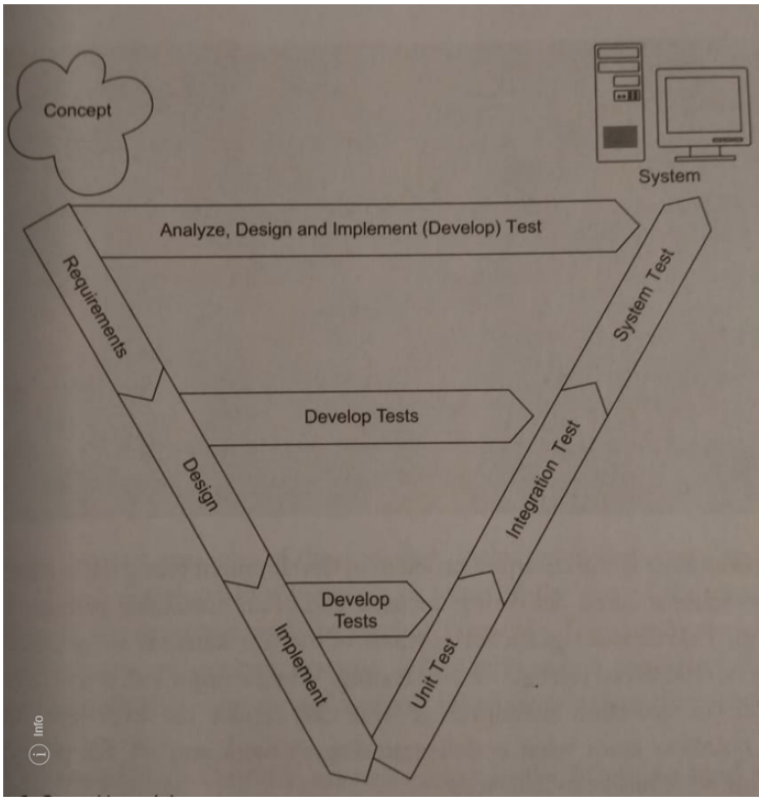

Figure 5.1: V-model [11].

The sequential models are not the easiest for testing and thus the lifecycle

of sequential model causes certain problems for testing. Basically, all the problems involve the problem schedule of the project in the form of strict deadlines of the phases and release date.

First problem for testing is that the deadline after testing is release date. Release date is usually something that cannot be changed and thus there is always a trade-off between the quality of the product and release date. The time during the end can be quite hectic depending on time reserved for testing and starting quality of the product. Also, test manager usually must name one tester that approves that product is ready for production. This situation for the one tester can create a lot of pressure from multiple co-workers.

The second problem is that as the project is sequential there are deadlines for everything, and they are not the most flexible. This can lead to situation where development groups are pressured by the sequential schedule and deliver unstable systems for testing due to lack of time to test their own production. Giving unstable systems for leads to situation where testing is not effective and can be retroactive unit testing that should have been done in the development phase. As the testing is not effective and the time is limited there is again problem in the trade-off of the quality and release date. Third common problem is that as everything is sequential testing is not informed of the need of testing after development of the product is done or the testing team is invited to late to the party. This leaves very little time to plan the testing and being organized. In these scenarios testing easily becomes ad hoc like testing where there is no clear image of the overall coverage thus no clear understanding of the quality of the product which can mean that there was no value in the testing. [11]

5.1.2 Iterative models

In iterative models (known also as incremental models) the model consists of iterative chunks of building the system and testing it. Examples of iterative models are Rapid Application Development and Rational Unified Process. [12] In one iterative chunk there can be steps of analyse, design, develop and test. After the chunk is done it is check if the system is ready. As the system is not ready a new iterative chunk is started. The chunks consist of different functions and capabilities of the system and can be in order based on a risk, for example. The size of the chunks of iteration can vary tremendously between each other. [11]

In iterative model it is normal that there so called fully integrated and functionally operative system much earlier than in sequential model. As the product is not polished and might work only in functional level, this

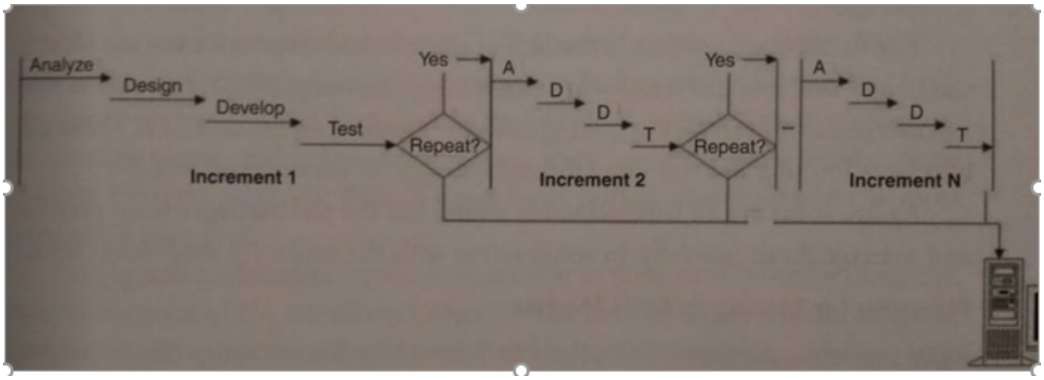

Figure 5.2: Iterative model [11].

can help the testing a lot as testing can be started earlier and some critical functionalities can be tested. As this seems to be great for testing there are also problems in iterative model. [11]

There are two common problems with iterative models for testing. As everything is iterative it means that the packets that testing gets can be quite a lot compared to other models. As in every iteration it must be confirmed that integration of the system is intact which means regression testing all the previous increments. Luckily it is possible in most cases to automate these regression needs. Second problem for testing and for the developer team in iterative model is time to fix defect found during testing. As the project is iterative after development team gives the system for testing, a new iteration is started. This can lead to situation where testing in done in previous iteration in standpoint of the developers and all defects reported are from the past iteration. In the worst-case developer have no time and no interest fixing the defects. This effects highly in the value of testing if no one has time reacts to defects and act on them. [11]

5.1.3 Agile model

Agile models have a lot in common with iterative model. Agile model consists of iterative chunks, but the difference is that the chunks are very short time depended sprints. Sprints are two to four weeks long and include entire team, including testers, in development process. Changes to the system are allowed at any time of the project and adjustments are made to scope based on changes. Example of the agile model is scrum process where the team has daily meetings where information about the progress is shared and each iteration is called sprint with a goal at end of the sprint.

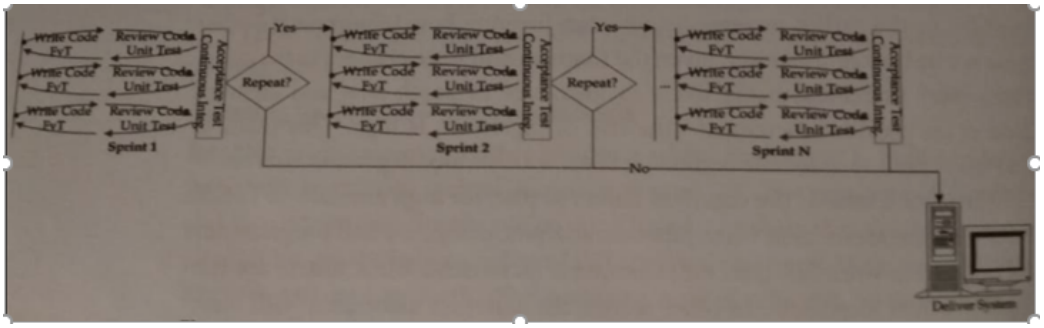

Figure 5.3: Agile model [11].

Testing has similar issues with agile model as in iterative model. But as the agile model is is more fast paced in the iterations are the found issues more relevant. It is common that in agile environment tester can be closer to the development team than other models, but it can vary considerably inside different organizations.

