UPTEC 11 006

Examensarbete 30 hp
Juni 2011

The test process and important testing techniques

Sebastian Mendez

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

The test process and important testing techniques

Sebastian Mendez

As software development gets more complex and high paced, the pressure on testing becomes greater to find business critical defects in the software as fast as possible. But testing that is only introduced in the last stages of development will most likely not give any valuable information. The fact is that testing needs to be an active part of development even at the early stages. Requirements need to be written with the input from the testers and testers need informed of what requirements matter, in order to start writing test cases early on. By making the process more effective testers have the chance to increase the quality of both the testing and software.

This thesis has been made with the help of the IT-department at ‘the bank’. Due to security issues I am not allowed to disclose the name of the bank. The goal of the thesis was to analyze the testing process from beginning to end present possible improvements. In order to make this evaluation a literary study was made in the subject.

An effective test process may vary a lot depending on how the organization is build. Factors like the test team’s size, resources and skills all weigh in heavily. Testing as also completely dependent on how development is managed. In order to improve the testing in the future the bank needs to start automating test on a regular basis. It will also be necessary for some structural changes to make the gap between analysts and testers smaller.

Handledare: Pär Johansson
Ämnesgranskare: Justin Pearson
Examinator: Anders Jansson
ISSN: 1401-5749, UPTEC IT 11 006
Tryckt av: ITC

Sammanfattning

I och med att mjukvaruutvecklingen sker i en allt snabbare takt och blir mer komplex finns det ett större behov av att testa mjukvaran. Testning är en relativt ny process som slog igenom på allvar för ca 15 år sen och som fortfarande är under stark utveckling. Idag finns det en skörd av metodologier och tekniker som kan använda för att genomföra testningen. Genom att effektivisera testprocessen kan man få en bättre kvalitet på tester, göra fler meningsfulla tester och spara tid.

Detta examensarbete skrevs i samarbete med en bank vars namn jag inte får uppge på grund av säkerhetsskäl. Målet med uppsatsen var att analysera deras testprocess från början till slut och ge bild av hur den ser ut idag hos samt hur den kan förbättras. För att ta reda på detta har en utförlig litteraturstudie gjorts som referenspunkt. Sedan har jag fått observera bankens test team under arbete samt intervjua dem.

Uppsatsen är uppbyggd i två huvuddelar där den första delen handlar om hur test processen är kopplat till utvecklingsprocessen, testprocessen i detalj, test tekniker samt kravanalys.

För att lyckas med mjukvaruutveckling krävs att man inkluderar testningen i ett tidigt skede. Testare behöver vara en del av krav-och analys processen för att kunna tillgodose den bästa nivån på sina tester. De behöver tid att skriva sina testfall samt att ge sin input angående oklara krav som kan försvåra testningen.

Just kravanalysen är en otroligt viktig fas som testning är helt beroende utav. Det mesta av testning som utförs idag är kravbaserat. Om inte kraven är bra och korrekt formulerade som kan man inte testa. För att enklast demonstrera detta har jag lagt upp ett antal punkter som visar vad som är viktigt med kraven från en testares perspektiv.

För att utföra själva testningen finns det ett antal olika typer av tekniker beroende på vad man ska testas. Eftersom testare idag är väldigt tidsbegränsade så har riskbaserad testning i princip blivit en standard för testare. Det innebär att man prioriterar krav som har högre potential att vara farliga ifall buggar skulle förekomma.

Utforskande testning är en annan viktig teknik inom testning. Det innebär att man börjar testa utan ett färdigt testfallsdokument. Poängen är att testarens kunskap om systemet och vart buggar är mer förekommande är mer effektivt än att bara följa anvisningar. Utforskande testning är betraktat som ett väldigt bra komplement till scriptad testning.

Automatisering kan förbättra eller försämra testningen markant beroende på hur den implementeras. Det finns många fallgropar, och det gäller att ha en färdig väl fungerande process innan automatisering utförs. För det första måsta man ha tester som passar just automatisering. Generellt kan man säga att dessa är tester som utförs många gånger med olika input. Automatisering kräver nästan konstant skötsel så det är nästan nödvändigt att någon jobbar heltid med detta.

Den andra delen handlar om banken och hur processen ser ut hos dem. Sedan görs en utvärdering baserat på vad som har skrivits i det första stycket.

Först görs en genomgång av vad hur testning går till vid banken. Vad som testas och vilken nivå. Eftersom utvecklingen är outsourcad till Indien så utförs den lägsta nivåerna av testning hos utvecklarna medan testteamet och klienterna sköter resten i Sverige.

Sedan görs en genomgång om hur testprocessen ser ut från början till slut. Hur ser testteamet ut och vilka roller har man. Det är viktigt att se att titta på hur detta är uppbyggt när man ska avgöra vilka typer av testning som är lämpligast.

Jag tar dessutom en närmare titt på det verktyget testarna använder sig mest utav. Ett test management program som kallas QC och gör en utförlig utvärdering om dess funktionalitet samt hur den används av banken. Här har jag bland annat kommit fram till att man inte tar vara på dess fulla potential, nämligen att inkludera kravskrivningen och använda sig av kravfunktionerna i verktyget. Detta skulle ge testare än mycket tydligare bild över vilka förändringar som har sket under projektets gång. Detta har visat sig ta upp väldigt mycket tid då testare måste söka igenom kravdokument i jakt på små förändringar.

För att knyta ihop säcken har jag föreslagit några idéer som jag tror skulle vara till stor nytta om man valde att göra några strukturella förändringar. Man skulle till exempel kunna ha någon som jobbar som testanalytiker. Denna person skulle jobba mycket med att göra kraven testbara och skulle fungera som en länk mellan testare och analytiker.

Table of content

Introduction

This thesis has been written by Sebastian Mendez as part of his education plan Information Technology Engineering Programme in Uppsala (2011). The work has been done in one of the big banks in Sweden where I have had the opportunity to follow their test team and study the subject of software testing.

'The bank'

Because of secrecy issues I am not allowed to disclose the name of the company. Instead I will refer to it simply as 'the bank'.

The testing at the bank is important part of development and maintenance, and is constantly being improved. The company uses a test management tool in order to organize software builds named Quality Center. The most important role of the test team is to perform integration and system integration tests on new builds. They also perform acceptance tests if they have enough knowledge about the system at hand. Otherwise they are supposed to supply the needed test support to the clients that will perform them.

Purpose

The reason for doing this thesis is that the bank is currently looking for new ways to improve their testing process. I have been brought in to evaluate the process and suggest improvements. This involved looking at how the process looks today, relevant testing techniques and testing connected to other processes in the development process. All of this also required me to evaluate the test tool as it is vital for the testing process.

The bank is currently in a nordic IT transition project. This means that they are trying to work in a more coherent and synchronized way with their branch partners in Scandinavia. But first there is a need to look at the structure of the process and evaluate them in order to determine what can be changed. Hopefully this thesis will give clear view on the overall status of the testing process.

Boundaries

I consider this thesis to be very broad in the way that it covers issues in the whole test process. It does not however deepen in specific test cases as this could be a potential security issue. It does not cover any economic issues that may exist or may arise due to suggested improvements of the process. I do however, not suggest improvements that require an unrealistic amount of new resources in terms of workforce or tools.

