Haaga-Helia
ammattikorkeakoulu Oy

Thesis

Test plan AX

- User Acceptance Testing for Company

Sanna Hyytiä

Thesis
Degree programme in IT
2018

Haaga-Helia ammattikorkeakoulu Oy

Summary

22.09.2018

Author(s)
Sanna Hyytiä

Degree programme
Degree programme in IT

Title	**Pages and Annexes**
Test plan for AX system	30 + 3

Title in Finnish
Testaussuunnitelma AX-järjestelmälle

The thesis is a functional work that consists of the user acceptance testing plan for an ERP system that is being developed for a retail company. The company will execute the user acceptance testing on the software based in the plan presented in the thesis. The test plan was part of a project executed in the author's workplace and was a part of the author's work assignments.

The functional part of the thesis was written in the start of the year 2018. The version of the test plan in the thesis is not the final one, since the project is not complete at the time when the thesis is finalized. The theory part to support the functional work was written during the summer of the same year.

The goal of the theory part is to give the reader an overview on software testing with the focus on user acceptance testing. The thesis also introduces the company's policy for project management and testing principles. The functional part of the thesis – the test plan – is based on the practices. The theories introduced will help the reader to better understand the test plan and the reasons why the plan is constructed the way it is as well as understanding the exclusions made in the plan. The theory provides also high-level information about software development.

Index terms
Testing, software development, ERP, user acceptance testing

Contents

1 Introduction

The thesis is a functional work where the output is the test plan for user acceptance testing of the new ERP (Enterprise Resource Planning) system (Microsoft Dynamics AX) for Company Oy.

The Company works in retail and has a national chain of convenience stores. There are ca. 560 Company stores all around Finland operating both on franchisee principle as well as some company-owned stores. The Company has a centralized warehouse from where goods to the stores are delivered from. The stores also place orders for other vendors, but also these orders go through the Company systems. The company provides also financial services to its franchisees.

The ERP project is very extensive, since it involves all the activities and departments in the firm. The Company has already used Microsoft Dynamics AX for some of its financial functions and as an item master. The on-going project will replace the current software used for logistics (Kilo). Some other business specific features will also be switched from old systems and new ones are developed from scratch. Along with the ERP project will also be executed a change for the item hierarchy used in the company. The project will have implications on other systems currently in use, e.g. the reporting system. There won't be a direct impact on the stores.

The project language is English, because the system provider is foreign. Therefore, all documentation in the project will be made in English, including the test plan. Hence the thesis is also written in English.

1.1 Scope

The theory part of the thesis discusses software testing in general and the planning of software testing with focus on user acceptance testing. The theory part is based on literature and research of software testing as well as interviews with IT management in the Company. The empiric part of the thesis consists of the latest version of the master test plan for the user acceptance testing of the new ERP-system. The test plan document will show the version history and changes made and annexes. The test plan is written according to the company's policies and guidelines for testing and project management which are also introduced.

The thesis won’t discuss the project and its tasks and outputs in any other aspect. No other documents that originate during the planning of the testing or during the actual testing will be discussed in the final plan. Also, the progress and results of the testing and the project are excluded from the thesis.

2 Software testing

2.1 What is software testing?

If a software is not working correctly, it can lead to numerous problems – loss of money and time and in worst cases injury or death. Testing offers a way to minimize those risks and to make sure that correct things are being developed and they are done in a right way. With the help of testing it's possible to check that the deliverable is what it was supposed to be (i.e. it works as defined) and to identify if and how the deliverable varies from what was planned. (Kasurinen 2013, 10; ISTQB syllabus 2018, 13.)

People can have a wrong idea about testing, thinking that it's just running test scripts and checking the results against the specifications. There are many other tasks in the testing process such as planning, analyzing, reporting progress and results to name a few. Testing can also be used to validate the quality of the product, to make sure that the system meets the user and stakeholder needs. So, testing is not totally based just on the user stories, code and specification documents. (ISTQB syllabus 2018, 13.)

Errors found during testing are deviations from specification documents. It's not possible to make logical testing without a specification, because without specification it's not possible to verify the results of the tests. Usually used specifications for testing are functional and technical specifications. With the help of testing it's possible to prove the presence of errors, not the absence of errors. This is true even in the simplest cases, even a simple case can have several variables. In practice it's possible to test just a fraction of all functions in software testing. This doesn't mean that it's useless to invest in testing, but rather that regardless of excellent test results there still might be issues with the functionality. (Haikala & Mikkonen 2011, 205 – 206.)

Mistakes occur because humans are fallible and especially when dealing with complex problems, code changes and integrations, people are more prone to make mistakes. These mistakes can lead to defects in the system and cause failure when the system is used. Failures can also be triggered by environmental conditions. In software development there are several stages and some or all of those can contain mistakes that lead to defects. It's not uncommon that mistakes are made already at the requirement or design level. The mistakes are more difficult and more expensive to fix if they're caused in the early stages of the development. (Graham, Van Veenendaal, Evans & Black 2008, 3-5.)

There are several terms that can be used when a software fails to what is expected of it, most generic ones are: bug, defect and error. But it's not uncommon to hear terms such as failure, fault or incident used in software testing. But whatever the term used, there is a general description what a bug is. Most of the errors can be determined using the specification documents; either the system doesn't do something that it should do according to the specification or it does something it shouldn't do or the software does something that's not mentioned in the specification or fails to do something that is not in the specification but should be. The fifth rule takes into consideration the user point of view; the software usability is poor, hard to use, slow etc. (Patton 2006, 13-15.)

2.2 Testing and software development

Testing is not an activity that is done by itself, it has its place in the software development life cycle. Whichever method of software development is chosen, it will mainly determine how the testing is organized and what testing methods and strategies will be used and have a significant impact on the testing. The chosen software development model is dependent on the project and its aims and goals. There are several methods and models ranging from light and fast to fully controlled and documented and everything in between. In each model the various phases of the software development are specified and organized accordingly. Testing should follow the guidelines set by the model; if the main goal is time to market, testing must also be quick and efficient to reach project goals. (Graham etc. 2008, 35-36.)

In the traditional waterfall method testing is the last phase to follow other project life cycles such as system requirements, definitions and programming. This model was one the earliest methods of software development where the process of the development is straightforward moving from task to task up to production and maintenance level. The waterfall has become outdated mainly because to fully collect and comprehend all the requirements and to make the definitions can be almost impossible in many projects. Testing is executed at the end of the project life cycle which means that bugs are found close to the release date. Another drawback is that in the waterfall model the testing is executed only in one phase. (Kasurinen 2013, 13; Graham etc. 2008 36.)

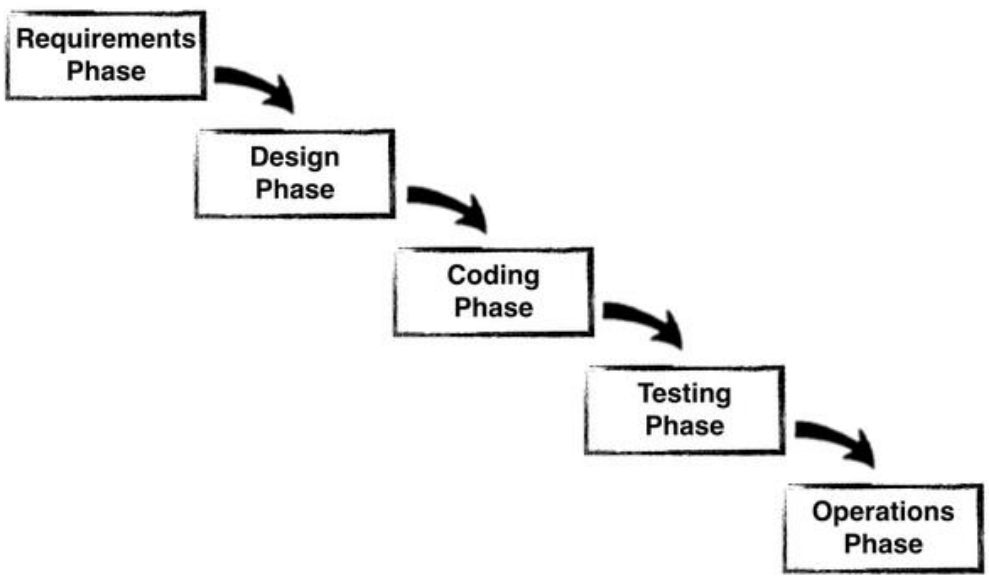


Image 1. Waterfall model
Source: https://medium.com/@davidjbland/the-state-of-waterfall-9e06a64665aa

The V-model was developed to tackle some of the issues in the waterfall method. It is similar to the waterfall method, but in the V-model testing is not considered as a separate phase. Instead all phases have their own distinct testing categories. This way it's possible to catch defects at an earlier stage than in the waterfall model. In the V-model there are three main phases for technical testing in the V-model: unit testing, integration testing and system testing. These phases are called test levels, because testing is made on a different level in each phase. In practice this means that programming is tested using unit testing, definitions with system testing and general system requirements with a separate acceptance testing. To be released, the system should pass all levels of testing. In addition to the technical testing, the V-model usually contains also acceptance testing where the system is validated against user needs, requirements and business processes. (Kasurinen 2013, 14, 51; Graham etc. 2008 36.)