Agile methods create quite a few challenges for at tester [11]:

- Volume and speed of change
- Trying to remain effective in short iterations
- Increased regression risk
- Inconsistent unit testing
- Unsatisfactory test oracles and shifting test basis
- Possible problem of meeting overload
- Siloing inside sprint teams

Agile methods also create opportunities for the tester [11]:

- Automated unit testing
- Static code analysis
- Code coverage
- Continuous integration
- Automated functional testing

- Requirement and test reviews
- Reasonable workload
- Control of technical debt

5.1.4 Spiral model

In a spiral model prototypes of the system are used as iterations to design the system. Spiral model lifecycle goes from another prototype to the next. The lifecycle of the spiral model goes through a spiral consisting of design, prototype, testing and redesigning of the prototype. This spiral is repeated until all risky design decisions have been proven or disproven by testing. Spiral model is not commonly used and the reason for it is that the spiral model is suited best on the largest and most complex projects. One example of these kind of projects is United States missile defence system.

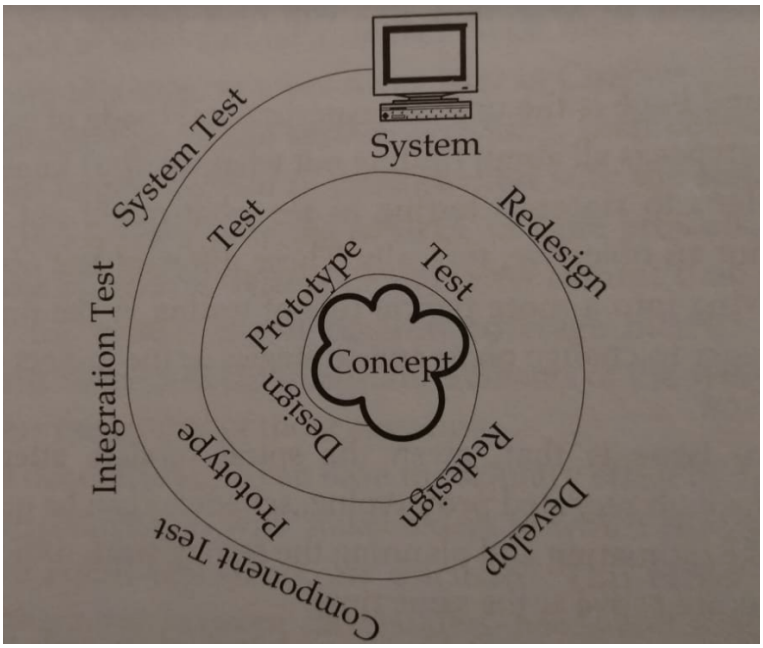

Figure 5.4: Spiral model [11].

As in all models spiral model has challenges for testing. Firstly, there will be many prototypes of system which each has to be tested but the differences are that prototypes are not some iterations. A prototype can be a totally different creation. This mean that there can be enormous differences

between each prototype which will make the comparing and creating the test data difficult and planning of the testing as a whole. Secondly, testing in early stages is mostly experimental. Testing in this stage is more about finding what is not known at the stage, not about testing itself. This leads to situation where testing must be very flexible. Thirdly, schedules in spiral model are usually quite unpredictable. This makes again planning of the testing difficult especially in estimating work hours of testing used for the projects. [11]

5.1.5 Model of coding and fixing

This is more as a sarcastic model that appears in some cases and is regrettable. In this sarcastic model of code and fix project has no real development lifecycle model. At start of the development it is possible that know one really knows what the system should look like. Instead of creating protypes or doing excessive design the development team begins to assembly the final product from the start. An example steps of these model can be that programmer writes code and debugs it a little bit but does not bother with unit testing. As something is ready it shipped for tester with no real meaning. As such tester usually finds multiple defects from the commitment and the programmer fixes them on the fly to the testing environment. This will lead easily to a situation where all the fixes are not committed to the code repository. All these mistakes in coding and communication are repeated multiple times. Finally, the model of coding and fixing, which mostly used by little start-ups (only a few people), leads to exhaustion of money, time or patience by the project team size of dozens or even hundreds. [10]

5.2 Risk definition

Risks are undesirable effects with negative effect to an outcome. Risk can be defined as a problem that might happen and thus decrease perception of the products or projects quality by, for example, user or a stakeholder.

There are two types of risk in testing. First type of risk is a quality risk or also known as product risk. Quality risk, for example, can be possible reliability defect causing failure inside the system and crashing it. Second type of a risk is a planning risk. Planning risk includes everything project related but not the product itself. Planning risk is sometimes called project risk which in some cases is more fitting. An example of planning risk shortage in the staff carrying out the project. Shortage can lead project delays and other problems.

As testing can be risk based it is important to identify both planning risks and quality risks. To identify both kinds of risks there are different techniques to be used during the project [13]:

- Expert interviews
- Independent assessments
- Use of risk templates
- Project retrospectives
- Risk workshops
- Checklists
- Past experience

5.2.1 Cost of bug

It has been mentioned that the testing itself does not have economic value. But a major bug can have huge impact of in the economic value. As quality assurance minimises the risk of these major bugs in the software quality assurance has an impact to the economy of a company even though it is hard to evaluate. If the company is purely a software company, it possible that even one major bug in the right time and right place can be the beginning of the end for the company. It is possible that there is price tag for a bug, for example, if a price of products are wrong and the products are sold without possibility of refund from the buyers. One common costly defect nowadays is a defect that requires service to be out of use for a period of time. As service is unavailable, for some service, it can be evaluated how much does it cost for every second that the service is down. Thus, it is quite easy to understand that a bug can cost a fortune for the company.

Finding and fixing a bug in different stages costs different amounts money. Easily put the earlier the bug is found cheaper and faster it is to fix. The later the bug is found and fixed the more it can cost. This is one of the reasons that testing should be included as early as possible to projects. [20]

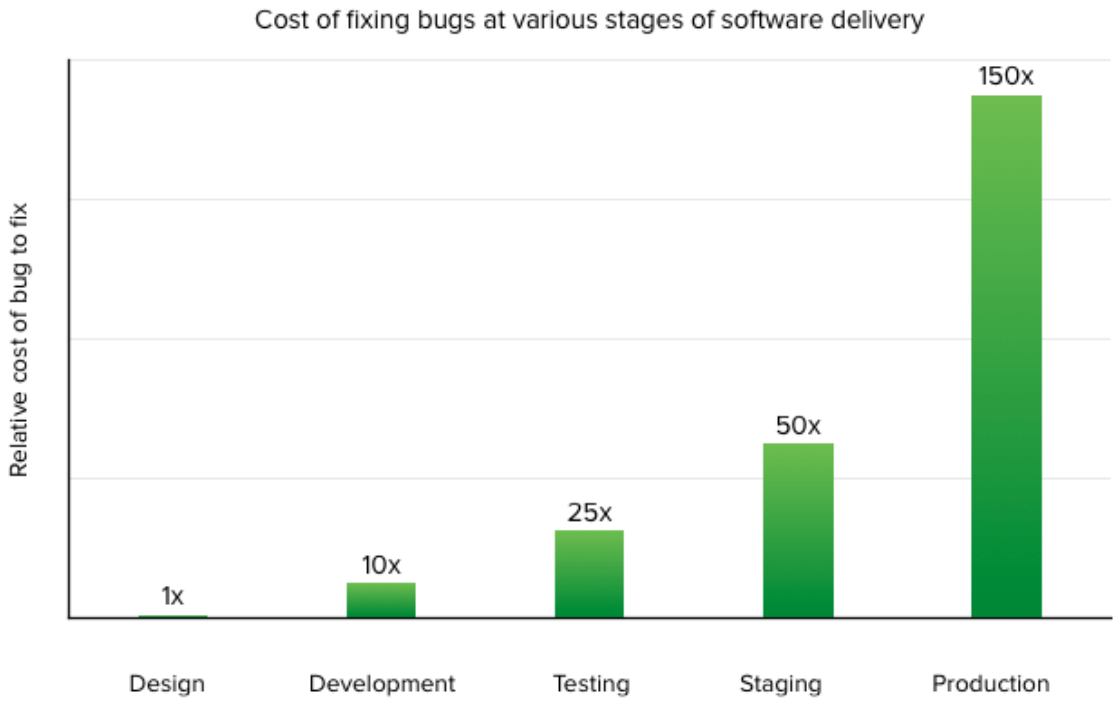

Figure 5.5: Cost of bug in different stage [14].