Even though the thesis comes with suggestions of improvements it does not discuss any implementation of them. Nor does it cover any specific future improvement plans in detail.

Issues

These issues where predetermined in the project plan:

How is the testing now and how can it be improved? How is the development process connected to automation of tests? How far has the bank progressed in regards to automation? When is it important to automate and when should you test manually? How are test environments and data used? Is the testing tool an obstacle? And finally, how is testing connected to development and which other processes are involved?

The test process

The test process connected to the development process

Before I describe the testing process, I have to mention the development process. This is because they are very dependent of each other and without some form collaboration between developers and testers the project will most likely fail (James Bach, 2002).

There are a lot of different development methodologies (Waterfall, V, Agile etc.) and each has their pros and cons. The process however always has the basic structure no matter what model an organization has chosen to use. Here are the basic development steps and the test activities during these steps. These steps are executed iteratively until the client (a stakeholder that has made the order) is satisfied with the software:

Initiation

Here an idea is stated by a client and the basic requirements are written. The client can often describe what he wants by describing different business scenarios. In this step, it is beneficial for development to be tightly integrated with the testing process. This is because we want to make sure that the idea created will later on be built in a testable way. If the idea has flaws at this stage of development you may encounter faults as late on as in the acceptance testing.

Requirements

In this phase of development, the requirements that explain how the system is supposed to run are defined and analyzed. This is done by specifying the "business process" and the functionality needed. The requirements are not usually written in testable way. They are often incomplete, unclear and ambiguous.

Specification and design

In this step the specifications for each module has to be written in a more detailed way than in the previous step to make sure that all the subsystems satisfy some form of requirement. The main difference between this phase and the earlier is that the design that is written here will make sure the scenarios and requirements are met by the suggested system.

Here, as in the first two steps the testers could be involved in order to ensure that the design is flawless in terms of testability. This means that all the business scenarios have been included in the requirements and captured by the new software. If they are not, it will be impossible to verify during system testing.

Build

Now developers can start to write code. In the meanwhile, the testers start to build their tests. Here is where all the work the testers have put in to planning, analysis and design comes to play. All the test cases, test data, test environments etc. are created. All this material will help the testers work fast and efficiently once the build has been delivered to them.

Test
In this stage the code has been written and is sent to the test team. The testers can start executing the tests that have been built for the software. The development team tests their software at a more low level of unit tests while it is the tester's job to check overall functionality and that the requirements are met. This is done by passing it through testing environments, doing system and acceptance testing, among other things. In these environments the results of the execution are recorded and checked for faults. Characteristics that are both functional and non-functional will be covered.

Evaluation
When the testers have tested the system for it to be robust enough to deliver what the requirements specified. Then the process is finished. By then there must be assurance from the testers that the defects have been identified and corrected. Time constraints usually make it impossible to test the entire system so instead the goal is to reduce business risk to an acceptable level.

Implementation
The software has been approved and corrected and is now ready for production.

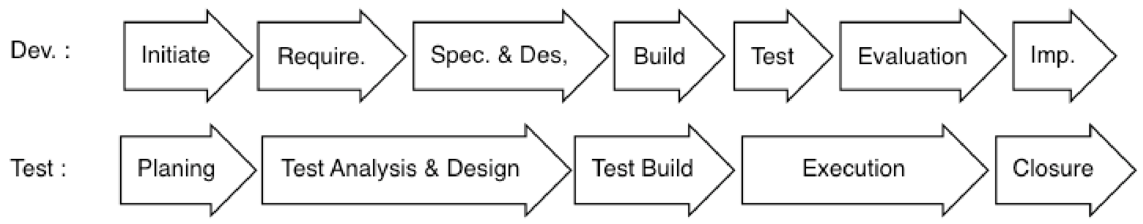


Fig. 1: Development and testing timeline

This is the basic process cycle for software testing. Methodologies may differ in steps but they all have this basic structure.

The test process
During the development process, the testers do not just wait until the build is ready to start working. A lot of time is spent on preparing and presenting the tests. The process of going from requirements to execution always looks the same and only varies in how much time is given in for the cycle intervals. Here are the workflow steps:

Planning
The test manager initiates the cycle by creating a test plan. The test plan defines what will be tested and why it is tested. During the whole testing cycle it is the duty of the test manager to document and communicate the progress that has been made.

Test analysis and design
A test analyst defines how to test the system by writing a specification. The test specification will contain the work made during the analysis such as meeting notes and documents. The test design will contain multiple test cases needed to validate and verify the software. We also specify scripts and other information needed to perform the tests

such as test data and test environments. As the documentation becomes more specific, the testing can become more complex and reach lower levels.

Build tests

Now the test engineers have all the information they need to start to build the scripts. The analyst supplies them with the needed test data and test environments. At the same time the test manager has created a test schedule where it is specified when the scripts will be used.

Execution

The testers are the ones executing scripts. They will document faults and when they are finished, they produce an execution record.

Closure

Here the testing cycle is rounded up with a final report and some form of lessons learned session that the whole team can use to discuss the issues of this particular cycle and work on improving their testing strategies.

It is important to also have a similar session sometime after the releases when some undetected defects may have been found. All the tests that have been performed should be stored for future regression testing.

Roles and responsibilities

Generally there is always a test manager or test leader organizing the testing and a couple of testers building and executing tests. The amount of testers depends on the organization. The roles and responsibilities vary between different organizations but they are usually defined like this:

Test manager (team leader) - The test manager produces a test plan. He organizes the test team and delegates work load. He is in charge of test preparation, creating the final reports and following the progress of the test team throughout the cycle.

The test manager will also update and modify the schedule depending on how the testing has resulted. If the build is not acceptable, it has to be sent back to be fixed and this can delay testing and retesting.

Test analyst (test designer/analyst) - The test analyst specifies how the test will be conducted and writes a test specification and a test design (more on documentation later on). He also is in charge of providing test environments and test data.

Test executors (testers/test engineers) - Build and execute test cases based on the documentation handed down from the manager and analyst. Record the results and write test reports.

(There can be more or fewer roles in the test team depending on how big the project and workforce is. These are however the most common ones.)

Documentation
Here are some descriptions of how the different documents usually are written:

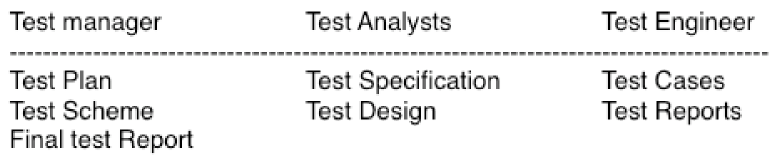
Test plan – Describes what will be tested, why and under what time frame.

Test specification and test design - Describes how it will be tested. The specification contains detailed analysis including meetings, documents and workshops. The design includes test cases needed. It also specifies but does not build test environment, test data and test scripts.

Test case – This is the outcome of all the requirements and planning. Includes very detailed testing instructions and the expected results.

Test scheme – Describes when to perform the testing.

Test report – Describes what has been discovered.