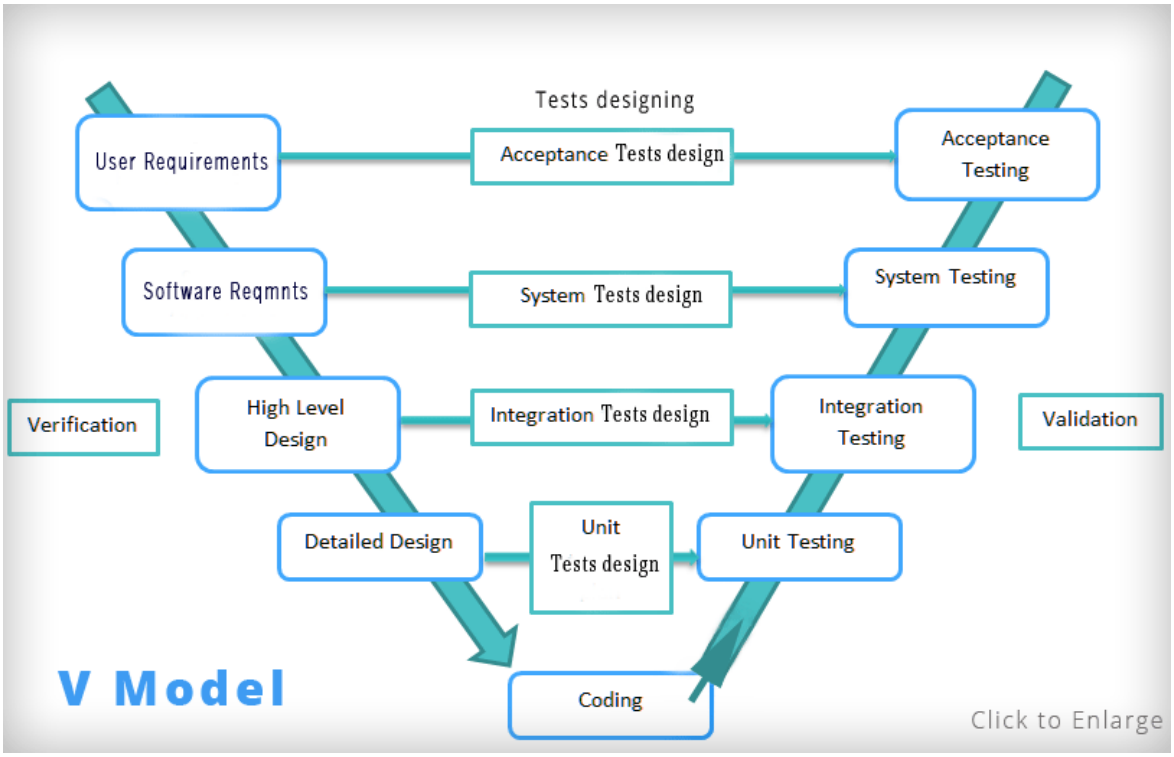


Image 2. V-model

Source: http://www.testnbug.com/2014/12/software-development-life-cycle-models/

But both methods (as well as most other system development methods) have the same problem with testing – the testing starts too late. When considering the system requirements or architecture, it would be beneficial to think how to verify these and test these before the whole package is complete. Also, usually if the project is behind on schedule (which most projects tend to be), testing is something that often suffers due to this when to either cut costs or to stay in schedule testing time is shortened. For testing point of view this is unfortunate, but looking at the financial point of view it might be even tragic. It's usually easier and more cost-efficient to fix issues in the first prototype or in the development phase than it is to find and fix issues in a deliverable already in use. (Kasurinen 2013, 14-17.)

These stiff noniterative methods can't also keep up with the increased competition where time to market is a vital competitive advantage - if not even the only way to keep the demanding twenty-first century customers happy. To meet with these growing demands, lightweight and rapid software development methods have been created alongside the old ones. These agile development methods emphasize iterative and incremental development where testing is also highlighted and in a more significant role than in the traditional methods. The agile methods are customer-oriented and aren't affected by change like the old methods – quite the opposite: changes are welcome and prepared for. Flexibility is important in these methods, but the core is in customer satisfaction and the customer is

key in executing a successful agile project. In agile methods there are no singular development process and many quick development methods can be labeled as agile. There are three common features in all agile methods: customer involvement, significant testing and short iterative development cycles. There are several different methodologies for agile development and each has its own unique characteristics; Extreme Programming (XP), Scrum, Essential Unified Process to name a few. (Myers, Badgett, Sandler 2012, 175 – 177.)

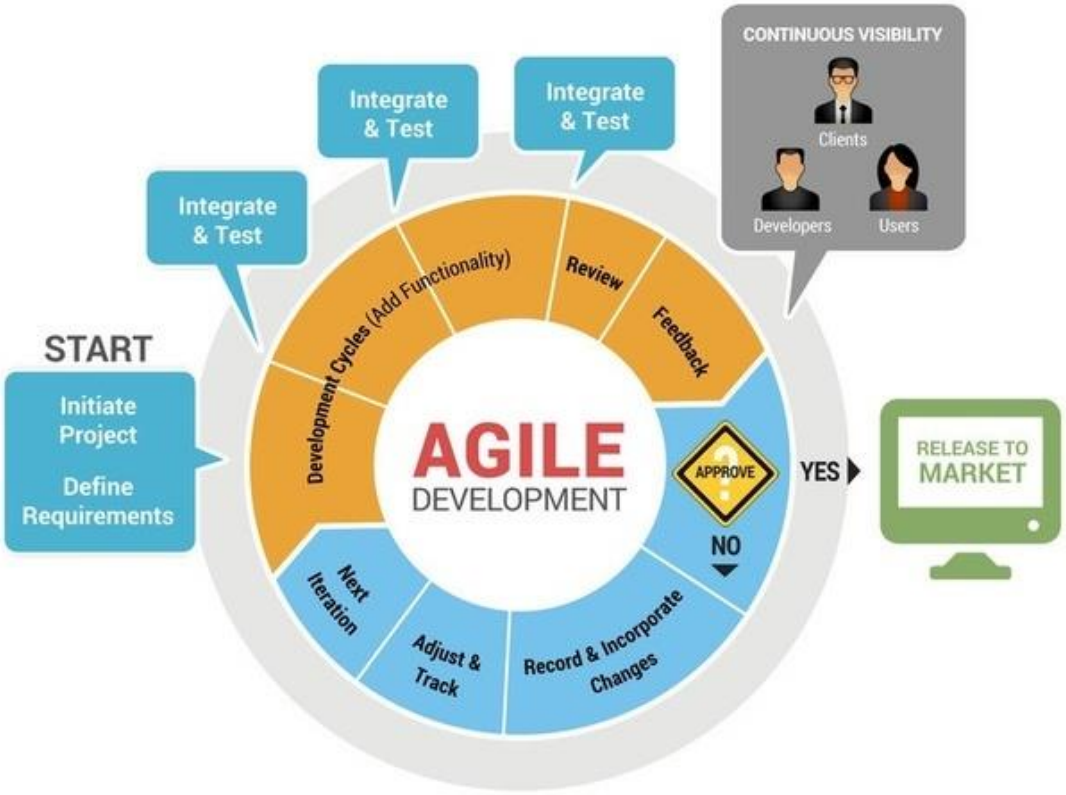


Image 3: Agile development
Source: https://www.otssolutions.com/blog/how-to-build-an-app-using-agile-development/

2.3 Software test levels

As mentioned in the previous chapter, there are different test levels in software testing and each has its own purpose and distinctive marks. Each test level is also executed at different stages in the software development cycle.

Unit testing:
Unit testing (also called component or module testing) is the most common form of testing and generally used in all software organizations. In unit testing one single module, component or function is checked immediately during execution, usually by the programmer or developer themselves. The purpose of unit testing is to ensure that the developed function works in principle at least. The problem with unit testing is that one single component by

itself can't usually do anything. Often the components have interactions with one another. To help with this it's common to build test components (mock objects) to simulate the traffic between different parts in the system. (Kasurinen 2013, 51-52.)

Integration testing:
Integration testing is performed after unit testing and involves several components and their interactions. In integration testing the different components of the system are set together to get the system working as a unit. The most important objective of integration testing is to make sure that the individual units are working when put together. The test cases are more extensive than in unit testing, but they don't yet cover the whole system. The test cases might consist of checking communication between modules or making sure that units using the same data base are working together correctly. The baseline for integration testing is that a new part is added to a working ensemble and making sure that it still works. (Kasurinen 2013, 54.)

System testing:
System testing covers the whole system in a controlled environment that corresponds as closely as possible with the production environment. It's the third phase in the traditional V-mode and often the final test made by the development team or an independent test team consisting specialist testers. The purpose of system testing is to find as many bugs as possible. System testing is executed after components have been unit tested and then built and tested as a working ensemble in integration testing. System testing should address both functional and non-functional aspects of the software. (Graham etc. 2008, 43; Kasurinen 2013,56-57.)

Acceptance testing:
After system testing the software is delivered to the customer for acceptance testing. With the help of acceptance testing is determined whether the software can be released or not. In acceptance testing is checked if there are any risks in releasing the software. Acceptance testing is usually carried out by the end users and testing is made in an environment simulating the real environment. Acceptance testing studies also the usability of the system. There can be identified two different test types in acceptance testing for business-supporting systems: user acceptance test has its focus mainly on the functionality and operational acceptance testing is used to validate that the system meets the requirements. (Graham etc. 2008, 44- 45.) User acceptance testing is addressed in more detail in chapter 3.

2.4 Test methods

In black-box testing the goal is to find out if the software is working according to its specifications. The method is called black-box, because the software is treated as box which contents aren't visible. The tester isn't aware of the code and the structure of the software, the software is given inputs and checked whether the results are as expected. (Myers etc. 2012, 8-9.)

The opposite of black-box testing is white-box testing. In this case the boxes contents are visible and the tester has access to the software code and can study it to help with the test effort. This approach might cause the tester to neglect the specifications. This testing strategy is also referred to as structural testing. (Myers etc. 2012, 10; Graham etc. 2008, 48.)

Functional testing is based on the software's functions, i.e. "what the system does". There can be two perspectives: requirements and business process. In requirement-based testing, the requirements specifications are used to create test cases and test approach. In business oriented approach the every-day business processes are used to test the system. Black-box testing is often regarded as functional testing where the tests are based on the functions introduced in the specification documents or use cases. But black-box testing includes also non-functional testing. Performance testing, stress testing, usability testing, reliability and load testing are considered non-functional testing. Overall non-functional testing is used to check how efficiently the system works. (Graham etc. 2008, 46-47.)

If changes are made to the system, those need to be tested too. There are two types of testing that is made regarding changes to the system: re-testing and regression testing. re-testing is made, if the test fails on the first run due to a defect in the system. The test case should be run precisely the same way as in the first time. The re-test tells us whether the issue has been fixed or not. But as the code has been changed, other issues might appear and to minimize the risks regression testing is needed. In regression testing test cases are run again even though they have passed in the previous software version. (Graham etc. 2008, 49.)

2.5 Planning the testing

2.5.1 Goals and strategy

Testing and its management should be considered as an entity which involves different tools, people, decisions and regulations. It's also important to understand that there's no such thing as perfect testing where all possible scenarios for all possible cases are tested. Hence testing must be controlled and it's vital to understand what exactly needs to be tested and make decisions on the testing. (Kasurinen 2013, 17-18.)

High-level expectations should be discussed when planning the testing. These are often dismissed, since there can be a common misconception that these are things that "everybody knows". First, it's important to make sure that it's clear to all what is tested – is it for example a new release or a maintenance fix? There should be a full understanding on the product for the whole team – testers and programmers likewise. In the planning should be recognized the goals and objectives in the project, both from the customer and stakeholders point of view. The goals should be clear and measurable - it must be possible to prove if they are met or not. For example, if the goal is to make the software fast, a benchmark should be set – how many transactions per second there should be possible to run. There are two ways to set goals for testing: first should be checked that the system is correctly built and fulfils the requirements set for the system. On the other hand, during the testing process can be checked that the correct product has been developed and no requirements are missing. There should also be a consensus on the quality and reliability goals for the product. (Graham etc. 2008, 21; Kasurinen 2013, 63; Patton 2006, 265-266.)