Chapter 6

Case study: Testing of two different slot games at Veikkaus

Case study focuses on testing two different projects. Both projects have similarities as they both are slot games created at Veikkaus. The two projects used for the case study have been selected so that there are differences between them. Another slot game project being quite simple game with focus only on the game whereas the other slot game has features needing a lot of development of supporting systems of the slot machine itself. Thus, another being a game project and the another more like organization cross-sectioned project where multiple cells of the organization are needed. As for testing these both projects appear as the same, a slot game project. This leads to situation where both projects start with same testing process designed for a slot game project as it can be already figured out that organization cross-sectioned project might need a different testing project than a simple slot game project.

In this case study we will introduce different common tools used to help testing, introduce testing process used in slot game projects as well as development life cycle of slot machines. After that we will look to the testing of both slot game projects and in the end do a comparison between the two projects and conclusion of the two slot game projects.

6.1 Testing tools used

Testing requires tools to be efficient. Testing tools are something which helps in certain areas of testing. These areas of testing can be communication in testing, test case management, documentation of testing or test automation. Tools can be quite simple, for example, for communication it can nearly

anything making communication between the developers and testers easier and more efficient.

In this chapter of case study we will introduce few main tools used in testing slot game projects in Veikkaus. These tools are Testrail, Jira, Slack, Confluence and Robot Framework. Testrail focuses on test case management and creation of test runs. Jira, known as a project management tool, provides perfect way to report defects. Slack is simple communication software making communication between the developers and testers easy. Confluence is a collaborative software making documentation easy to access. Robot Framework is a test automation tool that enables corner case testing and enables making of tools to make testing more efficient.

6.1.1 Test case management

One of the most important part of testing is to have of test cases to help testing and keep track of what have been tested and create coverage through the test cases. There are many ways to create test cases, but the test cases must be stored somewhere and creating test runs might need an extra effort. As it is not ideal to have a text document stored somewhere containing a stack of test cases it is important to have a tool for creating and storing test cases as well as have an easy way of creating test runs from existing test cases. The tool used for test case management in testing of slot games in Veikkaus is Testrail.

6.1.1.1 Testrail

Testrail is modern test case management software made for testing teams and development teams as well. Testrail is web-based licenced software making it easy to start and access for anybody. With Testrail it is efficient to manage test cases and test runs. With a single look the progress of test run or runs can been seen which gives indication of progress of testing. New test runs can be created from made templates with few clicks. All test cases can be divided into folders making it efficient to find the needed test cases. [5]

6.1.2 Communication management

Communication can be done in multiple ways and most used way in projects is talking things through. In testing it is important to communicate all defects, malfunctions and errors as clearly as possible. Also, testers might need the insight of the developers from time to time to continue testing efficiently. As reporting defects is important and must be as clear as possible the best

way is to have some tool where tester and developers can see all the defects as precise as possible. Jira is the tool used in testing slot games in Veikkaus for reporting defects. As communication with team cannot always happen face-to-face and people are occupied at different times, it is important to have communication channel which always open and multiple people are able to answer questions. Slack is the chosen tool for open channel communication.

6.1.2.1 JIRA

Jira is software development tool used for agile environments. Jira is built to plan, track, report and release software by using tracking items with different purposes. In planning phase stories and tasks are created as a backlog for a project and from the backlog teams can create sprints. Every Jira ticket containing different information can be tracked by using different state decided by the team. Example states can be open, in progress, under review, final approval and done. From the states of Jira tickets it can be quite easy to evaluate the progress of a sprint or the project. As for testing creating defects under the project, same values apply than to normal tickets. Defects have states and it can be monitored are the defects being reacted to. Jira is also licenced software product. [2]

6.1.2.2 Slack

Slack is a real-time collaboration application for work. Slack can be used as any other real-time chat application. In Slack different channels can be created for example for different projects, teams and departments. Thus, only the ones needing the information get it. All conversation in Slack are searchable by everyone in the company which make it easier to find, for example, answer to an old problem already discussed in Slack. Integrations are big part of Slack. Slack has multiple ready to go integrations, but it also possible to build your own bots or integrations to Slack quite easily. Slack enables to communicate beyond company. It is possible to invite people outside the company to desired channels in Slack. Thus, it is possible to communicate with clients, vendors or partners on real-time. File sharing, voice and video calls are also supported. [4]

6.1.3 Documentation management

As in every testing related doing, documentation of testing is done. Depending on the project the documentation needs can be quite different. Some projects might need precise and comprehensive documentation whereas other

might need minimal documentation. In testing of slot games Confluence is used for documentation of testing in Veikkaus.

6.1.3.1 Confluence

Confluence is collaborative software published by Atlassian in 2004. Confluence is marketed as a company software as licenced software as a service. Confluence can be seen as an open and shared workplace for a company. Instead of using, for example, shared data storage via network drive anyone can easily access Confluence and find rather easily documentation they are looking for. Confluence is used by a browser and creating documentation to Confluence does not require any skills that updating a normal website might need. In Confluence anyone can create pages containing information and pages can sorted under different locations in Confluence making it easier to browse through different documentation. Documentation can be shared via single link to Confluence and also documentation can be given feedback directly to pages the document is created in. Confluence has many inside build tools for making certain documentation from simple templates to different macros. Also, other Atlassian products like Jira are integrated to Confluence making the documentation more efficient. [1]

6.1.4 Test automation management

As for nowadays software test automation is part of nearly every software development in some way. For slot games test are mostly done manually, but most of the test require help from test automation tools built to make testing more efficient and reasonable timewise. Testing of slot game has large number of corner cases based on random number generations. Thus, most of scenarios are too unlikely to happen in normal circumstances and that is why test automation tools are important. Test automation tools enables testers to go through all possible but unlike corner cases of a slot game. Also, a certain normal situation can be generated through test automation tools.

6.1.4.1 Robot framework

Robot Framework is open source automation framework. Robot Framework is mostly created for acceptance test driven development and robotic process automation. Robot Framework uses keyword-driven test automation which makes it easy to use even for novices of test automation. Robot Framework is coded using Python and has multiple build in test libraries implemented

with Python or Java. Robot Framework was developed by Nokia Networks but is nowadays sponsored by Robot Framework Foundation. [3]

6.2 Testing process used for slot games in Veikkaus

Development of the slot machines, the slot machine software and all that runs on the slot machine is done by the development teams in Veikkaus. This means that basically all information about the software of slot machines of Veikkaus can be found inside the company. Basically, everything is done inside the company which makes it quite unique position for testing. When everything is inhouse design it is can be very efficient for testing as solutions and problems to defects can be reacted as fast as needed and there will not be lack of information to find solution for defects. This means that the basic testing process can focus more on the other parts than raw testing such as helping in design and asking more quality questions than reporting distinct defects.

6.2.1 Development life cycle of slot machine

To understand fully timeline of testing process of slot game it is essential to understand development life cycle of the slot machine. In the world internet creating a version update to webpage can be quite simple. The release of a version can be done in few seconds and then it is ready to use for all users. But for a operating system, like Windows, it is not as simple as that. Probably the version must go through some piloting and installing the update is solely dependent end user. Then there are devices whose updating can be much more complex, like vending machine. Slot machines belong to this last group where every device needs to update itself separately. Thus, it is not the fastest process to update all 20 000 slot machines of Finland to new version. On these new versions the new games are included which makes the release process of slot machines important when creating the testing process of a slot game.