Test manager	Test Analysts	Test Engineer
Test Plan	Test Specification	Test Cases
Test Scheme	Test Design	Test Reports
Final test Report		

Fig. 2 Roles and Documentation

Requirements
During the requirement analysis there is supposed to exist some form of documentation. And this documentation is supposed to list the quality attributes for the requirements. The IEEE has a standard guideline for how these documents should be written. This specification is called the 'The practice for software requirements specification'. This standard along with others can be found in IEEE website http://ieeexplore.ieee.org/.

The IEEE specification is rarely used in practice. Organizations develop their own documentation based on their needs. Generally the requirement specifications are documented to vaguely or to precise. In either case, any type of analysis will require a lot of background knowledge in order to understand what is needed. If testers and developers have a hard time understanding the documents they probably will not maintain or use them as the process continues.

In order to write good requirement documentation, they have to be scaled down and adapted after the needs of the organization. The most important is that they explain a set of ideas that the developers understand and that is practical to implement.

The requirements are mainly written for the developers and the needs of the tester are not always taken to account. It is the tester job to look at the requirements and find the risk of the software.

Requirement principles

If testing is done based on requirements (which is not always the case), there are some principles that are needed so that the tester can do his job correctly (Bach, James; 1999):

1. Without stated requirements, no testing is possible.
2. A software product must satisfy its requirements.
3. All the test that have been performed must satisfy some stated requirement.
4. Requirements must be stated in testable terms

These principles are very simplified and each one will now be studied in detail:

1. Without stated requirements, no testing is possible.

Even though this is very true, it does not mean that testers should just sit and check of requirements. This can lead to a very robotic work routine. In fact, the tester could have a more exploratory role than just testing stated requirements. Instead of just reporting missing requirements a tester can fill up the gaps and complete the requirement with some explicit information.

2. A software product must satisfy its requirements.

But the amount of verified requirement does not measure the quality of the software, because not all requirements are equally important. Time constraints are often an issue. Sometimes there just is not enough time to validate each requirement. That is why it is important for the tester to consider the risk in each requirement, and make sure to find the biggest faults.

3. All the test that have been performed must satisfy some stated requirement.

This is also a very simplified statement. Just by checking of requirements can lead to an unnecessary low understanding of the product. A tester should be able to explain the relationship between the tests and the requirements.

4. Requirements must be stated in testable terms.

Even though this is very important there may exist a risk in doing this to. If you scale down your requirement to a simple true or false statement, it can make it harder for a tester to understand and it will ultimately have a negative effect. The developers will have a hard time understanding them too. This rarely happens as the requirements are mainly written with consideration for the developers.

When have we tested enough?

How does a tester know when he has tested enough? Will more testing result in more critical bugs being uncovered? Are there enough resources and is there enough time to find them? There are a lot of considerations to have in mind. Today software is so complex that it is often released even though it is a well-known fact that they contain bugs. There just is not enough time to find and correct them all. So how do we decide when we have tested enough?

There is no general method that answers these questions. Assessments have to be made on a lot of issues in order to be as accurate as possible. But for this thesis I have chosen to boil it down to for questions that have to be analyzed before one can make a final decision (Bach, James; 1998):

1. Assessment of resources – Do the benefits of the software overweight the problems? When you think about this you have to consider a lot of perspectives, like the opinion of stakeholders, consequences of failure, time frames, importance for the organization etc.
2. Costs of testing – How much is the testing costing the project in terms of time and resources? Are the tests delivering an acceptable amount of information about the software per tests?
3. Decisions – If the testing has no effect on the outcome or decisions made, there is not any point in continuing doing it. Testing should be a decisive part of the development process.
4. Timing of all the above – For a tester, every project will have a time constraint. All these things have a deadline there are always compromises that have to be made.

Reflecting on these questions for a while will at the least give a good idea on what the next move could be.

Right results – Actual results
How do we know what output is correct? For a tester that is not extremely familiar with the system, it can be hard to know what the right output should be, especially when there are a lot of different types of tests to perform and the tester knows little about how the software is supposed to work. There are however different methods that can be helpful in some situations.

The most effective method is to automate the process of evaluating the correctness of the output. This method is called direct verification and needs good specification that explains exactly what the output should be for it to be effective. This method however is not always applicable and writing output checkers for a complex system can be as difficult and time consuming as writing the actual code for the software.

Another great method is to compare the output with some other output that was created by some correctly working software. This is a great method to perform on your regression tests, if the old version works correctly. There is however the risk that both outcomes are incorrect, and in that case, the testers would not notice it.

Test environments and test data

Test environments
There exist several levels of environments. There are the development environments where the developers test their code. Then there are test environments where the testers do their work, which include system and integration testing. And finally there are the end production environments that will be used by the users. If the software is complex there may be a need for several small test environments to perform system tests. Here you can test each module individually, and finally you have a big test environment where you

perform system, integration and acceptance testing. High level test environments are often called staging environments, which is the place the software goes right before being released, (Everett, Gerald D., McLeod, Raymond Jr.; 2007).

Test environments are the most common way to test software. The point is to set up a separate computing environment that is similar to the final product environment. The point of this is to generate real production behavior and interpret the results. These results should be the same that final user will experience.

Developers have different environments for each version of the software that they create. There they perform all the white box unit and module testing that is required of them before they send the build to the testers.

The more the testing environment behaves like the final product environment, the more accurate the testing results will be. That is because they are more likely to have similar results and therefore find the same bugs that exist in the production environment. In order to get the best results both the hardware and the software of the test environment should as similar to the production system as possible. Some factors that can be decisive are CPU speed, Network speed, monitor screen refresh rate etc. This is extremely important during acceptance testing.

Test environments are made by the development team. System and integration testing environments usually do not require more than the necessary functionality, a simple GUI that only displays what is needed. When they developed the environments the control is handed to the testers, this is important because then they can control what version of the software is being used and what test data also. They can also control that changes are made in the software and not directly to the test environment. This situation may seem farfetched, but have happened (intentionally or unintentionally) when development is stressed.

Test data

Test data is data used as input into the test environments. It is used to test the boundaries of the software with extreme values and also test common values that are similar to the expected production data.

In complex software development, it can be difficult for a tester to know what test data to use, as he might not know how the software will be used. In that case, testers can bring in business experts or clients to help generate the right input. They can help to generate test data that represents accurate business data and also show how it should be uploaded to the software and then reloaded after the test has been executed. This could also be done by the clients themselves during acceptance testing.

Stubs are normally used to replace missing components, by generating values similar to the ones that would otherwise be given. This is done by specifying some criteria of what should be expected. But they can also be used to generate test data. This is usually done at a low-level of testing. As you get higher there are demands for more realistic values than does created by stubs.

Testing Techniques

Risk-based testing

As time constraints make it impossible for testers to perform full coverage testing (testing every possible input or combinations of inputs) on software, compromises have to be made. Testers have to expect that not all requirements will be fully tested. Usually this means that more time is given to the most important requirements. These requirements are usually called high-risk requirements. So how do testers know which ones are high-risk and which one are low-risk? This is an important part of the testing process and is referred to as risk management.