In the planning phase the scope, risks and objectives of testing should be identified: what potential risks are involved in the testing and what impact they might have. Also, the approach for testing must be decided. Here must be taken into consideration how the testing is executed, what methods are used, who is needed in the testing and the deliverables of the testing. The policies for testing should be agreed upon, or if the company has a set policy, this should be followed. The test strategy defines ow the tests will be executed and how the testers will use the software, for example will the testing be made using black-box testing or white box testing. Strategy is a key element in the success of testing and it can determine the success or failure of the tests. (Graham etc. 2008, 21; Patton 2006, 271-272.)

In the test planning should also be considered how to write the test cases and how they are used. There should also be discussed on how to track and record found issues (bugs). It's imperative that all issues can be tracked. It's also important to think about how errors

are reported and what documents and reports will be created during the testing process. (Patton 2006, 274; Kaner, Bach & Pettichord 2001, 233 – 235.)

2.5.2 Resources and schedule

The planning of the testing must also take into consideration the resources; this includes people, test environment, software, hardware, office space, outsourcing etc. – all that is used in testing. Everybody must know who are on the test team and how to contact them. Everybody should also have access to documents related to the project, so it must be made clear to all where these are stored. Additionally, it's vital that everybody knows and agrees on the words and terms used in the project. The resource requirements depend heavily on the project and it must be taken into consideration that it might be difficult or impossible to obtain resources late in the project. Tasks should be also assigned to testers on high-level so that each tester knows what is their responsibility. Testing tasks and responsibilities should be planned and scheduled so that they can be tracked throughout the testing. In the test schedule all beforementioned issues are put together in the overall project schedule. The schedule for the testing should take into consideration the amount of time the testing will require and in what parts of the development cycle testing will be done. (Graham etc. 2008, 21; Patton 2006, 266 – 272.)

2.6 Test plan

The test plan is a general project document that tells what is being tested, at what stage and what strategy is used. The test plan is based on the company's testing strategy and is usually aligned with company policies, but in some cases the plan can also deviate from the normal policy. The test plan is usually composed by the project manager or the test lead who is responsible for the testing. In some cases, the test plan can be just a collection of policies, but usually it's a cohesive document that states the main objectives for testing. (Kasurinen 2013, 116.)

According to the IEE Standard 829-1998 the purpose of the test plan is: "To prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task and the risk associated with the plan.". (Patton 2006, 264.)

The test plan is the project plan on how the testing is done, not a collection of test cases or procedures. Writing the test plan helps to understand the testing process. Using a template for the test plan helps to remember all important issues that must discussed in the planning, but there's no general rule what the test plan should look like or how it should be

written. There are some standards developed for testing, but the test plan and its contents and form depend on the company and the project. (Graham etc. 2008, 133; Kaner, Bach & Pettichord 2001, 233 – 235.)

The written test plan along with the planning process are tools for communication in the project team and other stakeholders. For testing to be successful it's vital for the testers to know how the testing is made and what techniques will be used. Test plan tells what tests will be performed and when, how they will be arranged and what results are to be expected. A preliminary plan is drafted in the definition phase and it will be supplemented later if need arises. In smaller projects usually one test plan is written, but in some cases, it's better to write multiple test plans for one project, for example integration and system testing are usually occurring at different phases and have different objectives. Because there might be an overlap in the testing and the plans, a master test plan that deals with common elements can help to reduce repetition in the documents. The plan also helps in change management when situations change, the plan is adapted accordingly. The test plan works as a baseline for measuring changes. By updating the plan as tests are run it's possible to keep up with the project needs. (Haikala & Mikkonen 2011, 216-217; Graham etc. 2008, 133 - 134; Kaner, Bach & Pettichord 2001, 233 – 235.)

There are many test plans templates and public standards available which can be modified to suit different projects. But using a standard can't be measured whether the plan is good or not – the plan is the document used to present the test process, but the written documents are not the only source of information on the testing process. Therefore, too much emphasis shouldn't be put into the appearance of the document or writing the document. This approach can give too much emphasis on the document, not the planning. It might lead to test leads just to copy an existing plan and modifying it just a little to suit their company and project. Instead of using a template, certain topics should be discussed and understood in the team. Patton suggests high-level topics that can be applied according to the project. The planning and discussions output will be a document (plan) of some sort, whichever format it is in is up to the team to decide on. (Patton 2006, 265; Kaner etc. 2001, 255.)

The project can also define the test plan and its form. The organization might have different types of strategies meant for different projects where testing is done differently. But in practice the test plan contains certain elements in all projects. For example, the ISO/IEC 29119 test standard plan contains the following basic issues:

- **Project description.** General information about the project, dates concerning testing and general information what is done in the project
- **Description of the test item.** Description what is supposed to be tested, and what are the main components and their relations. Description what the components do and what the system itself should do (or not do). Also, instructions how to find more information on the components.
- **Test scope.** Description of what parts of the system are tested and a list of the known issues of limitations and constraints.
- **Test strategy.** What strategy is used or define the used strategy for the project who, how, when and where the deliverable is to be tested. If the test strategy differs significantly from the company's standard strategy, the test plan should have a mention of it.
- **Schedule and work distribution.** Schedule for testing, containing all set deadlines, viewings and goals.
- **Risk assessment, action plan.** If the deliverable has some significant risks or there's a separate definition document, there should be a list how the requirements are verified or risks avoided.
- **Resources.** List of the test team, who is responsible for what and what skills and information the testers should have before they can start the testing (Kasurinen 2013, 117.)

The ISO/IEC 29119 standard is not the only way to approach and plan testing, it's only a recommendation. Smaller organizations can use simpler test plans that don't necessarily utilize the higher organization documents such as test strategy or policies. SPACE DIRT method is an example of a test plan that is developed to be used on project level. In this method the test plan is composed and defined according to the following features:

- **Scope** – Definition what components are tested and what are not tested
- **People** – What skills are needed for the testers, what responsibilities they have and what is the schedule
- **Approach** – What test methods are used in each stage of the testing
- **Criteria** – What are the criteria for staring, ending, suspension and resumption of testing
- **Environment** – Description of the test environment needed
- **Deliverables** – What are the deliverables of the test process
- **Incidentals** – What special features or anomalies are related to the testing and who is authorized to change the test plan
- **Risks** – The risks and how to prevent and prepare for them

- **Tasks** – Tasks belonging to the test process
  (Kasurinen 2013, 117-118.)

Often the test plan is just written just because it must be written and it's not really used nor read after creation. The focus on the planning should be in process, not just the writing of the document. This doesn't mean that the test plan document is unnecessary, but it's important that the plan itself is not the reason for planning. Kaner et co suggest that the test plan is "whatever ideas guide what you do" and how and if at all those ideas are documented is not the most important issue. Their opinion is also that document writing is not key to good testing – good testing can be executed without a written plan. The important thing is to make certain decisions for the testing – strategy, logistics and work products. (Patton 2006, 264; Kaner, Bach & Pettichord 2001, 233 – 235.)

3 User acceptance testing

3.1 What is user acceptance testing?

UAT means user acceptance testing and it refers to the end-user software testing made before a new system is taken into use. The key objective of UAT is to ensure the new system does what it should and meets the set requirements. ISTQB (The International Software Testing Qualifications Board) defines the UAT: “Formal testing with respect to user needs, requirements, and business processes, conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.” (Hambling & Van Goathem 2013, 15.)

Other testing takes place before UAT, but UAT is the first time the system is tested by users against their needs. Testing during development is based on ensuring that the system and components match the technical specification, but only user testing and experience can identify problems that might occur in business context. Even though the system has passed all previous tests, it can still fail the user acceptance tests. In UAT all business processes should be tested from start to finish. (Hambling & Van Goathem 2013, 16.)

It’s vital to know exactly what will happen when the first users log on to the system, and a well-planned and executed UAT is key to this. Even though UAT is important, it’s sometimes also considered as an expensive cost and waste of time and money. UAT should be independent of the development (IT), but work in collaboration with IT to achieve the best possible results. (Hambling & Van Goathem 2013, 20-21.)

3.2 The scope of UAT

The scope defines what will and won’t be tested during UAT. The scope should be defined in the test plan. The business processes aren’t tested before UAT and they must be the center of the UAT and tested from start to finish. The most critical features should be tested first and these should be tested within the process. Regression testing should also be made to ensure that the found issues are fixed and that the fix hasn’t caused any new problems. In UAT should not be tested things that already have been tested during unit, integration or system testing. This of course entails that there is proof of these tests having been implemented. Performance testing is usually also not a part of UA testing, except for some basic performance measures which occur during business process testing. (Hambling & Van Goathem 2013, 46-47.)

Business requirement are used to describe what the system should do. The more clearly the requirement specifications are written, more chance there is of getting a system that does what is needed. As well as defining for the developers what the system should do, the requirement documents will tell what to test in the UAT. (Hambling & Van Goathem 2013, 34.)

When developing a system, there may be many reasons why requirements do not exist or they don't match with was developed:

- It might be difficult for the end-users to figure out what the requirements are going to be. Ideas about the system change over time during development.
- There might be communication issues and requirements documents may not correlate with the intended.
- The developers might interpret the requirements incorrectly.
- Human error is always possible when the code is written.
  (Hambling & Van Goathem 2013, 18.)

3.3 Planning the UAT

At the time development is ready, there are certain deliverables: a set of business requirements, test results from the development team and possibly from independent testers and a system that should be complete and ready for UAT and a trained UAT team. These things should be ready and in such a state that it's possible to work with them and the project and the system should be ready for UAT. If the UAT is started with outstanding critical defects, the system code is changing all the time and every test must be repeated again and again. Not only is this expensive, but there's a risk that change control will become too complicated. (Hambling & Van Goathem 2013, 103 -104.)

Entry criteria is needed to avoid such issues. The criteria define what should be done before testing can commence and on what level the system should be. In an ideal world all previous testing is completed and there are no unresolved issues or defects, but in practice the criteria should allow for some defects (not critical ones) and open issues that have no effect on testing. As well as defining a goal for the development team, the entry criteria also give the UAT team some leverage in the project. It's too easy to pressure the UAT team to get the testing done and to deploy the system, if a project is behind schedule. But if UAT is done this way, there's not much use in doing it at all. When the entry criteria are set, it's possible to start planning the UAT. (Hambling & Van Goathem 2013, 104.)

The testers should work independently of the developers to ensure that the tests are designed solely from the user point of view. The problems with requirement specifications can occur, because there are different stakeholders that have different understandings on the system and business. Sponsors understand the business, but don't usually have a grasp on the technical issues. End users know what the system is supposed to do, but not how it will work and developers have the technical understanding but don't usually understand the business cases behind the requirements. These different views might cause misunderstandings in the documents and in the developed system. (Hambling & Van Goathem 2013, 43-44.)