The releasing process is quite simple. Start of the month a new version is released even though there would no slot games and minor modifications. Rarely if critical defects are found from production hotfix is required which breaks the process a little bit. For testing this means that there is always monthly release which requires acceptance testing and piloting before the actual distribution of the software to all 20 000 slot machines.

Acceptance testing starts one month before the target release date. The acceptance testing itself takes 2 weeks or less depending on amount of changes made to system of a slot machine and how much of the changes have been tested in advance. As defects are found during acceptance testing it is decided does the defect need to be fixed to the ongoing release. If so, a new iteration of release is made with wanted defects fixed in it. After the acceptance testing is done, the piloting of the latest accepted iteration can begin. Piloting starts with few slot machines rapidly increasing the number of piloting slot machines to hundreds to thousands. Finally, all slot machines are updated to the latest version before new month.

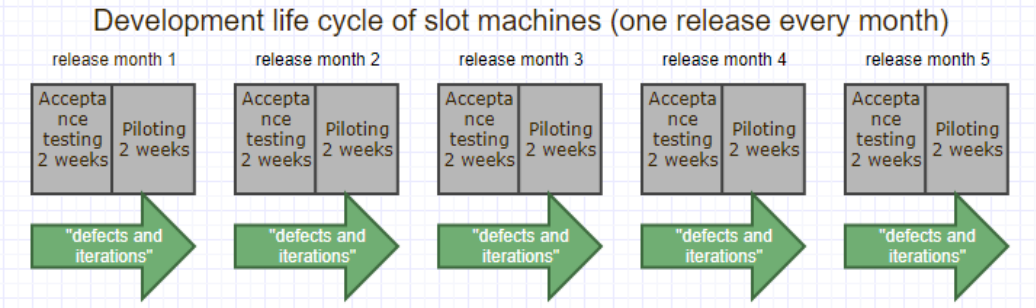

Figure 6.1: Release cycle of slot machines.

6.2.2 Testing process of a slot game

Every slot game project is different from each other. Team members can be different, the game itself can be totally different, a new producer and other possible variables. This does not mean that there should be a different testing process for every project. The testing process of a slot game is created to help testing any slot game project. The testing process is tightly bound to the development life cycle of the slot machine and game itself.

The game has a certain development life cycle depending is it going to be a big release or a smaller one. The big game release has more time in the development life cycle than a smaller one but for the most parts all the game projects have the same ingredients in the development life cycle. The game projects have four clear stages in the development life cycle. First is the designing of the product. Second is developing the product. Third is polishing the product. The fourth is to fix all possible defects that were not fixed during development or polish phase or found in integration testing performed after the deadline for development. Every four stages include

different sprints and entireties, but it does change how the projects proceeds. For a testing process these four stages are the most important key points.

The testing process starts at same time as the project starts. It is usually helpful to have testing participate from start of the project. The testing process of a slot game can be distributed in to three different stages. These three stages being design, agile testing and integration testing.

First stage is starting the project with the development team participating in design and other planning through out the project. In the design part static testing can be done as well as risk analysis. Different risks and findings can be presented to the development team during different meetings, like daily meetings or sprint plannings. Starting the working with the development team from start will create healthy relationship between tester and developers and will create trust between each other.

As the team starts to develop the game, testing is not yet required. Thus, it is important to wait that there is something to test that will benefit the development process. As there are functionalities to test and the development team feels that some amount of testing is required the agile testing can start. In agile testing environment where the product can change in a day to different direction it is important to understand what relevant information to the team is and what is not. This second stages continues through development phase and polishing phase meaning that the mind set of tester must change to stricter as time passes. In start of the development process it is important to only ask question about the product, give opinions when required and familiarise with the game. As the development proceeds and the game is playable and the mechanics are locked the questions can become more precise closing to defect like questions. As the game is ready for the polish period or little bit before that tester can run test runs and start the actual testing of the game with criticism. At this point it is still important to ask questions because the product is not ready, but it is fine to do official defects of the product that will be hopefully fixed during polish period. During this last phase of agile testing of the product information about the quality of the game should be gathered and documented. This information is relevant when starting the integration testing for the game.

Integration testing is done after the development the game is finished. The idea is that the game has no featured yet to be completed or any other task that are undone. In the period of integration testing the purpose of the development team is not develop the game anymore, but to react reported defects and fix remaining defects. In integration testing the purpose of testing is to run full scale test regression for the game and integrations coming from the slot machine and verify all fixed issues. In this stage there can be lot of question to team, but usually if something is worth of question from

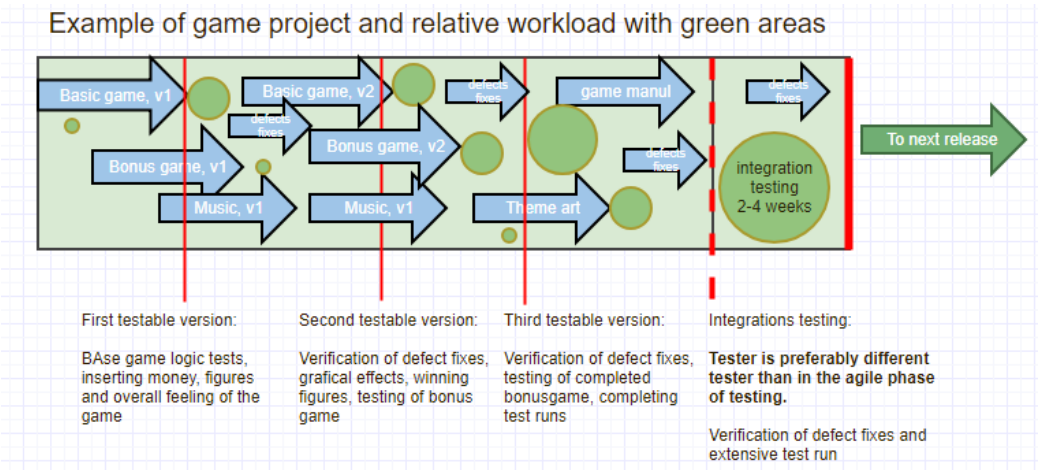

Figure 6.2: Example of a slot game project testing

completed product it might be as well a defect of some kind. In integration testing it is important to report all defects, malfunctions, errors and anomalies with minor threshold.

After the integration testing is over and the product is ready it is appended to development life cycle of the slot machine. In the next acceptance testing of the slot machine the game is included to the release. In the acceptance testing the game is not the priority, but it will get some regression.

6.3 Testing of a minor slot game

The first project we are looking into is a slot game consisting only of the game, meaning that there are no need of upgrading the core systems of the slot machine and background systems supporting the slot machine. The project consists only of developing the game and thus making it simpler to test and develop than a cross-section project requiring a multiple teams working together. The game itself is a normal poker game with a twist of possibility to win free games during normal rounds.

6.3.1 Test plan and strategy

Object of testing is a poker game created to slot machines owned and created by Veikkaus. It is normal poker game with a little exception of winning free games with kings. The estimate amount of work required total from testing for such a product is roughly five working months. The five working

months includes all sectors of testing from the start of the project to the end. The development time reserved for the project is six months. Testing will start with agile testing after 3 months of development and continue with integration testing and acceptance testing after the development deadline.

Stages of testing will be agile testing, integration testing and acceptance testing and piloting. QA lead will participate from the start of the project to all scrum-based events in the project consist of planning, designing and daily and weekly meetings. Agile testing will start as soon as there something to test. A feature has been made playable or there are documents that need verifying. For making the communication as smooth as possible it is recommended that the QA lead will try to operate in the same team space with the development team as the deadline is one month away. The agile phase of testing consists mostly of experienced-based testing with support of test cases and static testing. All the completed features of the game are tested at least once during this phase. Integration testing will start right after the development deadline has been reached.