Risk management is not just about assigning testers to do different types of tests. For example, assign three testers to perform scalability tests and one to perform penetration tests. Some managers think that is all that is needed, but an even better idea is to start earlier and creating a risk-based test design.

When a test leader or analyst is in the process of creating a test design you should gather all the risks you can think of and make a list of them. With a clear view of potential risks the tester is more likely to look for high-risk faults with the right tests.

The risk list created should have specific criteria. The risk should only explain problems that could occur with the product, and be related to some requirement. If it is not, then that could mean that a requirement is missing.

Risk-based testing is a self-correcting process. When a new inexperienced tester is supposed to test a new software product, he really does not know what he is supposed to be looking for, which faults are business critical and which ones are not? It is natural to that some big faults slip thru. When that happens it is important to learn from those experiences and be better prepared for the next time.

It is important when you are making your test design that you realize that risk-based testing is all about intuition. Those intuitions might be completely wrong, and that is a good reason not to put all your efforts on performing risk-based tests, or any other specific strategy. Instead, make sure that some of the testing is based on functional coverage, or exploratory testing.

Black box/White box

When testing software functionality there are two different techniques of testing to choose. In black box testing you only look at the output and compare it to what is expected or adequate results. With white box testing you dig deeper, checking the internals of the software making sure that the software is not only giving the right outputs, but also that it is doing it in the correct and intended way.

These differences lead to several advantages and disadvantages depending on what tests that needs to be performed.

Black box testing - When you are only interested in checking the functionality of the software this is the best choice. The majority of tests are for checking functionality. Examples of such tests are stress and maintainability tests.

White box testing - If there is a need to look at the structural mechanisms of the software. This is done by checking that the code is correct line by line through path coverage or logical coverage.

Generally, white box is best used for checking the correctness in the software. It is a good way to do software maintenance, looking for dead code and checking that algorithms are correctly defined. White box testing requires that a lot more resources are utilized.

Black box testing is the more used of these two. It can be applied for most types of test and requires less time and resources than white box testing. The disadvantages are that even though the software may be working correctly, there is no guarantee that the internals are working as they are supposed to. This means that a fault could be hidden in some test case that has not been tried. Generally white box is performed by the developers at a unit and module level of testing while when black box is used when the build reaches the test team.

Exploratory testing

Exploratory testing is testing without a script. Instead of just following plans, cases and scripts, the tester is set free and given permission to explore the software and find faults. When you are doing exploratory testing you are actually doing the second and third step of the test process simultaneously. That is, design, analysis and execution. It requires spontaneity from the tester, as the test is not completely pre-defined and seldom is turns out according to the original plan.

Requirements can be extremely precise and specific, but if it turns out that something has been left out then a scripted test would not catch that bug, and a tester would probably not detect it. It could be something very obvious, but the tester is more likely to miss it when he is following the steps in a test case.

Although exploratory testing gives the tester a lot more freedom, there still are constraints and documentation on the tests that needs to be performed. It starts by creating a list (a test chart) about everything that needs to be tested, an indication on how each test could be performed and what results to expect. This documentation turns out to be very similar to the results from a corresponding test script.

As test cases are not written, there is no way to evaluate them before the tests are done. That means that incomplete requirements or implementations cannot be caught in an early stage, which is a big downside.

Some people may confuse this type of testing with ad hoc testing but there are clear differences. Ad hoc testing is almost completely improvised testing. That means that everyone who uses the software is actually doing ad hoc testing. Ad hoc and scripted test can be seen as two opposite poles of extreme testing where exploratory falls in the middle, but closer to ad hoc. Exploratory testing is much more thought out and documented compared to ad hoc testing. The huge flaw with ad hoc is that the lack of documentation makes the results extremely hard to recreate.

When the tester's current test is being affected by the results of the previous one, then he is performing exploratory testing. It is appropriate when the tester cannot tell what tests he should perform or in what order. Some early testing may show that another test strategy is more efficient. But there is not usually much uncertainty about what should be tested, so it is often generally more efficient and reliable to use scripted tests

Systems also tend to become immune against scripts, so it is a good idea to sometimes leave it and do some exploring. These test cases are rarely performed in the exact same way, so that increases the possibility to find new defects.

Another reason to use exploratory testing is that scripted test tends to disrupt the interactive process that makes the tester think more about what is happening in the product and finding important bugs quickly. This is an important point because tester should be encouraged to understand the product that they are testing. Exploratory testing is the best practice when efficiency and repeatability is important.

Test automation

Test automation is basically when the execution part of the process is performed automatically by a computer. It sounds very compelling, but has a lot of pitfalls. If not performed properly, automation can result to be useless and a waste of time.

Automated testing is best applied to tests that cannot be performed manually, or are extremely resource consuming. For example, load tests would otherwise require hundreds or thousands of testers to perform. They can also be applied to tests that require a lot of repeated work, generating test data is an example of this.

Before a tester decides to perform any automation, he needs to have a good test design. If not he may end up with automated tests that are easy to set up, but are bad at finding defects.

Instead of trying to automate directly, testers should make a couple of manual tests and get a better idea the appropriate testing procedure. Then he will have a sense of what is an appropriate decision and find any testability issues that may exist. Otherwise there is a risk that he finds a more cost-effective procedure later on and the work that has been put on automation has gone to waste.

Before deciding if automation is needed, testers must consider some issues (Kaner, Cem; Bach, James and Pettichord, Bret; 2002):

1. There are usually a lot of requirements when software comes to the testing phase. But there are usually some requirements that are more important than others. If these require a lot of tests with different values (data driven), then this is a good

candidate for automation. Manual testing can be applied on the rest of the requirements.

2. Evaluate if the software is in good conditions to automate. Important factors here are software architecture and important components, software code language and environments. How all these things interact determine how the interface is used. A too complex system can turn out to be difficult to automate.
3. How good are the testers programming skills is also a factor. To do test automation is to develop software. The skills of the testers determine if automation is a good investment.

Test automation requires good routines and a proper testing process. You cannot expect to have a good automation if the requirements for that test suite keep changing. It also requires a great deal of structure when it comes to the layup of the testing process. If it is not organized, the automation will not succeed.

Many argue that GUI testing is a natural candidate for automation. But this requires constant maintenance as GUI is always changing. If the changes are too big, the automation needs to be redone from scratch. The best thing to do is to abstract the interface in your automation, making it resistant and flexible to changes, and not dependent on specific interface attributes. This can be done by for example, creating window maps or task libraries.

Another candidate is regression testing. Unfortunately this also has its pitfalls. The percentage of bugs found in automatic regression tests is often very low. This is because automated regressions tests often die prematurely, primarily because of GUI changes. Because of this a lot of tests that used to pass will fail even though nothing obvious has been changed. Automated tests are extremely fragile for changes (Bach, James; 2002).

The best situation for test automation is when there is a need to do a lot of data driven tests. Data driven test are tests that needs to be performed many times with different variables. These tests can be automated and the values can either be generated randomly or predetermined by the tester.

The cost of maintaining automation can get really high and you do not want the testers to spend more time diagnosing and repairing automation then they are actually performing tests. Automation maintenance is probably the biggest issue that testers will have when automating their tests suites. If you automate things you wished that you had time to test, you probably will not have the time to maintain the automation. Most test automations break down sooner or later so it should be important enough that you know that you will invest time on it. Automation is an iterative process.