As the system requirements may have changed along the way, a review to check what changes should be made to them is useful before starting the UAT. The original documents should be reviewed against the changes made and it's usually beneficial for the whole UAT team to take part in this. The review will give the team more knowledge on the requirements as well as the logic behind the system. It's important to prepare for the review meeting by checking the documents and noting down inaccuracies, ambiguities, omissions and questions as well as issues with clarity (grammar and text quality). This checklist can be used to check that the documents are clear and understandable – even if the reviewer doesn't know what the requirement ought to be. The aim is to identify deficiencies in the documents that may have caused errors in the development. It's also used to identify gaps that might cause that the implemented system won't meet the business requirements. (Hambling & Van Goathem 2013, 107 -108.)

When planning the UAT it's imperative to know also when UAT is finished. For this purpose, it's necessary to set acceptance criteria for the UAT. When setting the criteria, it should be taken into consideration what might happen if not all criteria is met – in other words, the criteria should not be too strict, for example – no defects, release is on time and everything works correctly 100 %. In most projects there's a high probability that not all the obvious criteria are met. That's' why the criteria set should be realistic in its expectations. This is done by examining the criteria and deciding how much it's possible to deviate from each demand. We must know what are the consequences if the system is released late or what might happen if the system is released with some defects and what level defects can be allowed and how many and what is the impact on these defects on the business. On the functionality side should be considered which functionalities are critical and which more of a cosmetic nature. There are critical functions without the system cannot perform what's needed, then there are functions that are important, but there might be a workaround to do these things. Then there are those functions that aren't essential,

but without them the system might be a bit more complex to use. (Hambling & Van Goathem 2013, 102.)

Acceptance criteria (exit criteria) is set criteria that the system must match to be accepted by the user or customer. The acceptance criteria tell that the system is ready to be released and the testing can be stopped. There are different criteria that can be applied, for example a certain percentage of the requirements must be tested, or it's possible to set quality criteria that there can be a certain amount of errors remaining at the time of release. It's important to define accurate and clear criteria to accomplish successful testing. The criteria can also be used to estimate the remaining amount of work and testing required to reach all criteria and so get a rough schedule for the delay. (Hambling & Van Goathem 2013, 39- 40.)

3.4 Testing approaches for UAT

Acceptance criteria sets the goal for the system release but they can't be used to decide how to reach that goal. For that a strategy is needed, that tells us how to manage the UAT and defines objectives how the desired system status can be reached. There are different reasons for doing UAT and these should be taken into consideration when the strategy is discussed, whether it's to reduce risk, gain confidence, assess the readiness of the system or to facilitate the transition for real-life use. The strategy should have objectives that help to approach the acceptance criteria so that all aspects of the UAT are taken into consideration and measured during testing. (Hambling & Van Goathem 2013, 105.)

Based on the acceptance criteria and UAT objectives can be defined a strategy to achieve the goals for the UAT. The strategy is the basis for the UAT test plan – what needs to be tested and to what level. The extent of the plan depends on the project – in a small project it can be efficient to use the acceptance criteria to define the tests. In the UAT strategy is defined what testing activities should be made to achieve the milestones based on UAT objectives and acceptance criteria. After a strategy has been decided on, a test plan for the UAT can be created. The plan will state what tests are required and how they should be organized so that they are most effective and useful. (Hambling & Van Goathem 2013, 105.)

In UAT the testers don't necessarily have any experience in testing nor in IT. Instead their close knowledge of the business processes gives them a perspective that can't be gotten elsewhere and their experience will add value to the testing and to the system. UAT is not based on testing outcomes against a specification, like other tests. Instead there are three

elements: business requirements, business processes and user expectations. Therefore, the approach for the testing must reflect these three elements and the test cases should be designed according to these. (Hambling & Van Goathem 2013, 70.)

Requirements-based test cases can be related to a specific requirement. Business process-based test cases help to make sure that the system will support the business processes. The test cases should be designed so that they follow the path of the business procedure. User interface-driven test cases are based on data entry, interactions and reporting. These can be included in the business process-based test cases. In user interface testing might be checked for example the required fields, data entries in correct format, confirmations, links etc. (Hambling & Van Goathem 2013, 71 -72.)

Because UAT is the last crunch before releasing the system to production, there is usually some pressure on getting the tests done on time. Therefore, the test cases should be prioritized so that if some tests aren't completed, these are of low priority and not critical to the system. This risk-based testing requires understanding of the processes and the system. In UAT should also be taken into consideration the fact that even though the system might pass all technical tests and do exactly what the requirements say, it might still fail the UAT due to poor usability or simply not fulfilling key business needs. UAT is made to ensure that the system is what is needed, not what was specified (71- 72.)

3.5 Test management controls

In the planning phase of the UAT should also be taken into consideration the management controls for the testing. The defects affect the decision for acceptance so an effective method for tracking defects is essential. Test logging is also needed so that it's easy to tell how much of the testing is completed and how much there is still to do. Mechanism for test logging should be set up before the testing starts. Change control procedures should be decided on so that each change is identified and followed up. Each fixed defect means a change in the system code and it's imperative to be able to keep track of the changes and versions in the system to determine what additional testing is required and to what measure regression testing is needed. (Hambling & Van Goathem 2013, 119.)

Test log lists the planned tests and captures finished tests and their outcome. With the help of the log, it's easy to generate statistical data to measure progress and predict the completion date. In the log should also be recorded changes and rescheduling made to the tests and the follow-up for failures – retesting and regression testing. If a test fails, an incident is recorded. At the time of recording the incident it's not always clear whether it's

a defect or not. The only conclusion at that point is that the result of the test doesn't match with the expected result. The cause might be faulty system code, but it can also be an issue in the test or the test environment. Therefore, the incident report written should be clear enough so that the developer can repeat the test, get the same result and find out what's causing it. (Hambling & Van Goathem 2013, 120.)

4 Project management and testing strategy in Company

The Company has a small IT department and it uses different software vendors to acquire systems and software. Company's IT has its own project model called "Ajatuksesta tuotantoon" – "From thought to production". The model describes how projects are planned and managed and it also contains templates as well as guidelines for testing. This model of project management can be used regardless of the software vendors system development method (waterfall, scrum etc.) No certain common standard for testing and test planning is used in the company, but there are certain similarities to the ISO/IEC standard as well as the SPACEDIRT method. (Talikka P. 14.5.2018.)

The usual tests run in the Company are acceptance tests for acquired systems and version releases before deploying the new system or changes to production. The software vendor will perform unit and system testing. Whether its new development or bug fixes, all are checked. Not only is the system tested against the specifications, but also usability is tested as well as tests required by the business (financial tests). All tests are planned before testing and documented. Depending on the system and the project, the testing is carried out by either the IT department or the end users in business and/or finance departments.

In the Company testing the test plan is usually written by the project manager who also usually creates the test cases. This practice may wary depending on the project and its scope. The tester(s) create the tickets as agreed in the test plan and creates a log as well as transcript (notes) on the testing. The project manager makes the conclusion of the tests based on these documents. The conclusion should contain information about open issues and their classification and the recommendation whether the system can be deployed to production or not, i.e. does the system meet the set acceptance criteria. In addition, it's good to note any new knowledge acquired during the testing.

The error classification used generally in the Company testing is:

- A = Critical, prevents the use of the system
- B = Prevents the use of a function
- C = Cosmetic fault
- D = Development feature

The usual acceptance criteria for systems are that there are no outstanding class A or B defects, but all acceptance criteria are set for each project and can vary.

In the annex 2 is the current template that is used for test plans in the Company. The user acceptance test plan for AX is constructed based on this template. The test plan answers the following questions:

- What is tested?
- How the tests are run (strategy)?
- When are the tests executed?
- Who is doing the testing?
- What dependencies are there?
- What is the test environment and its specifications?
- How is the testing process controlled?
- How is the error reporting and management handled?

5 Company AX UAT

5.1 Background

The Company is replacing the current ERP system Kilo and taking into use a new system - AX. The Kilo system is no longer viable and doesn't support all the current or future business features used in the Company. Some functionalities (e.g. Master Data Management) are already used in AX. In this project, the central warehouse (CWH) functions and functions related to that are moved to AX (orders, procurement, inventories etc.). There will also be some new features added to the MDM. The CWH will still use the Kilo system. In the project scope the old partner balancing system will also be replaced with a new system. A new feature for rebating campaigns will be implemented. The new system will allow further growth and provide additional features that will also support the Company franchisees better and lessen the manual work done in the Finance and Procurement departments. The new system will be more efficient and it will be beneficial to have all functions in the same system. Also, the old system is quite old and has some limitations to it and it isn't any more feasible to use in the current competitive situation in the retail market. (Keränen T. 4.5.2018.)

The project started in 2017 with selecting the vendor for the system and going through the system requirements and writing specification documents. The system was checked for features already matching with the Company requirements to avoid tailoring as much as possible. But since the Company business has many variables and has also some unique features to it, there were quite many specifications to be made. The chosen vendor was Sonata Europe Software Limited. A project team covering all aspects of the business was gathered. (Keränen T. 4.5.2018.) I myself joined the project only in the beginning of this year when I returned from study leave. Planning and managing of the user acceptance testing is my responsibility in the project.

5.2 Planning the testing for AX UAT

The goal of the AX user acceptance testing is to ensure that the system is developed according to specifications and make sure that the developed features meet the business needs of the Company and can be understood and used by the end users. This goal is met with careful planning of the testing - to plan and schedule the testing extensively so that different processes and their links and relations are taken into consideration and the testing runs smoothly. This process thinking is already considered in the planning of the

test cases. The target group of the test plan is the AX project team that consists of the stream (process) owners and the end user testers.

Understanding the different processes and their dependencies is vital for the success of the testing. The test plan will state the big picture of the project to the test team and create understanding to all users to help perceive their own role in the business and in relation to other processes. There will be other documents also for a more detailed testing schema and schedule, but these aren't included in the thesis. There will be references to these in the master test plan. There will also be respective test plans for certain elements of the system and processes, such as partner balancing, integration and rebate campaign.
The environment where the project has an effect is quite extensive as can be seen in the high-level picture (Image 4. Environment).