In the integration phase the entirety of the game is being tested before the acceptance testing. For the integration testing a new tester takes the lead instead of the QA lead. This is done to get a fresh look into the game. Integration testing takes usually to from two to four weeks.

After the game is tested it goes to release cycle of the slot machines. In the release packet there are multiple other features developed to the slot machine itself. In the acceptance testing the focus of testing is not in the game but rather the slot machine itself but many of the test cases are run on the game and thus making a little more regression to the game. Acceptance testing takes 2 weeks and after that a piloting period of two weeks begins.

The types of testing used for the project are mainly exploratory testing of the features of the game and test case-based testing. Usability testing, corner case-based testing and performance testing are also required during the testing.

The targets of testing of the game are:

- Basic game
- Doubling
- Free games
- Graphics
- Sounds
- Rules

- Winning combinations
- Win table
- Money purchases (physical and card)

Testing is performed on three different slot machine type. The targets of testing do not include working with other games and operating functionalities of the slot machine.

Reporting of testing during the agile phase is done in Confluence using test run links and progress of the testing by tables. Also, during scrum meetings with the team, the QA lead will state the progress of testing.

The testing has starting and ending criteria for every stage of testing. Agile testing can be started as a feature of the game has been developed to playable and a testing packet has been made. It is required to have a have a slot machine available of the same type used in development. Agile testing phase can be stopped when the final development sprint of the game has been finished.

Integration testing can be started if the game is feature complete, there are no open critical defects, there is a working automation tool and an official integration packet of the game has been made. In the integration testing it is required to have at least one of each type of slot machine (currently 3 different types required). Integration testing can end when a comprehensive test regression set has been run to the game and all the found defects have been addressed properly.

Acceptance testing starts at the beginning of every month. If the game has gone through integration testing before it, the game will be included to the acceptance testing packet. Testing can be started when the official release packets are done and there are enough slot machine types ready for testing. Acceptance testing can be ended when regression test runs are done, and all critical defects have been fixed. Piloting of the release packet will start after acceptance testing and end when the packet is officially released to all slot machines in Finland.

Controlling defects happens in Jira under the project of the game. All defects are reported to Jira but face to face reporting is also used especially in agile testing. Producer of the game is responsible for reacting to reported defects. The producer has authority to decide how to act to different defects, is the defect fixed or does not require fixing. Developer fix the defects and the testing team is responsible for verification of the defects.

6.3.2 Test results

As results for testing there are multiple ways of communicating different style of results. Coverages in different areas like coverage of mobile devices, coverage of features, coverages of tests run. There can be estimates of the quality of the tested software or quality estimates of the features of the software. In this results section we are focusing on found defects and the severity of the founds defects and what that could report of the quality of the slot game. Tests run are used as an indicator of coverage and size of the content of the game.

6.3.2.1 Defects found

There are total of 104 defects found during integration testing and agile testing phase. Most of the defects have no priority (59 out of 104) which mostly means a few things. Priority of the defect does not matter, the defect has been found during agile testing period and has been seen irrelevant to have priority at that time or tester and developer have simply forgot to include the priority to the defect. As to get information about the overall quality of the game the most important part about found defects are the number of found critical and high issues. When the software reaches point where everything is complete there should not be any critical issues. A few high issues are fine.

It can be seen from the figure 6.3 that there are critical issues found, seven to be exact. The number of critical defects found is quite high for simple slot game. A critical defect is a defect that blocks the release of the game. The game has chance of crashing during normal game play or has some problems concerning money transactions. These are the most commons cause for critical defects. What is little bit odd is that there are same number of high and medium issues reported as critical defects. Usually there should be more high defects than critical defects. This can be explained by the high number of defects without a priority. Defect without priority is usually priority from high to low, mostly being medium to low. The figure 6.3 tells that game has not entered testing in a totally complete state which is true. Due to schedule pressure of releasing the game it entered testing before it polished. The final development period happened a little bit one upon the other which is not optimal.

One interesting fact that tells about the quality of the software is how many integrations testing packets are needed to be done for integration testing after the development of the game is complete. In a perfect world only few testing packets are needed due to few defect fixes. But as it was noticed

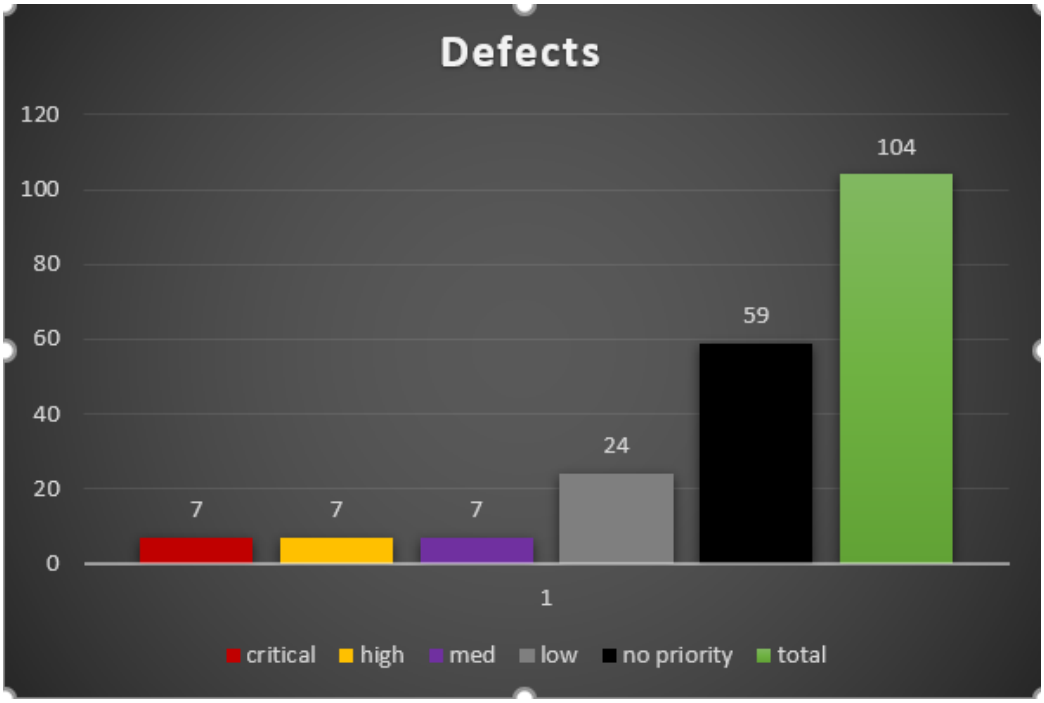

Figure 6.3: Found defects of a poker slot game

there where 104 defects found which resulted in creating of seven testing packets for integration testing.

6.3.2.2 Tests run

There were total number of 442 tests run to the game. This does not mean that there were the only test runs but just the ones that are documented in the test run itself. The number of tests run does not tell about the quality of the product but can tell about size of the content the game has. Whereas the number of the failed test cases of the run test cases tells about quality of the software.

From the figure 6.4 it can been seen that there are 24 fails out of 442 tested. This is quite a small number of failed tested as some test runs are done in agile phase of testing and more than one fails are duplicates from another test run due to having three different slot machines which each have their own test run. There are also few blocked or not relevant test cases. This can happen when chosen test does not actual test anything about the game or is blocked due to something not working at the time like card payment system or feature is incomplete. What is interesting is the relation between reported

Figure 6.4: Tests run to a poker slot game

defects and fails. There are 104 reported different defects and only 24 fails which some are duplicates. This can imply a few different things. Test runs are just a little part of testing and most of testing relies on totally different techniques of testing which is mostly true. Most of testing is experienced based testing and testing done around the test case but not the test case itself. It could also mean that the test runs are not the best for catching all the defects of the slot game. This is also somewhat true as the test cases are created to be generic and test basic operations of the game and slot machine.