Process study at 'the bank'

Introduction

My goal with this thesis was to evaluate the testing process, including the tools that are used. In order to do this I had the opportunity to follow the test team and their progress in their current projects. During this period, I had access to their test management tool so that I could see how it is used and study old project material.

Types of tests

Most of the development is being outsourced to India. There is also some testing being performed there. The developers do not have access to the bank's core system so they cannot perform system or system integration testing. These tests are done in Sweden along with the acceptance testing.

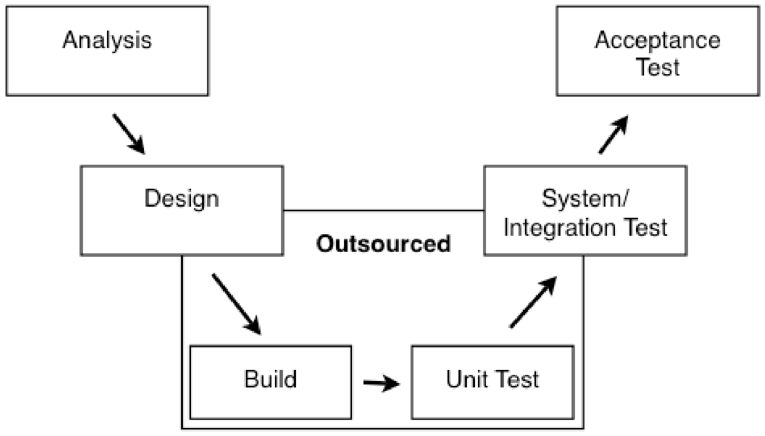


Fig. 3: V-model with outsourcing

System testing – Determines if the system meets the requirements. System testing assumes that every module work independently and that the system works as a whole. Runs the entire system from end to end to ensure that it works as expected, with meaningful results.

Integration testing – Checks if the interfaces between the modules connect to each other as expected. Also looks at high level condition and boundaries, and checks that the functionality works as expected.

Acceptance testing – Test the full system from end to end, to ensure that the software is ready for production. This can be performed by the client or a business specialist if necessary. It is the client who must determine that the requirements are met by the software.

In the bank, most of the testing consists of functional black-box testing. Acceptance tests are done by the people who ordered the product or update. Even so, it is the testers at IT who are the driving force in these tests. They provide all the data and support needed to perform these tests.

Testing Process

The tests are performed according to this model. Every release or project goes through this process.

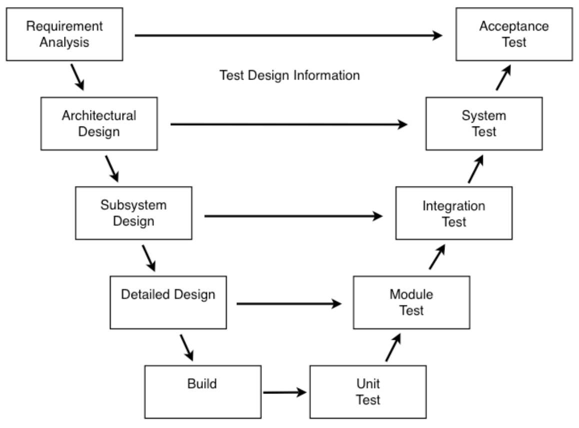


Fig. 4: The V-model

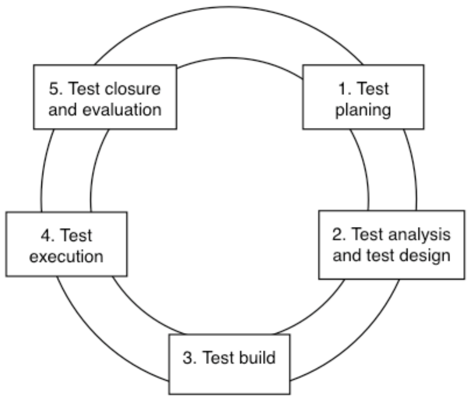

5. Test closure and evaluation
1. Test planing
4. Test execution
2. Test analysis and test design
3. Test build

Fig 5. The test cycle

The planning is made by the test leader. It is his job to hand out work assignments and schedule tests. He usually does this by assigning test by competence, in order for the testers to work as effective as possible. The testers have the responsibility to write the test design which is often the same thing as the test analysis document. The tests design contains the test cases and is supposed to be written before the software is received and should be based on the requirements given.

Test cases include step actions and all the necessary input data needed to execute the tests. The step actions can vary a great deal in detail depending on how specific the requirements are. Work on test cases usually starts early on in a project. They are then updated as new specifications are revealed, along with the test plan.

When a test execution has failed the step actions are documented. They may include information to recreate the defect like screenshots showing the defect. And finally some form of description.

The executions are done on test environments that resemble the product environments as much as possible. When for some reasons the GUI is not available, a simple test environment is created by the developers that only include essential data. The core system is terminal-based software and all the new systems are applied on top of that software. The test environments show how the communication between these two works.

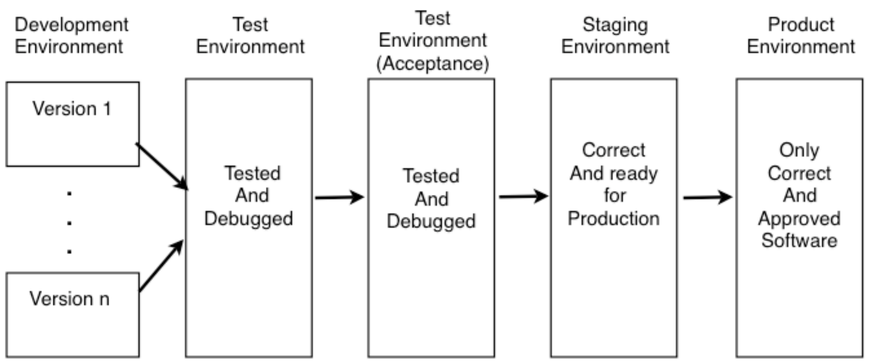


Fig 6. The different levels of testing environments

Instead of using status reports, the testers present their progress in the test team's weekly meeting. Any issues or concerns are to be reported to the test manager, who documents it. Sometimes a status report is written. In that case they are written for the project leader or system manager.

A final test report is written at the end of every test cycle. It can be written before or after the deadline. The report describes how the testing has been and what problems occurred. What has been missed during testing, what has been tested too much and what could have gone more successfully using another strategy.

Testing techniques

When the new software has been received along with the matching test environment it is time to start testing. When defects are found they are documented and reported to the developers. When fixes come in they have to be retested. The communication between testers and developers is usually done through QC (the testing tool). Acceptance testing is performed by the tester to some extent, but more than often they are done by the users or business experts that have ordered the software. It is the tester's job to prepare the test environment, deliver the needed test data and explain how the software works to the clients.

All the testing performed is more or less risk-based. It is critical for a bank to find business-critical bugs in the software, and fast. This could be things like functionality that is directly used by costumers and that will have a direct impact if it fails. Risk-based approaches are often implemented in the design stage, and a priority-system is used to determine which tests are the most critical. This risk evaluation changes throughout testing as more information is uncovered thanks to defects. A defect could maybe turn out to be so critical that further tests have to be performed in that part of the system.