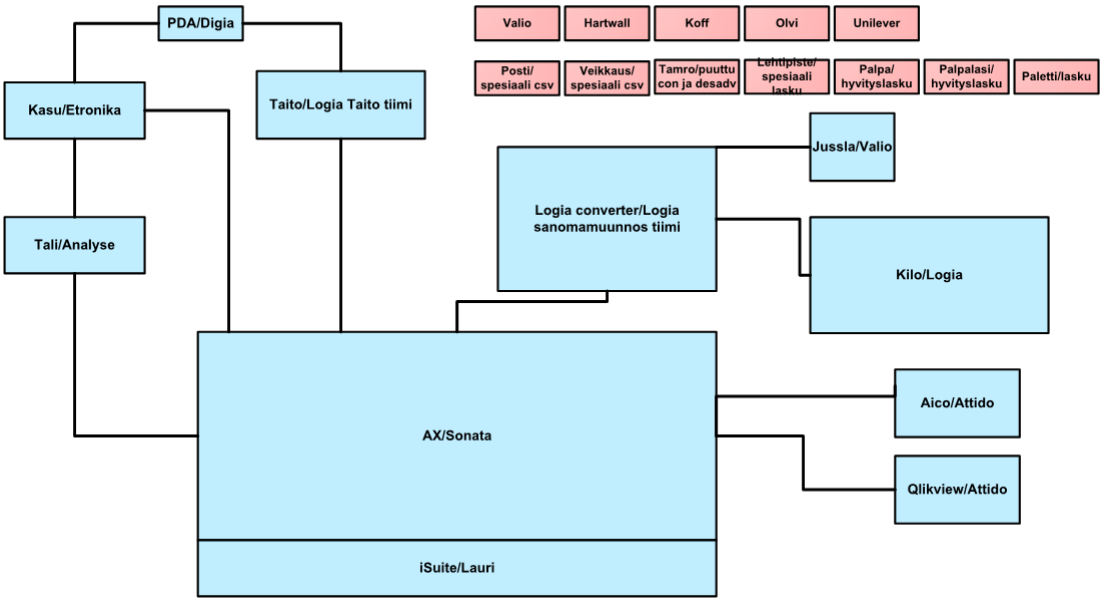


Image 4. Environment

Besides knowledge of the business processes and the environment, to write an effective test plan the author must have experience in testing, especially user acceptance testing, but also understanding of system testing and testing in general is required. To successfully coordinate the testing, it's essential to be able to comprehend the big picture, be ready to work with a variety of different personalities as well as have skills in problem solving.
The testing consists of checking and testing the processes related to sales to stores, creating and updating legal entities and item masters, sales at stores, rebate campaign system, partner balancing and invoicing and all the financial aspects of the afore-mentioned features.

A general master plan will be made and, also individual plans for each bigger segment (Kilo to AX, Partner balancing, Rebate campaign and Integration). The master test plan will handle the testing in a generic level and will also specify the testing strategy and resources, environment and risks of the testing as well as the schedule. Also, the links between each entity are described in the master test plan. The detailed, separate test plans of each entity will contain more detailed information about each stream and their testing as well as resources and tasks. These individual plans aren't discussed in the thesis.

6 Test plan for AX user acceptance testing

Here are described the contents of the test plan on a high-level. The plan in full can be found in annex 1. In annex 3 is the glossary for the terms used in the test plan.

6.1 Introduction and test organization

The first two paragraphs in the test plan are introductory paragraphs containing general information about the project and introducing the test organization. The introduction also tells the reader what issues are discussed in the plan and what aren't.

6.2 Scope

The paragraph explains the scope for the testing, both in-scope and out of-scope issues are discussed. The AX project scope is quite extensive and even though they have a relation to one another, they're tested independently. Therefore, the scope is divided into sub-paragraphs and each segment is explained with a short description of its content. The paragraphs are: Integration, Kilo to AX, Rebate campaign and Partner balancing. This division is the same as is used when additional test plans are created. In each paragraph is portrayed the main features and processes to be tested.

6.3 Test strategy

The test strategy paragraph tells how the testing is going to be made and what testing will be made on the Company side. The approach for the testing is explained here as well as what kind of testing will be made in order to reach the goals. In the AX UAT plan is included also the build acceptance testing and pre-testing. These are done to prevent problems in the UAT and to make sure that the definitions and business requirements are understood by the software provider and that there are no misunderstandings in the development.

The chapter also describes how the UAT is executed. The focus is on positive testing, testing the processes from start to finish to ensure each step is working correctly. There will be some negative testing also made, and of course regression testing is executed whenever needed.

6.4 Schedule

The schedule tells the overall schedule for the testing and contains a link to a more detailed schedule of tasks. The detailed schedule is not maintained here, because the

schedule changes during the project and updating here would be strenuous as well as difficult to follow for project members. In this chapter is also explained the overall process flow for the streams and what will be tested in what stage of the testing. There are also some explanations of the processes and what processes require the most effort. In the schedule is defined when the testing must be completed so it's possible to decide on the product deployment. The schedule doesn't address the go-live schedule and tasks related to that.

6.5 Tasks and Resources and responsibilities

Chapters 6 and 7 list the tasks and resources involved in the testing and some background on why some tasks are executed. The resources and their responsibilities for the testing are also presented here. The responsibility list is not detailed on person level in the master plan, the tasks and resources are discussed in more detail in the respective test plans made for each segment. Both chapters in the master test plan discuss the issues only on a high-level.

6.6 Test environment

There are certain requirements for the test environment which are listed here. There are software requirements as well as data requirements. This environment list doesn't contain the full picture depicted in the chapter 5 of the thesis, since these are again discussed in more detail in the respective test plans.

6.7 Entry, Suspension, Resumption and Acceptance criteria

In chapters 9, 10 and 11 are told the different criteria that define when the tests can start, be suspended and resumed if suspension occurs. The chapters also tell who will make the decisions on these matters. Chapter 12 lists the acceptance criteria on the testing. In this project are used the same criteria that are commonly used in the Company.

6.8 Dependencies & Requirements

There are dependencies that affect the testing and those are explained here, both personnel and software. These are discussed in chapter 13 on high-level and a more detailed discussion on these is in each segment's test plan. Chapter 14 tells where the requirement documents for the streams are saved and how they are updated if need arises.

6.9 Risks & Control procedures

Chapter 15 discusses what risks there are in the testing, what control measures are taken to control the risk and who is responsible for this. In this project the schedule presents the most viable and severe risk. The next chapter (16) explains how the progress of the testing is monitored and how the defect reporting is executed and how the defects are prioritized.

6.10 Tools & Documentation

The final two chapters in the plan tell what tools are used in the testing. These are usually (also in this case) common Office tools. The last chapter lists what documentation arise during the testing and as a result of the testing, such as meeting memos, defect reports and the final summary report.

7 Summary and deliberation

The goal was to make a test plan that could be used in the company's user acceptance testing and in that the I find that the result was successful. The plan was written according to the policies and templates used in the company and more information was gathered from various sources from the internet, books and training materials. As a result, the test plan template used in the company was revised and updated, which was already due since the last update was made several years ago. Even though there were no major changes made to the company's policies and strategies concerning software development and testing, I learned valuable information on testing and software development that can be used in future projects in the Company. Also, the knowledge about testing also grew in the end users and in the company while planning the tests, writing test cases and executing tests.

Since the scope was limited to the master test plan, the reader of the thesis will get just a whiff of the project scope. Also, the extent of the test planning doesn't show in the final work as much as I would've wanted it to show. There are the stream-specific test plans, schedules, test cases etc. that would've given more information about the project itself and of the variety of documentation that goes into the planning of testing and executing it.

Even though the scope of the thesis was just the master test plan, it would be nice to write here that the project went smoothly and testing is finished on schedule. But software projects have the tendency to get delayed due to various reasons, and this was also the case here. The project is not finished and the testing is still on-going as I'm writing this and the test plan is working as it should and still in use. We've had our first round of UAT and the test plans have lived accordingly and the master plan has been used successfully – even though it has gone through many changes as schedules as well as some requirements have changed. The process has taught how to best approach the test planning task and how to maintain documentation and to guide and help end user testers.

It was challenging to get involved in the project when it had already going-on and a lot was already completed. Although the business procedures in general were known to me, there were several issues that had to be learned and assimilated quite quickly to understand the big picture and figure out the best way to plans and execute the testing. But to participate in the project was beneficial to me and as mentioned before, the process has taught me a lot. My employer also got some benefit out of this arrangement, since they needed a motivated person to take care of the testing in the project.

As for the writing the thesis the most challenging part was to find material for the theory part – it seems that not much has happened in the testing scene in recent years and the sources were pretty much in agreement with one another. Also, material exclusively about user acceptance testing proved to be difficult to find. Deciding what to include in the theory part was quite a challenge due to the nature of the thesis where the most prominent aspect is the company's policies and template for the test plan. But I feel that in the end the theory part and the functional work are well balanced and support each other quite well and the finished work is well-rounded and justifiable.

Reference list

CTFL-Syllabus-2018-GA.pdf, luettu 3.7.2018

Glenford J. Myers, Badgett Tom, Sandler Corey. 2012. The art of software testing. 3rd edition. Word Association, Inc. Hoboken, New Jersey

Graham Dorothy, Van Veenendaal Erik, Evans Isabel, Black Rex. 2008. Foundations of Software Testing, ISTQB Certification. Revised edition. Cengage Learning EMEA.

Haikala Ilkka & Mikkonen Tommi. 2011. Ohjelmistotuotannon käytännöt. Alma Talent.

Hambling Brian & Van Goathem Pauline. 2013. User Acceptance Testing: A Step-By-Step Guide. BCS Learning & Development Limited

Kaner Cem, Bach James & Pettichord Bret. 2001. Lessons learned in software testing. 1st edition. John Wiley & Sons, Inc. Canada

Keränen Tapio. Company. ICT-arkkitehti. Interview 4.5.2018

Kasurinen Jussi Pekka 2013. Ohjelmistotestauksen käsikirja. 1. painos. Docendo Oy. Jyväskylä.