6.4 Testing of a major slot game

The Second project we are looking into is a major slot game release. The game itself has multiple features including free games and a bonus game in a form a jackpot feature. As the game has a new jackpot feature it requires work not only from the development team of the game but also it requires work from multiple different background systems supporting the slot machine. Thus, the jackpot is such a large feature the game actual has two game development teams working at same time. The other develops the basic game and the free game feature and the other focuses only to the jackpot feature. The jackpot feature requires changes to the core systems of the

slot machine to operate properly as well as support of different background systems. The testing plan and strategy focuses on the game parts of the project including all the parts of the game but not the background system testing.

6.4.1 Test plan and strategy

Object of testing a is jackpot-based slot game created to slot machines owned and created by Veikkaus. It is slot game with free games and jackpot features as a bonus game. The estimate amount of work required total from testing for such a product is roughly seven working months. The seven working months includes all sectors of testing from the start of the project to the end. The development time reserved for the project is nine months. Testing will start with agile testing after 5 months of development and continue with integration testing and acceptance testing after the development deadline.

Stages of testing will be agile testing, integration testing and acceptance testing and piloting. QA lead will participate from the start of the project to all scrum-based events in the project consist of planning, designing and daily and weekly meetings. Agile testing will start as soon as there something to test. A feature has been made playable or there are documents that need verifying. For making the communication as smooth as possible it is recommended that the QA lead will try to operate in the same team space with the development team as the deadline is one month away. The agile phase of testing consists mostly of experienced-based testing with support of test cases and static testing. All the completed features of the game are tested at least once during this phase. Integration testing will start right after the development deadline has been reached.

In the integration phase the entirety of the game is being tested before the acceptance testing. For the integration testing a new tester takes the lead instead of the QA lead. This is done to get a fresh look into the game. Integration testing takes usually to from two to four weeks.

After the game is tested it goes to release cycle of the slot machines. In the release packet there are multiple other features developed to the slot machine itself. In the acceptance testing the focus of testing is not in the game but rather the slot machine itself but many of the test cases are run on the game and thus making a little more regression to the game. Acceptance testing takes 2 weeks and after that a piloting period of two weeks begins.

The types of testing used for the project are mainly exploratory testing of the features of the game and test case-based testing. Usability testing, corner case-based testing, anomaly testing and performance testing are also required during the testing.

The targets of testing of the game are:

- Basic game
- Doubling
- Free games
- Bonus game (Jackpot)
- Menu (Jackpot)
- Graphics
- Sounds
- Rules
- Winning combinations
- Win table
- Money purchases (physical and card)

Testing is performed on three different slot machine type. The targets of testing do not include working with other games. Some of operating functionalities of the slot machine are required to be tested due to changes made to the core systems of the slot machine software.

Reporting of testing during the agile phase is done in Confluence using test run links and progress of the testing by tables. Also, during scrum meetings with the team, the QA lead will state the progress of testing.

The testing has starting and ending criteria for every stage of testing. Agile testing can be started as a feature of the game has been developed to playable and a testing packet has been made. It is required to have a have a slot machine available of the same type used in development. Agile testing phase can be stopped when the final development sprint of the game has been finished.

Integration testing can be started if the game is feature complete, there are no open critical defects, there is a working automation tool and an official integration packet of the game has been made. In the integration testing it is required to have at least one of each type of slot machine (currently 3 different types required). Integration testing can end when a comprehensive test regression set has been run to the game and all the found defects have been addressed properly.

Acceptance testing starts at the beginning of every month. If the game has gone through integration testing before it, the game will be included to the acceptance testing packet. Testing can be started when the official release packets are done and there are enough slot machine types ready for testing. Acceptance testing can be ended when regression test runs are done, and all critical defects have been fixed. Piloting of the release packet will start after acceptance testing and end when the packet is officially released to all slot machines in Finland.

Controlling defects happens in Jira under the project of the game. All defects are reported to Jira but face to face reporting is also used especially in agile testing. Producer of the game is responsible for reacting to reported defects. The producer has authority to decide how to act to different defects, is the defect fixed or does not require fixing. Developer fix the defects and the testing team is responsible for verification of the defects.

6.4.2 Test results

In this results section we are focusing on found defects and the severity of the founds defects and what that could report of the quality of the slot game. Tests run are used as an indicator of coverage and size of the content of the game.

6.4.2.1 Defects found

For Major slot game project, it can be assumed that there are more found defects than in a minor slot game project. This applies basically to every software. As there are more features in the software the more complex it gets, and thus more malfunctions can be found between the features and in the features themselves as it is more likely the more there are features.

There are total of 184 defects found which is quite a large number for a slot game. Defects are found during agile phase of testing and in the integration testing. Again, like in the minor slot game, most of the defects are marked with no priority. This makes it harder to estimate the quality of the game. The reasons for having no priority are same as in minor game.

From the figure 6.5 the relation of critical, high and medium defects is rising. This is quite normal trend. The number of critical defects relate well the quality of the product. As there are 13 critical defects it can predicted that there will be more lower priority defects. The relation between all defects and critical defects feels high but is under 10 percent. As there are multiple reasons critical malfunctions, it is expected that the game has critical issues. This due to number of features and changes needed to core system as well

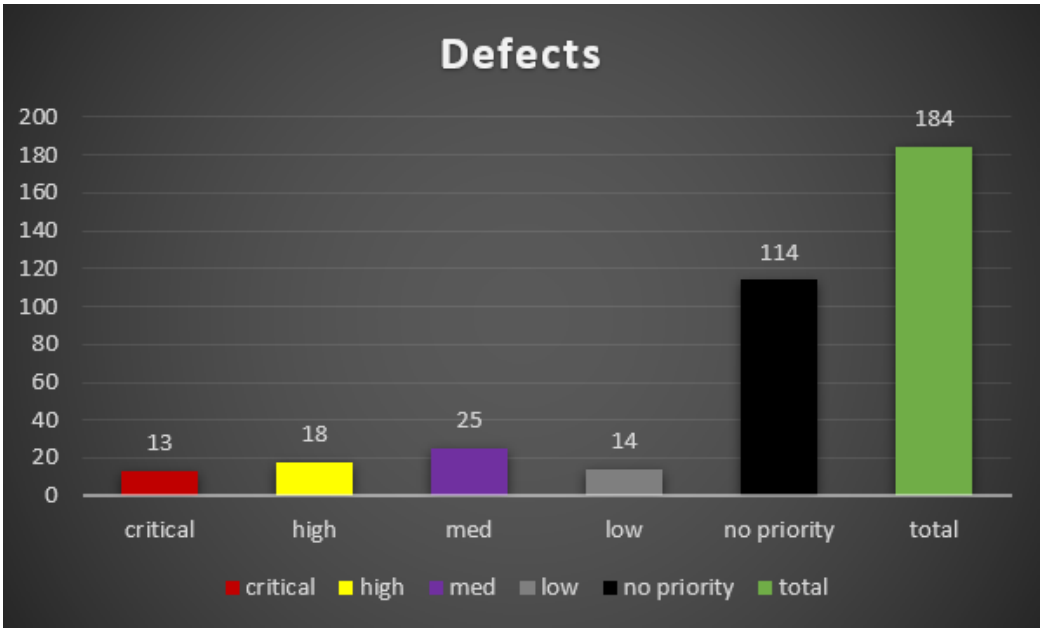

Figure 6.5: Defects of the major slot game

in background systems. The more things are modified, the probability of finding defects increase. The number low number of low priority issues can imply few things. First it possible that in a major slot game when something is quite right the impact can not easily be low. The another explanation is that most of the no priority defects can been seen as medium to low defects.

The number of testing packets made in the integrations phase of testing for the major slot game is 14. This is a large number of iterations made from a completed software. The number of packets needed can be explained by the number of defects founds, the poor overall quality of the product when entering to integrations testing and complexity of the product.