The techniques to perform the testing depend on the testers. Some are very script-based and use QC to do most of their work. Others take a more exploratory approach to testing, and do most of the work through other tools. Usually no tester works with just one technique. Sometimes a tester will notice some defect while following a script for another one. In that case, he has unintentionally performed a successful exploratory testing session. When time constraints are a big factor, exploratory testing is common practice.

When writing test cases in QC, it is important to include the inputs necessary to complete the test without giving exact test data. It should just include a description of what the test data needs to be.

Roles
This is the current structure of the test team at the bank:

Test team manager - Writes the test plan and work duties to the team members, also performs testing. He also provides the clients or business experts with the information and support necessary to perform acceptance tests. Sets the test policies and communicates with other managers involved in the projects.

Test engineers - Designs, builds and deploys the tests. Analyzes and reports results to the developers and the test leader. Work on projects depending on their expertise.

In the bank the tester work very independently from each other, as they seldom work on the same projects. This means that the test engineers more than often write their own test plans and are in charge of their own testing without input from the test manager.

Test environments
In the bank, like in the literature, there are different levels of environments depending on what needs to be done. The developers have their own environments. This may differ depending of what version of the software that is in use. The testers are given two separate environments for functional tests and acceptance tests.

The testing tool

Description
The testing tool used in the bank is Quality Center (QC) by HP. It is a tool that is used for organizing and managing all the parts of the testing process, from planning projects to tracking defects. This tool is web-based and can be used by the whole business. It also supports communication between testers and developers.

Basically, a test case is written in QC, deployed in a test environment and the results are then documented in QC where the developers can view them.

Managing test scripts
The tool does most of the managing through organizing cycles, requirements, scripts, tests and defects with some sort of naming convention or disk file structure. All these things can then be linked together so that the progress can be followed from end to end.

Requirements
With the help of QC, the tester can specify how many resources that should be invested in different requirements. This can be done by declaring business criticality and failure probability for each requirement. Business criticality states how important this requirement is for the business. Failure probability tries to measure how likely the requirement is to fail. This is based on the complexity of the requirement.

Requirement analysts can also import requirements that where written in regular text files to the tool. But this does not give you the advantage of an automatic message to the testers and developers through a flag or mail when a change has been made.

Managing cases and scripts
QC allows you to manage both manual tests and automatic tests. If you test manually, there is support for planning the tests step by step and then start testing fast. If you want to create and automatic test scripts you have to create it with another tool and import it or the results of that script. These tools are available in the bank but as there is little automation implemented, these tools are rarely used.

Test cases are written directly in to the tool and include information like name and ID, designer, creation date, status, a description about the case, comments, step actions and expected results.

Whether tests are performed manually or automatically, the expected results need to be put in by hand. When the test has been performed, the tester must check the results against the expected results, so there is no form of automated direct verification of the output.

When a design has been made, a tester can decide which tests can be automated. The automation is done in another tool called Quick Test Professional. It is also a tool created by HP. The tester can generate a test script in QC and then complete it using Quick Test Pro. This is however not how the automation is performed in the bank. If the testers want to automate something it is outsourced to the test resource in India. So there is little knowledge about automation among the testing team.

Report analysis
With QC you can create reports and graphs easily under any time of development. They include information about requirements, defects, and statistics based on the time and dates that these thing where handled. Every report is then stored for future reference.

Other tools
If some test automation is carried out it is done with Quick Test Professional, which is also developed by HP. This is a capture/replay tool that records the steps that the user is performing and then plays back those steps automatically. You can then easily export all the results back to QC. As they are created by the same manufacture, they offer synchronized tool-to-tool communication.

Evaluation
One of the goals of the thesis was to evaluate the testing tool used at the bank. During my time at the bank I had the chance to study it and analyze it. Here are the results.

Usability
The tool is web-based which means it is not dependent of any particular OS. It is automatically updated when changes have been made by some other tester or developer.

QC is not a difficult tool to use. When I first got access to it, I also got an introduction on how to use it by the test leader. I also studied the manual. The essentials are easy to understand and for an experienced user this should be no problem at all.

Testers can send messages through QC to the developer that it concerns and vice versa. This allows them to follow the progress without missing valuable information. This feature is not dependent of the tool, which means that a developer does not have to work with QC and still receive information about the project

Pros and cons

One of the mayor pros with the tool is the high quality of communication between the testers and developers. Everything that has been done by someone is stamped with that person's signature. Defects are test and defects are all easily traceable which makes it easy for someone to find out if all tests have been performed and if all defects have been reported, fixed or retested.

Today, only three of the eight modules in QC that are directly connected to testing and that are used by the test team. Even though they are not supposed to use them all, this amount is too little. The rest of the modules are intended to be used by analyst and developers. If they were used, the testers could then use them to collect information more effectively. The whole process could benefit from using QC and it would allow smoother collaborations between the different departments.

Surrounding issues

The requirements are written in text documents. This gives the testers a lot of extra work analyzing these documents every time they are updated. When a requirement specification is changed (which happens a lot), the tester has to read through the whole document and try to find the change which has not been flagged or highlighted. QC has a great requirement management function for this single purpose. It gives the testers and developers a message when a change has been made and also points out what that change is.

The development team in India also consists of a tester that executes tests on builds before they are sent to Sweden. However he has not access to the needed test data to perform test cases, so he has to ask the test team for it. The test team in Sweden wants this control as they can be in charge what test data is used. As mentioned, the test data is not endless and giving him full access would risk using them all up on low priority tests. Today however, the collaboration between him and the testers in Sweden is not optimal. There is a risk that the same test cases will be performed on both sides. There are plans for the test leader to fly to India and optimize the testing process.

Potential improvements

Today, the developers are not using the QC as their primary test management tool. They use other tools for other projects and only QC when they work with the bank. This is not a great situation in terms of communication. On top of that, some testers use an old tool that is tightly implemented with the backend system and has not fully crossed over to QC. This means that documentation has to be done in both tools. If communication is handled through different tools, mistakes will be made. The ideal situation would be that everybody involved used the same tool.

There is also room for more use of some of the functionality of the test managing tool. Right now there are three modules being used by the testers. And these three are also the ones that are the most test-related. In order for the testers to have use of the rest of the modules there is a need for bigger parts of the project being done in QC.

There is still room for creating better reports with statistics based on the tests and defect documented in QC. The basic information is already there. What is needed from the tester is better documentation of the dates when test where planed and then performed. It is also necessary for the analysts to start writing requirements in QC.

The new updated version of QC offers a whole new set of futures for requirements. With very basic information a test manager can easily find out exactly how much time that he should spend on each. These predictions can be completely based on risk and resource constraints, if necessary.

If the requirements are not written in QC, there is no way to link the tests to the requirements. This is one of the big advantages of using QC. In order to get the most out of this tool there is a need to get the people who write the requirements to use the tool too.