Patton Ron. 2006. Software Testing. 2nd edition. Sams Publishing

Talikka Pentti. Company. Tietohallintojohtaja. Interview 14.5.2018

Annexes

Annex 1. Test plan AX master

Annex 2. Company template

Annex 3. Glossary

Annex 1 Test plan AX

Sisällys

Version history:

Version no	Date	Author	Change
-	6.2.2018	Sanna Hyytiä	draft
0.1	8.2.2018	Sanna Hyytiä	edit
0.2	14.2.2018	Sanna Hyytiä	streams added to ch 2.2
0.3	16.3.2018	Sanna Hyytiä	XXXX
0.4	12.4.2018	Sanna Hyytiä	Valuecode added to Finance
0.5	23.4.2018	Sanna Hyytiä	Changes to schedule
0.6	11.5.2018	Sanna Hyytiä	XXXX
0.7	20.6.2018	Sanna Hyytiä	Changes to schedule and to strategy

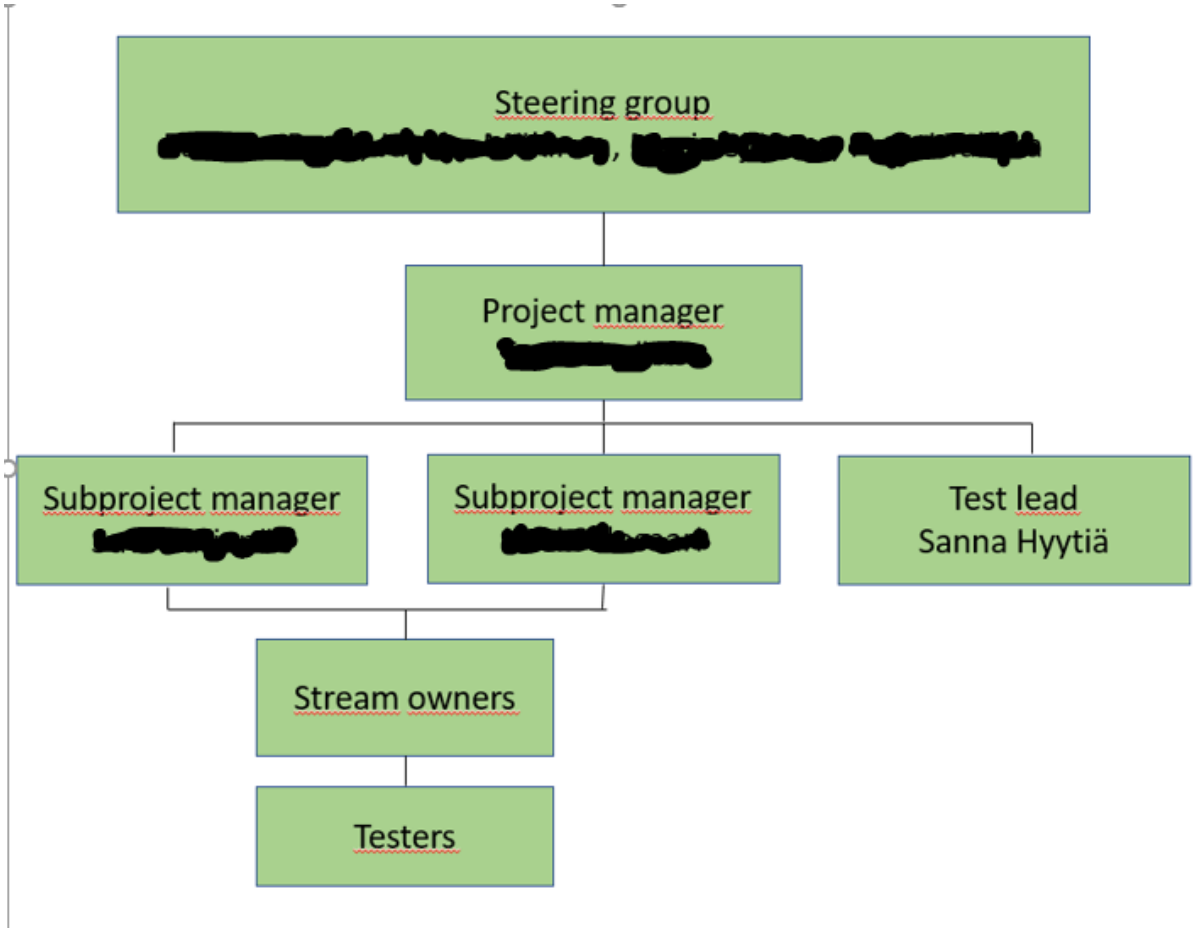
1 Introduction

Company is replacing the current system ERP system Kilo and taking into use a new system – Microsoft Dynamics AX. The current system is no longer viable and doesn't support all the current or future business features used in Company. Some functionalities (eg. Master Data Management) are already moved to AX. In this project also, the central warehouse (CWH) functions and functions related to that are moved to AX (orders, procurement, inventories etc.). There will also be some new features to MDM. The CWH will still use the Kilo system. In the project scope the old partner balancing system will also be re-

placed with a new system. A new feature for rebating campaigns will also be implemented.
The new system will allow further growth and provide additional features that will also support the Company franchisees better and lessen the manual work done in Finance and Procurement departments.

This document describes how the testing of the new system and its functions will be executed. The purpose of the plan is to give a general understanding on the scope of the testing and testing strategies. The plan also describes the resources, responsibilities and dependencies that the testing has. This document is the master test plan for the whole project and more specific plans will be made of each segment of the scope.

2 Test organization



3 Scope

The scope of the testing is divided into following segments:

- Integration
- Kilo to AX
- Rebate campaign
- Partner balancing

These segments contain several different functions and parameters which are defined in more detail in the respective paragraphs.

3.1 Integration

In this stage are tested the integrations from and to AX. The integrations that will be tested are described in full detail in the Excel-file "Integration Checklists- Phase B" which can be found in Y:\Yhteinen\AX\2.Määrittelyt\Liittymat\Excel for integrations. Visio charts for the integrations can be found in: Y:\Yhteinen\AX\2.Määrittelyt\Liittymat
The integrations affect nearly all the other scopes in the testing and these must be working and checked before other functions.

3.2 Kilo to AX

In this stage are tested the processes and functions that were previously performed in Kilo and will be performed in AX in the future. The processes contain several separate functions and business requirements from different departments. The list of business requirements and functions are listed in the Functional Requirement Document (FRD) saved in Y:\Yhteinen\AX\2.Määrittelyt\Määrittelyt\PROCESSES (COMPANY-AX2012-CU10-FRD-Process-V4.0.pdf). The functions and parameters that are not part of AX basic functions and need tailoring are recorded in the respective Functional Design Documents (FDD).
The table below lists the functions that belong to each process and the software and applications which are some way involved in the testing.

Process	**Functions**	**Software**	**Related stream**
Procurement	Purchases Receiving of goods Vendor MDM Vendor invoices	AX Kilo	Order processing CWH, Jussla & DD Purchase invoice process
Item management	Product hierarchy Product mapping	AX QlikView Aico Kasu Central Kasu POS	Product creation Product mapping Product setups CWH (bottles, cans)
Sales orders (to stores)	Regular order Express order Pre-order	Kasu Central PDA Taito	Sales process/wholesales - Sales regular, PRE,

	Distribution Own use items Proposal Refund Report on missing orders Store invoices	AX Kilo IW Aico	Proposal, Distribution order, Sales-own use orders Call center Sales order types/codes Delivery day price Assets (own use items) Reservations Sales invoice process
Inventory & Stock Events	Warehouses Stores Stock events	Kasu Central PDA Taito AX Kilo	Company Oy CWH/Jussla inventory Store warehouse inventory Transfer inventory/Legal entity
Returns	Callback (Stores to CWH) Request (Stores to CWH) CWH to vendor	AX Taito Kilo IW	Vendor return Sales orders
Legal entity	New legal entity (franchisee) New store for legal entity Legal entity changes	AX QlikView IW	Legal entity creation & setup Legal entity master& setup data, future AR setups for Franchise legal entity

Finance	Sales at stores Cash and Bank Ledger	AX QlikView IW Kasu POS Kasu Central Aico	Retail sales AR setups for Franchise legal entity Services (electrici- ty, security) Value Code

3.3 Rebate Campaign

Rebate campaign is a module in AX, which will be used to rebate campaign items to stores and to invoice suppliers. The owner of the process is the Purchase department. There are four different models to rebate campaign items which will all be tested:

- **Case 1** where the fee is calculated based Company purchase orders for direct delivery orders (Company -> Suppliers)
- **Case 2** where the fee is calculated based on the store sales data (Company -> Stores -> Suppliers
- **Case 3** where the fee is calculated based on store sales, linked with campaign or item (Company -> Suppliers)
- **Case 4** Fixed fees based on vendor agreements

Applications related to Rebate Campaign:

- AX
- Kasu Central
- Kasu POS
- IW

The applications mentioned above either contain information that must be checked during testing or require some actions to be performed in them during testing.

3.4 Partner Balancing

Partner balancing system is used to make sure there are no mismatches in store sales data and partner sales data and to invoice partners. The old partner balancing system is replaced with a module built in AX. All partner balancing and invoicing will be made via the new system in the future. The stream owner for partner balancing is Finance department. In partner balancing the following functions are tested:

- Partner balancing master data management
- Partner balancing reports

- Partner balancing invoicing
- Partner balancing transaction data
- Partner balancing corrections
- Partner balancing integration (Adaptor/Partner/Kasu)

4 Test strategy

4.5 Build Acceptance Testing

The system provider Sonata will perform system integration tests to ensure that the source code is working and there are no major open issues. These test results must be accepted by Company. Company will run their own build acceptance tests on the system to verify that it's testable. These tests are high level tests to make sure that key level features are working and no major bugs are present. The build acceptance testing is performed in Sonata premises.

4.6 Pre-test

To minimize failures in the build acceptance testing and in the UAT-phase, some preliminary tests will be performed by Company to the system. These tests will comprise of simple checks to the forms that are developed and deployed to the test environment. With these quick checks we can verify that the developer has understood our business needs and the development is made according to the FDDs. The checks will be made by the stream owners and no test cases will be written on these.

4.7 User Acceptance Testing

The UAT consists of a series of different tests executed in the new system and its integrations. The primary purpose of the tests is to confirm that the system is developed according to the specified user requirements and all modules are ready for operational use. We also need to be sure that the integration is working and all files are in the right format and data is moving flawlessly between systems. The UA-testing will be executed by Company end users in various departments.

The functions will be tested by creating test cases that describe the processes step by step where each desired and actualized result can be recorded and checked. Test cases are planned so that each department can use the same test case and test material and the process can be confirmed from start to finish.

This is due to uncertainties regarding some old Kilo functions where it hasn't been clear what ramifications some actions can cause to other systems. By testing the new system co-operatively and communicating openly across department boundaries, it's possible to avoid

these issues in the future. This way the testing process is also a tool for us to learn and clarify unclear issues.

The test cases will be created and performed by the end users with the help of the Test lead. Even though the system provider is English-speaking and the definition documents are written in English, the test cases are created in Finnish to lessen the work load on testers and to make it easier to create the test cases. The failure notices will be written in English as well as all other deliverables from the testing.

The test cases will be planned so that all steps and pre-requirements for the process are taken into consideration. Also, all the participants and applications involved should be considered when the test cases are planned. To be able to cover all aspects of the process, the test cases are published to the whole test team.

To make the testing process easier, we will use model cases from production where-ever it's possible. This helps in verifying the test results. The test cases are also planned so that the produced data can be used in as many processes as possible. This can be achieved by agreeing on the data specifications before testing.

Some parameters are covered in the test case for a process, but some parameter checks will have their own test cases. These are mainly tests involving master data management.

The focus is on positive testing, although negative testing will be executed at some level. Regressive testing will be made after bug fixes. Regressive testing is limited to cases where the fix might have an impact. So not every process and function will be tested from start to finish after bug fixes.