6.4.2.2 Tests run

There are total of 1235 tests run to the game. Total of tests run to the whole project including the core systems of slot machine and background systems is much higher. The number of tests run tell only about the size of content of the slot game. But the number of fails and retest tell more about the quality of the game.

For the quality purpose the most important factor of figure 6.6 is number of failed test cases. 149 fails out of 1235 can be seen as a lot. The explanation for having this amount of fails can be tried to be explained by multiple test

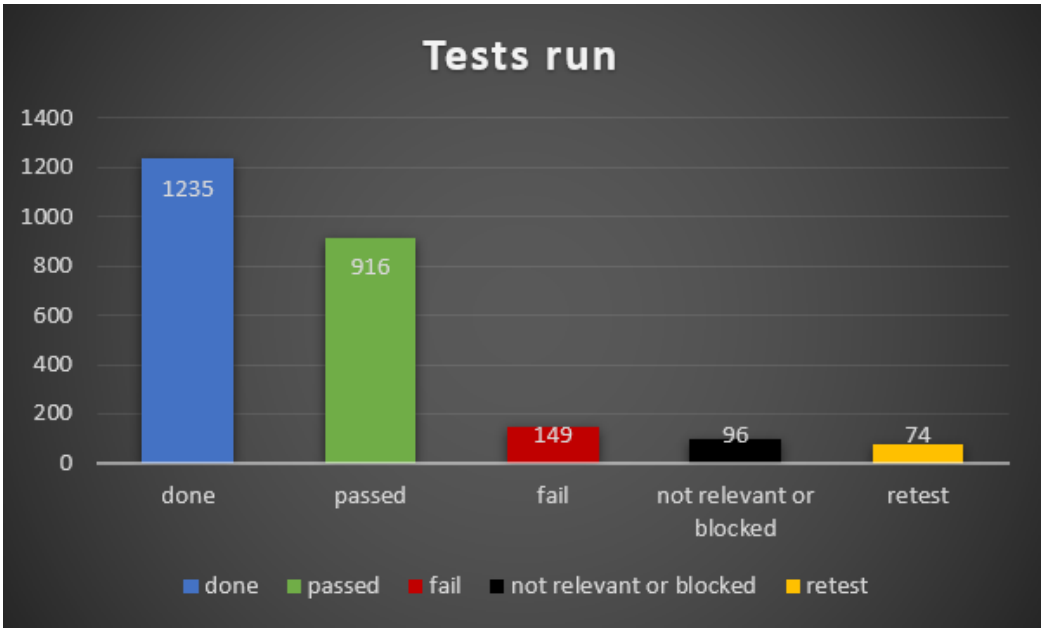

Figure 6.6: Tests run to the major slot game

runs in three different slot machine types. Fails are easily duplicated to each test run making the actual number of fails twice or even thrice lower. Still the number of fails tell that the quality of the game has not been perfect. Retest mean that the test is suggested to be run again in the next test run for different reasons. There are multiple tests put into a retest which means that the testing with test cases have been started so that not all features are fully operational and rerun of the test runs are made during testing.

The relation between fails and defects is 149 fails and 184 defects. As it has been explained that some of the fails are duplicates it can be noticed that many defects are found outside the test cases.

6.5 Conclusion of two different size of projects

The expectation for differences of the two similar projects of different size were that more testing time and resources is needed for the major slot game and that more defects are found from the major slot game due to complexity of the project.

As the expectations were quite vague and easily predicted, there were a lot requirements that made a large difference between the two slot game projects. The major slot game had needs that were uncovered through out

the development and testing:

- Need of more testing techniques
- Need of better architectural planning and documentation
- Need of understanding a big picture of the product including background systems and core systems of the slot machine
- Better tools for testing required
- Experience of the testers was more important than before
- The management of testing required much more than from a normal slot game

As requirements and different needs appear middle of the project, it takes time and resources to get the requirements. This usually effects the testing itself negatively. Even though there a lot of differences in the two projects the test process itself does not need to vary between a major and minor slot game project as the differences are more in the level of planning and strategy. Thus, for future projects the test plan and strategy must be more precise for major slot game projects as it is obvious that there are more requirements to be met in the major slot game project than the initial test plan and strategy suggests.

6.5.1 Comparing of test results

Comparing the result of two different size of slot games by found defects and tests runs makes the expectation simple. The major slot game has more defects and more tests run than the minor slot game. As for the relation between critical and total defects and relation between tests run and fails, it was expected that the relation could be the same or that the major game could have a worse relation due to complexity of the product.

6.5.1.1 Defects found

It can be seen from the start that expectation of major slot game having more found defects was correct. The major slot game had nearly twice the number of defects found which can be explained by the having more features, more development teams and requirements requiring changes to core systems and background systems of the slot machine.

The relation between the critical and total number of defects for both games is roughly the same, about 7 percent of the total issues were critical.

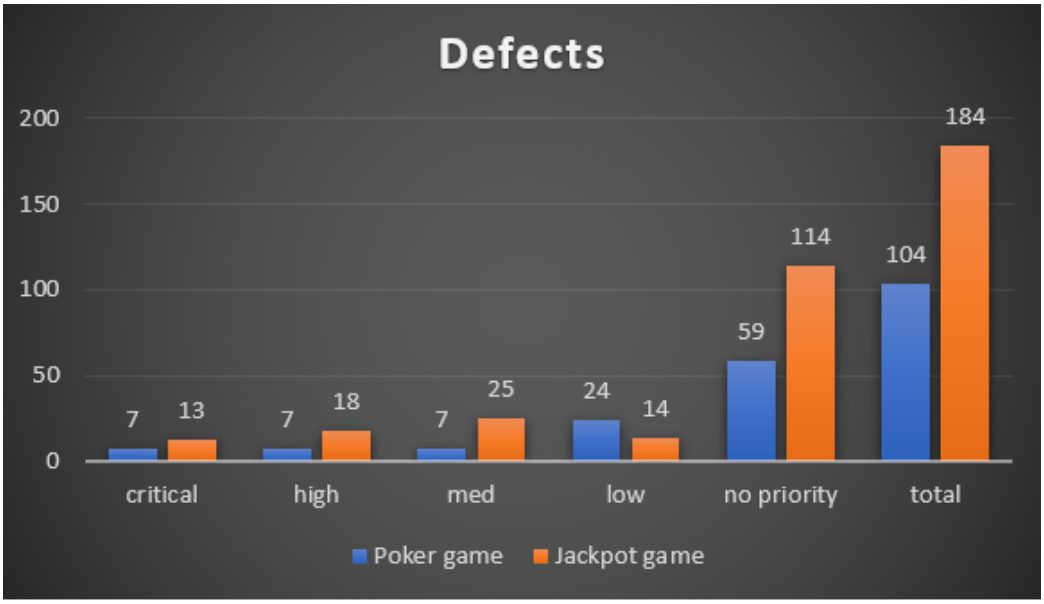

Figure 6.7: The defects of both projects in comparison

6.5.1.2 Tests run

The expectation for test run was that major game would have more tests run than the minor game. Thus, it is expected that there would be more fails are there are more different tests run. The major slot game has nearly thrice number of test cases run which is above the expectation. This can explained by creation of new test cases made for the new features of the major slot game and need of more regression tests during testing. As expected the major game having more tests run effects on having more fails.

From the figure it can be seen that the minor game did not need any retest as state for a test case. This can be explained by the number of iteration of packets required in the integrations testing for the major slot game. The major slot game needed 14 iteration in the integration testing as the minor slot game required only seven. The real difference between two projects and a failed expectation is in the relations of fails and done test cases. The major slot game has 12 percent of fails from the total of 1235 tests run whereas minor slot game has roughly 5 percent of fails of total number of 442 test cases. The difference can be explained from the structure of test cases. The cases are made generic and test basic functionalities of the game and slot machine. As the major slot game required changes to core and background system it can be expected that there are more fails in basic test cases.