Improving the process

Static testing

During development it is extremely helpful if testers are included when writing requirement specifications. Even though it is not the tester's job to verify the specification, this would happen naturally if the tester during the analysis phase would do some checks, called static tests (Everett, Gerald D., McLeod, Raymond Jr.; 2007 and Bornelind, Robert; 2011).

According to Everett, McLeod and Bornelind, numerous defects are introduced during the requirements phases. Requirements are sent to developers without any form of inspection. The outsourced development team receives the requirements and assumes that they are sufficient. These defects will not be detected until testing. Defects that are discovered in late stages of development are a lot more expensive and time consuming then they would be if they were corrected early on.

There are numerous static tests that can be used to check if requirements are a "good enough" description of the system:

- Check that all the technical terms are defined and that every reference to them matches the definition.
- Are the requirements relevant to the software?
- Are the requirements "tagged" to all the parts in the system that it will affect?
- Have all the stakeholders given their input on priorities?

It is important for the tester to understand the purpose of the requirement in order to start writing test cases early.

Requirements

Not only the testers have time constraints, the whole development process is often rushed and when people do not have time to do their work mistakes are made. This is the case when the requirements are written too. They have to deliver detailed specification to both the developers and tester. This is difficult because they both need different thing of the requirements and there is no time to write one for each. The requirements are generally written for the developers, with the input from the testers. It is then the tester's job to find out what the client needs out of them.

There are other useful ways to determine what the most important needs are besides the requirement documentation (Bach, James; 2002):

1. Talk to the people that have important opinions.
2. Draw conclusions about what requirements that matter by looking at other things you know about the software.
3. Discover the specifications implicitly and explicitly and do the testing based on that information.

The last point can be a little tricky to understand. It simply means that some requirements are not very clear and some might be implied but not plainly expressed. Instead the tester has to intuit what they are by experience. These experiences may come from:

- Trying out competing/related products
- Using older versions of the same products
- Email discussions about the project
- Comments from the costumers
- The testers experience

When a tester presents an acknowledgement that has not been brought up in any requirements or specification he may get a response like "this is not in the requirements, but it should be." The tester probably also needs to motivate it because it does not come from an analyst or stakeholder. It could be like this "F2 is the paste command in all of our competitions software. We should also do it like this to not confuse our users."

Roles

The test manager is supposed to allocate the task of the test leader. Because of the little workforce in the test team, there are no clear distinctions in work roles, as presented in the roles documentation. Here is a proposed structure of work roles that could be applied:

Test manager (team leader) - Creates test plans, organizes workload and preparations. Writes the final test reports and follows the progress. He also helps out with the testing when needed.

Test designer (test analyst) - Analyzes and evaluates requirements. Creates test specifications. He can also help out with testing.

Test administrator (test analyst) - Creates test environments and test data. He also specifies criteria for the test cases.

Test engineers - Designs documents and runs test cases using the documentation. He records and documents the results.

In a small team there is not a big need to have different roles and responsibilities. A test engineer can do most things in the test cycle by himself. In 'Introduction to Software testing' (Ammann,Paul and Offutt, James; 2008), they describe a test team being a lot like the one in the bank, consisting of a team leader and some test engineers. I do however think that there is a need for someone to more closely analyze the requirements and verify that they are sufficient for testing. This however is not possible if there only are a few testers. In order to have more roles and build different skills there is a need to increase the amount of testers.

It is very healthy to have a mix if different skills in a team, but also individually. It can turn out to be extremely expensive for an organization to lose someone that has unique skills about the system or process at hand. This is a potential risk that many companies live with. An easy way for tester to learn different skills would be to work in pairs for some sessions or switching work roles more often. This is simply called session testing (Bach, James; 2002).

The testers need to have a good background in the functionality of the system in order to be able to test it. They need to know how the banking system works and what output is correct output. And when a test demands too much business intelligence, the testers can consult business experts for further information.

Testing techniques

There is a good balance of exploratory and scripted testing in the bank. In the literature the general opinion is that exploratory should complement scripted testing, and this is exactly what is happening in the bank today. Tips to remember are that exploratory testing is especially useful during initiation, and when an area is thought to be low-risk.

White box testing is basically unit testing and is commonly performed by programmers while black box testing is functional testing and is performed by testers. Gray box testing is when a tester has some knowledge about the code and thereby has a better intuition on where a risk may be present. I am not suggesting that testers should sit down and analyze the code in detail, but if a tester knows how the modules work and interact, he should put that knowledge to use. Gray box testing is important when testing internet applications, where there are a lot of loosely connected modules (Kaner, Cem; Bach, James and Pettichord, Bret; 2002).

No technique is so good that it does not need to borrow methods from another one. Even if a tester is working script-based, he will intuitively sometimes leave the script to explore some other potential defect. This is also good risk-based testing and should be encouraged. It is called "*The principle of diverse half-measures*: use a diversity of methods, because no single heuristic always works" (Bach, James; 1999).

Today, the time given to the tester between when they received a build and when the release date is can come to be extremely short. Even though everyone knows that tester can only test so much in that time frame, it becomes extremely frustrating for the tester if he feels that he cannot do his job correctly and reach satisfying result.

Test automation and test data

There is little work put on automating tests today. A lot of test today are repeated on a regular basis, such on for example GUI tests. These could possibly cut resource costs if automation proved to be successive. In a business where testing is given less time automation becomes an absolute necessity. In all the literature I have read preparing me for this thesis the general opinion is that some form of test automation is absolutely necessary for several reasons. Minimizing human error and making regression testing easier being among the most important ones. (Ammann, Paul and Offutt, James; 2008).

It would be great if data from the production environment was made available in the testing environments. It would give the test team a lot more test data. The test data is based on accounts. Therefore, the amount of accounts is a crucial issue if they want to implement automatic tests as they require a lot of test data. The amount of test data is also related to the quality of the testing. With a lot of test data, more tests can be performed and the test coverage becomes bigger.

Testers do not have direct access to accounts. When they want to perform some tests, they order the accounts based on some criteria specified in the test cases. One of the big issues today is that the test team has to make that order before the test cases are complete. This is because it takes a couple of weeks for that order to be delivered. And if the testers do not have time to finish the test cases they cannot write good test data criteria. Today there are plans to improve this situation. The test team needs either more time to write test data criteria or the delivery needs to be faster.

Outsourcing build verification testing

For the test resource in India I propose that he should be in charge off build verification testing. There are several names for this type of testing (smoke test, sanity check, release acceptance testing etc.) but they all have the same basic purpose, which is to test the basic functionality of the release. This is done through a relatively small set of test cases that ensures that the release is stable enough to start testing on. In the book 'Lessons learned in Software testing' (Kaner, Cem; Bach, James and Pettichord, Bret; 2002) they say "rejection of the build is typically automatic when a smoke test fails." So if the build does not pass the BVT, the build should not be sent to Sweden and it becomes the developer's highest priority to fix it. BVT has become a common practice when development gets outsourced.

BVT is very similar to a module level of regression testing, and could be made automatic, where the tester would be in charge of maintenance of that script, or it could be done manually. The important issue is to check that all the new and updated file versions are included in the build and work, (What you need to know about BVT, 2008).