5 Schedule

The test cases must be written before the testing can start. To be able to allocate resources efficiently, the planning and writing of the test cases is started in March and the test cases should be ready and reviewed by 13.4. After this additions and adjustments will be made to the test cases.

Pre-testing will be performed in March and April following the deployments to the test environment. The build acceptance testing will take place 16.4.-20.4.

There are two weeks reserved for user training to help with the UAT-testing phase. Company will provide internal training for the testers with the help of Sonata team.

The detailed schedule for the testing and its tasks can be found here. All changes to schedule will be updated to this excel-file. There will be two stages for UAT-testing:

- 14.5. - 20.6.
- 1.8. - 31.8

All processes and functionalities will be tested within each stage. These will be scheduled according to their relations with each other. Some tasks can't begin until others have been finished or at least started. The tests will start by verifying each stream and checking that it's working correctly. After that the whole processes are checked from start to finish.

The processes are tested in a logical order to assimilate the real-life processes as much as we can. A high-level process goes: start with vendor and store MDM, stock to the warehouse, stores place their sales orders, stock is counted and financial checks made at the end of each process.

In stage one we test that the basic processes are working by using only a few items and vendors and ready templates from production. After we've established that the process is working correctly and the data is moving as it should, we can expand the tests to involve bigger volumes of data. This will happen in the last stage. In the last stage we will also focus on the variables in the processes as well as negative testing.

We expect to find the most severe errors by the end of the first stage, therefore leaving the final stage if the UAT to consist of final checks and finetuning.

All segments are tested within each testing stage (except integration, which should be tested within the first stage). Kilo to AX is the biggest and most time-consuming segment of the testing and it has elements such as item and vendor information which are critical to Rebate Campaign and Partner Balancing. Therefore, it requires more time in the schedule. This stage also involves the most people and we must make sure that their schedules are compatible.

The necessary fixes will be made within each testing stage. All in-scope tests must be completed by 31.8. to allow time to make last necessary fixes and to agree on unresolved medium or low-level defects. All known severe or high-level defects must be resolved by 16.9.2018.

The system is scheduled to be deployed to production on 1.10.2018.

6 Tasks

Build acceptance testing tasks:

- Create test cases
- Execute tests
- Write summary report

User Acceptance Testing tasks:

- Set-ups for items and accounts
- Create test cases
- Review test cases
- Execute tests
- Bug reporting
- Create ticket log
- Write summary report

In addition to the above-mentioned tasks, the following tasks will be completed during the testing process.

- Write and update the test plan
- Create templates for test cases and tickets
- Write instructions on testing
- System training by Sonata

The testers aren't experienced in testing, so short instructions and walkthrough of testing principles will be beneficial to them. Common templates for the test cases and tickets help make the testing flow easier and easier for everyone to follow the progress of the testing. The system is new to all testers, and therefore Sonata will provide training for end users so that they are able to complete the tests.

7 Resources and responsibilities

Each stream owner is responsible for the testing of their respective fields and dividing the work according to the know-how of each tester. The testers are also responsible for production issues at the time of testing. The resource allocation and testing schedule takes these responsibilities into consideration so that they won't hinder the progress of the testing.

Resource	**Responsibility**
Project manager: XXXXX XXXX	Project schedules Overall success of the project
Test lead Company: Sanna Hyytiä	Overall success of testing

	Coordinate test schedules Coordinate review meetings Communicate test status to project manager Communicate with Sonata Test lead Monitor the testing progress Assist in creating test cases Assist testers execute and document tests Create failure tickets to Sonata Create and update ticket log Write and update test plan Write and update status reports Write and update final summary report
Stream owners: Purchasing dpt. – XXXX XXXX, XXXX XXXX Finance dpt. – XXXX XXXX, XXXX XXXX Call Center – XXXX XXXX, XXXX XXXX Marketing dpt. – XXXX XXXX, XXXX XXXX Integration – XXXX XXXX, XXXX XXXX	Provide testers Assist in creating test cases Accept test results
Testers: Purchasing – XXXX XXXX, XXXX XXXX Finance– XXXX XXXX, XXXX XXXX Call Center – XXXX	Create test cases Perform actual testing Report about defects to the test lead Document the testing results

XXXX, XXXX XXXX Marketing– XXXX XXXX, XXXX XXXX Integration – XXXX XXXX, XXXX XXXX	

8 Test environment

To start and complete the tests, the test environment and its setups must be ready when the testing starts. In addition to the AX deployment on the server, there are several issues to be checked and prepared for the testing. A checklist for the requirements can be found here.

The testing will take place on the server 10.39.40.30.

The system and servers as well as the LAN environment need to be available during normal working hours. Any downtime will affect the test schedule.

The connections between the systems must be working during the tests.

8.8 Software requirements

In addition to AX, the following applications are involved in the testing and should be available during all testing cycles:

- Aico
- Kilo (CWH integration)
- QlikView
- Taito
- IW
- Kasu Central
- Kasu POS
- iSuite
- Management reporter

8.9 Data requirements

Test data - items, stores, warehouses, vendors and legal entities should be made available to the testers during the testing periods. Detailed data specifications are documented in the test cases. Most of the required data is set in the MDM and involves the parameters of items, legal entities, stores and vendors, but some data is created in POS, Taito and the PDA.

Data will be copied to the test environment from the production. The testing team will be informed of what date data is copied so that they can use this information while preparing test cases.

Whenever possible, we will use data from the production environment and copy it to the test environment to avoid creating large and strenuous amounts of data to the test. Data from the production environment will also be used for data volume testing.

9 Entry criteria

The tests can start when the following conditions are met:

- The build acceptance testing is completed and accepted
- Test environment is ready
- The test cases are ready and accepted

The Project manager and Test lead will determine when UAT tests will start and end.

10 Suspension criteria

If any defects are found which seriously impact the testing progress, the Project manager and Test lead may choose to suspend testing. Criteria that will justify test suspension are:

- Hardware/software is not available at the times indicated in the project schedule.
- Source code contains one or more critical defects, which seriously prevents or limits testing progress.
- Assigned test resources are not available when needed.

11 Resumption criteria

If testing is suspended, resumption will only occur when the problem(s) that caused the suspension has been resolved. When a critical defect is the cause of the suspension, the fix must be verified before testing is resumed.

12 Acceptance citeria

- All in-scope tests are completed
- No unresolved known severe or high-level defects
- The stream owners (with the help of the Project manager and the Test lead) have agreed that the unresolved medium or low-level defects are acceptable

13 Dependencies

13.10 Personnel Dependencies

The stream owners and testers in different departments must work together and be able to coordinate the test cases and tasks to be able to complete the test cycles in the allotted time.

The test team is dependent on the Test lead to provide them assistance in the testing and test cases throughout the testing and in coordinating the tests.

13.11 Software Dependencies

Integration is a key part in executing tests. Some tests can be executed only partly without integration or they can't be verified. Therefore, it's important to have the integration tested and working at the first stages of testing. There are dependencies in all segments: Kilo to AX, Partner balancing and Rebate campaign.

14 Requirements

The business requirements are listed in the Functional Requirement Document (COMPANY-AX2012-CU10-FRD-Process-V4.0.pdf) which is saved in Y:\Yhteinen\AX\2.Määrittelyt\Määrittelyt\PROCESSES. Functional Design Documents (FDD) are based on the pdf-document's requirements. The FDDs can be found in Y:\Yhteinen\AX\2.Määrittelyt\Määrittelyt\PROCESSES in their respective folders. The test cases will be created according to these requirements and the results of the tests will be accepted or refused based on these.

All functional definitions are completed before testing and the test cases are written according to them. Any changes to the requirements could affect the test schedule and will need to be approved by the Change Control Board (CCB). The test plan and test schedule are based on the current requirements and Functional Design Documents.

If the need to change definitions arises, they should be discussed with the CCB. The CCB consists of the Project manager and the stream owners for the respective departments. The CCB will then take the matter to be decided in the steering committee if necessary.

15 Risks

Risk:	**Impact:** 1 = Low 5 = High	**Control measures:**	**Responsibility:**
Project schedule:	5		Project manager

The schedule for each stage is very aggressive and could affect testing. A slip in the schedule in one of the other stages could result in a subsequent slip in the testing. Close project management is crucial to meeting the forecasted completion date.			
Testing sequences: Test cases are dependent on each other and some cases can't be started until others are completed or at least started.	4	Careful planning of test cases to understand the dependencies of different functions and departments.	Test lead
Resources: It's important to have the required personnel available for testing. The testers must be able to plan their schedules ahead and be able to plan and delegate their other duties.	5	Scheduling is a vital part in securing the personnel.	Stream owners
Skills: The testers have no previous experience	3	Provide necessary information at the beginning	Test lead

from testing.		of the testing. Provide support in creating test cases. Provide on-going support for testers during testing stages.	
Communication: The processes that are tested involve many people from different departments. If communication fails, there's a risk that some functions don't get tested with enough care.	2	Make sure that all parties are familiar with the processes and their dependencies. Careful schedule planning helps also here. Extra attention should be aimed at the start of the testing.	Test lead
Management: Management support is required. If the project falls behind, the test schedule should not get squeezed to make up for the delay.	1	Management can reduce the risk of delays by supporting the test team throughout the testing stage.	Project manager

16 Control procedures

The testing team will have regular daily review meetings where the process of the testing and ticket situation is reviewed. Additional

meetings will be arranged if necessary. The test lead will be available to assist the testers through-out all the testing cycles.

16.12 Defect reporting

Defects will be reported as soon as they're found out. The testers will relate any found issues to the test lead who will help to evaluate the issues and repeat the failure. The test lead will then write the failure notices and send them to system supplier Sonata. The test lead will be responsible for communicating with the system supplier about the defects and the fixes for them.
The tickets will be prioritized according to their severity and effect to system usability. The prioritization levels are trivial, low, medium, high, blocker.

17 Tools

Common Microsoft Office tools (Word, Excel, Outlook etc.) will be used during the testing. Sharepoint site "AX Testaus" will work as a platform for the testing. The site is used to store and share documents and memos used in the testing (test cases, ticket log etc.) and for communication. Other communication channels are e-mail and Skype for Business.
The tickets are recorded and shared with Sonata via Google sheets.

18 Documentation

The following documentation will be available at the end of the testing:

- Test Plan
- Test Cases
- Review meetings MOM
- Defect reports
- Ticket log
- Final summary report

TESTAUSSUUNNITELMA

Testattavan järjestelmän nimi

pp.kk.vvvv

SISÄLLYSLUETTELO:

Dokumentin muutoshistoria

Versio nro:	Tekijä:	Muutos:	Päivämäärä:
0,1	XXXX XXXX	Ensimmäinen versio	25.3.2004
0.5	XXXX XXXX	Toinen versio	3.5.2004
0.6	Sanna Hyytiä	3. versio	18.11.2013
0.7	Sanna Hyytiä		18.4.2018

8 TESTAUSSUUNNITELMA MALLI

Testaussuunnitelman tarkoituksena on kuvata mitä testataan ja miten testaus aiotaan suorittaa. Mallissa on *esimerkkejä*, joita voi käyttää oman testisuunnitelman tekemisessä. Mallia voi muokata projektin ja testauksen laajuuden mukaan ja tarvittaessa jättää pois väliotsikoita, joita ei tarvita.