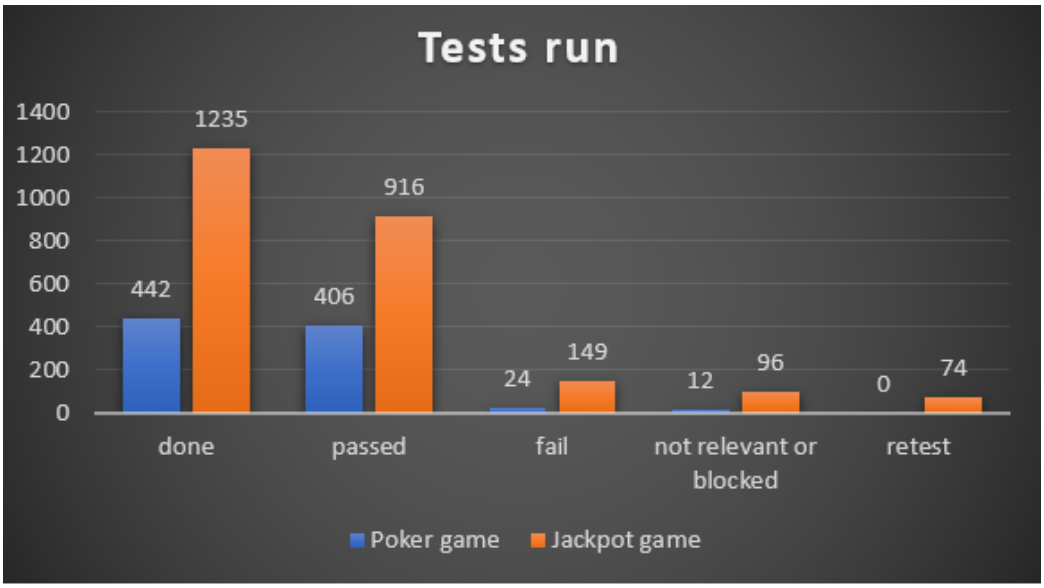

Figure 6.8: Tests run of both projects in comparison

Chapter 7

Conclusion

From chapters two to three the paper introduces different testing styles, levels and techniques for reader to understand what the different parts of testing and thus different part of functional and efficient testing process can be. Chapters four to five of the work tell about different basic testing processes and how the development process can have an influence how testing process is assembled together.

In conclusion of the theoretical part of the paper it can be noticed that there are multiple different testing processes ready to use which each needs to be properly modified for the testing process to be efficient for a random software project. Every software project has different needs and thus, it is important the testing process is always tailored to the software project in hand. A well designed and implemented testing process can increase the quality of the software a lot whereas poor testing process can make everything for the worse.

The case study is about two similar slot game projects which differ in the size of project as one is a minor slot game project and the other is a major one. Projects are compared by results of testing which include data from tests run and defects found. The idea behind the comparison of two projects is to find that is the current testing process flexible enough to use in both minor and major slot game project and what are the main differences between the two projects. In conclusion of the case study is it possible to use same testing process for two similar but different size of software projects but there must be adjustments in the test plan and strategy between the projects.

As for the conclusion of the paper there are four main conclusions:

1. Development process has a major impact in creating of a corresponding testing process

2. Two similar of different size of software projects might not need a separate testing process
    (a) Adjustments can be done in different level, for example in test strategy and planning
3. As many as there are software projects there are different possible testing processes
4. A great testing process itself is not a guarantee for a successful testing
    (a) Communication and experience of the tester is equally important

Bibliography

[1] Confluence. [https://www.atlassian.com/software/confluence/].

[2] Jira. [https://www.atlassian.com/software/jira].

[3] Robot framework. [https://robotframework.org/].

[4] Slack. [https://slack.com/].

[5] Testrail. [https://www.gurock.com/testrail].

[6] Iso/iec/ieee international standard - software and systems engineering – software testing –part 3: Test documentation. *ISO/IEC/IEEE 29119-3:2013(E)* (Sep. 2013), 1–138.

[7] Iso/iec/ieee international standard - software and systems engineering –software testing –part 1:concepts and definitions. *ISO/IEC/IEEE 29119-1:2013(E)* (Sep. 2013), 1–64.

[8] Iso/iec/ieee international standard - software and systems engineering – software testing –part 2:test processes. *ISO/IEC/IEEE 29119-2:2013(E)* (Sep. 2013), 1–68.

[9] Iso/iec/ieee international standard - software and systems engineering– software testing–part 4: Test techniques. *ISO/IEC/IEEE 29119-4:2015* (Dec 2015), 1–149.

[10] Black, R. *Pragmatic Software Testing.* Wiley Computer Publishing, 2007.

[11] Black, R. *Advanced Software Testing vol.2.* rockynook, 2014.

[12] Black, R. *Advanced Software Testing vol.1, 2nd edition.* rockynook, 2016.

[13] Black, R., and Mitchell, J. L. *Advanced Software Testing vol.3*. rockynook, 2015.

[14] Boehm, B. Barry boehm's equity keynote address, 2007.

[15] Cem Kaner, J. B., and Pettichord, B. *Lessons Learned in Software Testing*. Wiley Computer Publishing, 2002.

[16] Cem Kaner, J. F., and Nguyen, H. Q. *Testing Computer Software, second edition*. Wiley Computer Publishing, 1999.

[17] Cotroneo, D., Pietrantuono, R., and Russo, S. Relai testing: A technique to assess and improve software reliability. *IEEE Transactions on Software Engineering 42*, 5 (May 2016), 452–475.

[18] Garousi, V., Felderer, M., and Hacalo?lu, T. What we know about software test maturity and test process improvement. *IEEE Software 35*, 1 (January 2018), 84–92.

[19] Glass, R. L. A classification system for testing, part 2. *IEEE Software 26*, 1 (Jan 2009), 104–104.

[20] HOSSAIN, S. Challenges of software quality assurance and testing, 2018.

[21] International Software Testing Qualifications Board . Certified tester expert level syllabus improving the testing process (implementing improvement and change), 2011.

[22] International Software Testing Qualifications Board . Advanced level syllabus test analyst, 2012.

[23] International Software Testing Qualifications Board . Certified tester advanced level syllabus technical test analyst, 2012.

[24] International Software Testing Qualifications Board . Certified tester advanced level syllabus test manager, 2012.

[25] International Software Testing Qualifications Board . Certified tester foundation level syllabus, 2018.

[26] Jiang, Z. M., and Hassan, A. E. A survey on load testing of large-scale software systems. *IEEE Transactions on Software Engineering 41*, 11 (Nov 2015), 1091–1118.

[27] Juristo, N., Moreno, A. M., and Strigel, W. Guest editors' introduction: Software testing practices in industry. *IEEE Software 23*, 4 (July 2006), 19–21.

[28] Kasurinen, J. Elaborating software test processes and strategies. In *2010 Third International Conference on Software Testing, Verification and Validation* (April 2010), pp. 355–358.

[29] Kinell, B. Context driven test process improvement. EuroSTAR2018.

[30] Lamas, E., Dias, L. A. V., and Da Cunha, A. M. Applying testing to enhance software product quality. In *2013 10th International Conference on Information Technology: New Generations* (April 2013), pp. 349–356.

[31] Pradeep, S., and Sharma, Y. K. A pragmatic evaluation of stress and performance testing technologies for web based applications. In *2019 Amity International Conference on Artificial Intelligence (AICAI)* (Feb 2019), pp. 399–403.

[32] P.Shultz, C., and D.Bryant, R. *Game Testing all in one, second edition.* Mercury learning and information, 2012.

[33] Sneed, H. Value driven testing. In *2009 Testing: Academic and Industrial Conference - Practice and Research Techniques* (Sep. 2009), pp. 157–166.