Testaussuunnitelma tehdään kaikesta testauksesta. Testaussuunnitelman luettuaan testaaja ja projektin jäsenet ymmärtävät miten testaus suoritetaan ja missä aikataulussa.

Testaussuunnitelma vastaa kysymyksiin:

- Mitä testataan?
- Miten testataan?

- Milloin testataan?
- Kuka testaa?
- Mitkä ovat riippuvaisuudet?
- Millainen on testiympäristö?
- Muut vastuut testauksen aikana?
- Miten testausta kontrolloidaan?
- Miten virheraportointi tehdään?

1. Johdanto/Introduction

Kuvaa dokumentin tarkoitus ja lyhyt kuvaus projektista sekä järjestelmästä.

"Tämä dokumentti on testaussuunnitelma Yritys Oy:n projektissa testattavalle …… järjestelmälle.

Dokumentti kuvaa Yrityksen suorittamat testit. Testauksen suorittaa toteutuksesta vastaavat/erikseen testaukseen nimetyt henkilöt.

Järjestelmä on kehitetty XXX varten ja sitä käytetään…"

2. Testauksen kohde/Scope

Tähän kohtaan kuvataan testattava kohde ja mahdolliset rajaukset:

- mitä järjestelmää/järjestelmän osaa testataan
- testattavat ominaisuudet karkealla tasolla
- rajaukset, ominaisuudet joita ei testata (esim. joku järjestelmän osa tai toiminto, testataan vasta seuraavassa vaiheessa tai ei testata koska…)

Esimerkki, jos haluat käyttää taulukkoa:

Järjestelmä:	**Mitä testa-taan:**	**Milloin tes-tataan:**	**Laiteympäristö vaatimukset:**	**Ulkopuolisten resurssien tarve:**

3. Testausorganisaatio

Testausorganisaation kuva esittää organisaatiota testauksen aikana. Lisää tarvittaessa.

Esim:

- Ohjausryhmä (Ohry): hyväksyy testitulokset.
- Hankepäällikkö: Asiakkaan vastuuhenkilö
- Projektipäällikkö: Toimittajan vastuuhenkilö
- Aliprojektipäälliköt: Tekevät testaussuunnitelmat ja suunnittelevat alustavasti testitapahtumat
- Testausvastaava: Huolehtii testaajien saatavuudesta ja seuraa testauksen etenemistä
- Testaaja: Testaa jokaisen aliprojektin määrittelemät testattavat asiat ja dokumentoi ne

9

4. Testausstrategia/Test strategy

Miten järjestelmä testataan, mitä vaiheita on, mihin keskitytään, onko kyseessä hyväksymistestaus, validointi, tehdäänkö negatiivista testausta ym. Käytetäänkö jotain menetelmää (esim, Scrum, agile), testataanko järjestelmä osissa jne.

Toiminnallinen testaus

Testataan, että ohjelmasta löytyy kaikki toiminnot, jotka on erikseen määritelty

- myyntitoiminnot

- kampanjat

Positiivinen testaus

Testataan, että ohjelma toimii oikein käytettynä oikein. Esim:

käyttötapausten mukaan, raja-arvot, pienin arvo, suurin arvo:

- €-määrä
- KPL-määrä
- kentän pituus
- päiväys
- muuta…….

Negatiivinen testaus

Kirjataan ylös kaikki mahdolliset poikkeustapaukset testaus-suunnitelmaan

Esim:

- syötetään numerokenttään kirjaimia yms. (eli testataan, toimiiko ohjelma väärin käytettynä oikein)”

5. Aikataulu/Schedule

Kuvaa testauksen aikataulu ja päivitä tarvittaessa. Voit käyttää omaa, projektiin soveltuvaa mallia aikataululle tai:

Aikataulu:	Testattava asia:	Testauksessa mukana:

6. Tehtävät/Tasks

Mitä tehtäviä testaukseen sisältyy, esim. testitapausten suunnittelu, testien teko, koulutus, ohjeiden kirjoittaminen jne.

7. Resurssit ja vastuut/Resources and responsibilities

Kaikki resurssit, joita testauksessa joudutaan käyttämään, esim. talous, tarkastus, yms. Esim. taulukossa

Resource	Responsibility

Project manager: XXX XXX	 Project schedules Overall success of the project
Test lead: XXX XXX	 Overall success of testing Coordinate test schedules Monitor the testing progress Create and update ticket log Write and update test plan Write and update status reports Write and update final summary report
Testers: XXX XXX XXX XXX	 Create test cases Perform actual testing Report about defects to the test lead Document the testing results

8. Testausympäristö/Test environment

Testausympäristön ohjelmiston ja laitteiston kuvaus.

9. Kriteerit testauksen aloittamiselle/Entry criteria

Mitä pitää olla valmiina/tehtynä, että testaus voidaan aloittaa. Esim.

"Testaus voidaan aloittaa, kun

- *toimittajalta saatu ohjelma ja toimittajan itse tekemät testausdokumentit*
- *testaussuunnitelma on tehty*
- *testitapaukset ovat valmiita*
- *testiaineisto on luotu*
- *testausympäristö on käytössä"*

10. Kriteerit testauksen keskeyttämiselle/Suspension criteria

Listaa missä tilanteissa testaus voidaan/pitää keskeyttää.

"*Testaus voidaan keskeyttää ohjausryhmän päätöksellä, esim.*

- *Järjestelmä ei ole sovitusti käytössä*
- *Järjestelmässä on liikaa virheitä, jotka estävät testauksen suorittamisen*
- *Testausresurssit eivät ole käytettävissä"*

11. Kriteerit testauksen jatkamiselle/Resumption criteria

Millä kriteereillä testausta voidaan jatkaa, jos se on jouduttu keskeyttämään, esim.

"Testausta voidaan jatkaa, kun keskeyttämiseen johtaneet ongelmat on korjattu ja korjaus on verifioitu."

12. Hyväksymiskriteerit/Acceptance criteria

Kirjataan tähän kaikki testauksen hyväksymiskriteerit, esim. kuinka paljon saa jäädä avoimia virheitä ja minkä tason virheitä ja onko ne hyväksytetty. Kuinka kattavasti järjestelmä on testattu? Voidaanko joitain ominaisuuksia siirtää seuraavaan versioon, voidaanko tavoitteista tinkiä (laatu, kustannukset, aikataulu jne).

- *"havaitut virheet on korjattu ja hyväksytysti testattu*
- *yhtään virhettä ei ole auki*
- *viikkoon ei ole löytynyt yhtään virhettä*
- *kaikki testitapaukset on suoritettu hyväksytysti*
- *saadaan testattua prioriteetilla korkea olevat asiat*
- *laatu (paljonko virheitä sallitaan, jotka voidaan kiertää)*
- *kustannukset (testaamista voidaan lisätä maksimissaan nn htpv:tä, jos aikataulu ja tavoitteen saavuttaminen sitä vaatii)*
- *aikataulu"*

Virheluokittelu A, B, C, D:

- A = Pakollinen, järjestelmän käytön estävä
- B = Toiminnon käytön estävä
- C = Kauneusvirhe
- D = Kehityspiirre

13. Riippuvuudet/Dependencies

Listaa riippuvuudet testauksen onnistumisen kannalta, esim. järjestelmien välillä, henkilöstöön liittyvät jne.

14. Määrittelyt/Requirements

Mistä löytyvät sovelluksen määrittelydokkarit tai jos pieni projekti ja testaussuunnitelma on lyhyt, määrittelyt voi kirjata tähän.

15. Riskit/Risks 10

Listaa riskit testauksen onnistumisesta, arvioi vaikutus, miten voidaan välttää ja kuka vastaa riskistä. Esim.

Risk:	**Impact:** 1 = Low 5 = High	**Control measures:**	**Responsibility:**
Testing sequences: Test cases are dependent on each other and some cases can't be started until others are completed or at least started.	4	Careful planning of test cases to understand the dependencies of different functions and departments.	Test lead
Resources: It's important to have the required personnel available for testing. The testers must be able to plan their schedules ahead and be able to plan and delegate their other duties.	5	Scheduling is a vital part in securing the personnel.	Stream owner s

16. Testauksen seuranta/Control procedures

Kuvaa miten testien etenemistä seurataan, esim.

"*Testauksen eteneminen kirjataan testidokumentteihin - testitapausluettelo, virheraportti, testauspalaverin muistio, loppuraportti).*"

Muista tähän myös:

- Pidetäänkö statuspalavereja
- Muut seurannat ja tikettien raportointi ja priorisointi (alaotsikkona)

17. Käytettävät työkalut/Tools

- Mitä työkaluja testauksessa tullaan käyttämään - esim. Word, Excel, Power Point, Sähköposti, tiketöintijärjestelmä, Skype etc.

18. Dokumentointi/Documentation

Mitä dokumentteja testauksen tuloksena syntyy.

"Testitapaukset kirjataan testitapausluetteloon (Liite). Testauksen tulokset kirjataan testitapausluetteloon.
Testauksen lopuksi kirjoitetaan loppuraportti, joka sisältää yhteenvedon testauksen tuloksista sekä testauksen kulusta"

Muista myös virhetiketit, tikettilokit ja pöytäkirjat sekä käyttöohjeet.

19. Liitteet?

Mahdollisia liitteitä ovat. mm. testitapaukset, aikataulu, määrittelyt

Annex 3 Glossary

Adaptor	Module used for partner sales in the POS
Aico	Template model used to import data to Kilo/AX
CWH	Company's central warehouse
DD (direct delivery)	Vendors that deliver goods directly to stores
iSuite	Integration platform for interfaces
IW	Invoicing system used by Company franchisees
Jussla	Company warehouse for frozen goods
Kasu pos & Kasu central	Point of sales system and its background system
Kilo	Current software used for Company's logistics and finance, remains in use in Company CWH
Kilo to AX	Segment in the project that combines features that are moved from Kilo to AX
Legal entity	Either Company OY company or a franchisee operated company in the AX
Partner balancing	Tool for balancing partner sales and calculating fees
Partner sales	Sales made in Company stores via 3rd party applications
PDA	Handheld device used in stores to place orders and count inventory
Rebate campaign	Tool for calculating campaign fees
Taito	Software used in stores to place orders and count inventory
QlikView	Reporting system
Value code	B2B sales of value codes