This way of working would be beneficial for several reasons. It would mean that the test resource is more separate from the test team and thus does not need to synchronize on a daily basis with the rest of the testing team. The test cases would be more fixed and the test team would have a better idea of what is being done there. There would also be no need for the test resource to request test data based on specific criteria as he is mainly doing regression testing and can therefore reuse his test data.

Conclusions

In this section I discuss some of the things learned about the testing process while writing my thesis at the bank. These are more general thoughts and I think can be applied to any test teams process.

Missing step in the testing process

There is one last step in the testing process that normally is not mentioned in the literature as a part of the process. This is the evaluation step. This step is extremely important and part of the test managers duties. By analyzing the results of testing after the release, you can learn from what went right and wrong. What should have been tested more and should have been tested less. Should we have used other testing techniques or focused on other types of tests.

Black box testing makes you think ”outside the box”

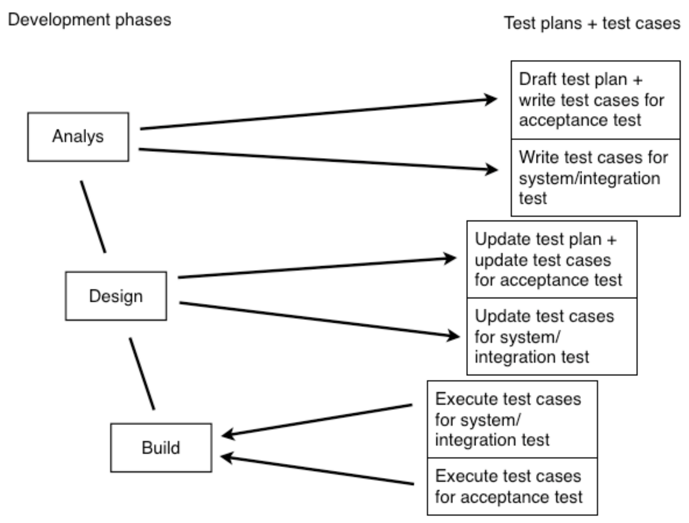
Black box testing is the most common way to test. The cost of resources when doing white box testing is just too high. The big advantage of black box testing is that the testers probably think differently than the developers, and, thus are likely to anticipate risks that the programmer might have missed.

The importance of exploratory testing

There is no perfect technique that finds all possible bugs when there are hard time constraints. They all have different effect based on what kind of software is tested and what kind the experience of the tester. What has become very clear is that exploratory testing in a great complement to scripted testing. By doing both you will have the advantage of finding all the faults that where predefined by the test plan, while still putting the testers intuition and know-how of the system to find bugs that had not been thought off .

Change in the requirements – late test cases

A tester is not going to spend time writing specified test if the requirements are going to be changed in the last minute. This is often a reality in development. This means that the testers often have a very little time frame to write the test cases, with may lead to a poorer quality of testing. For the testers this means that they should write test cases and test plans carefully in the beginning. When the project progresses, the test cases are updated and become more detailed.



Development phases
Test plans + test cases
Analys
Design
Build
Draft test plan + write test cases for acceptance test
Write test cases for system/integration test
Update test plan + update test cases for acceptance test
Update test cases for system/ integration test
Execute test cases for system/ integration test
Execute test cases for acceptance test

Fig. 7: Workflow through development for testers

Do not postpone testing

It is a common mistake to postpone testing until the final phases in the development process, when the software has already started to be implemented, or even when it has ended. The testers are left with a short time frame and few resources to do their work. All the planning and designing gets rushed through and not given time it needs to be analyzed. Testers cannot be expected to make bad software good in the last minute.

Test automation

There are a lot of different opinions about test automation. Some say for example that regression testing without automation is equivalent to no testing at all. The truth is that automating requires a lot of resources and maintenance. Automation is a full time job and requires a team that has someone who can work with it on a regular basis, otherwise the costs of resources will be too high.

Do not put too much weight in writing documentation

There is a lot of documentation that need to be written according to the litterateur. Most of this never gets read. Depending on how the organization works, look at what documentation is being read and what is not, and dismiss everything that is not useful. Otherwise you are just spending valuable time that could be used for testing.

Sometimes Waterfall is necessary

The big downsides with working with the waterfall model instead of working agile is that a lot of time is wasted on documentation that will never be read and that late changes costs a lot. The more stakeholders there are, the more difficult it becomes to work with agile methodologies. When many people need to give their input, it slows down the process.

Method

In order to compile the needed information for this thesis I used several different ways in order to do so. Here are the ways that I have gathered the information needed to write this thesis.

Planning

During the course of this semester I have kept a diary of what I have done that day and what needs to be done during the coming days. The diary entries where made each day at the end of the day.

Meetings

I took part of all the test-team meetings, some of the IT meetings and even some staff meetings. This has been a great source of information. In the meetings there is open discussion about what the work that is being done. Here I also presented my progress, issues and coming plans.

Literary study

Before the start of the thesis I took part of a course in testing methodology in Uppsala University. I did so in order to get an introduction to testing. Besides the course book, which I of course read, I also read several other books and articles related to the subject of software testing.

In order to study the test management tool I was given access the official manuals, and access to the tool and an old project from 2010. This was very valuable to the research as I could look at test cases, performed test, defects etc.

Interviews

I also made several interviews not only with the test team but other IT-experts related to the testing process. Most interviews where preformed on the test manager which was extremely valuable as he had a lot of information regarding all the issues.

Analysis and discussion

Here is where I discuss the good and the bad event that occurred during the time of the thesis. How well have these issues been solved? And what new issues where found throughout that time, and why.

Based on the conditions I had during my research I think the thesis has gone very well. Before I started my thesis at the bank I was pretty well-read in the subject of software testing. I had taken a course in the subject and had also started reading books and articles before the start of the thesis.

One of the best experiences was attending all the meetings, which was a great source of information. Many things that were unclear where at the beginning where discussed and explained during these meetings.

I have not followed the project plan that closely when it comes to the interviews. That is to say, instead of preparing interviews, performing them and then analyzing them I have mixed all these steps and basically prepared interviews a day or two before the interview. At the same time I had already begun writing on a draft.

One thing that has not been mentioned in thesis that could have been interesting to investigate is how other banks handle their testing and compare the processes. I think however that it would have been difficult as they would have nothing to gain by disclosing something potentially sensitive to someone who is doing his thesis for another bank.

One of the things that I had intentionally planned on studying more in detail was how automated testing was handled in the bank. This however quickly turned out to be infeasible as automation performed is ordered from the development team in India. Instead of automation I wrote more on the issue of writing good requirements with the testers in mind and the importance of including them in the process.

The biggest mistake made during the thesis was not attending a meeting at SAST (Swedish Association of Software Testing) in February. It could have been extremely valuable for the thesis as the theme of the meeting was test leadership. The reason I missed it was simply because I was not aware that they existed at that point in time. I have since then studied the PowerPoint presentations that are available at their home page.

The best thing with doing this thesis at the bank has been that the testing process is not optimized, so this has given me the opportunity to give advice on potential improvements. Hopefully the bank will have good use of them and implement some of them in their work.

Literature

1. Everett, Gerald D.; McLeod, Raymond Jr.; 2007: “Software testing; testing across the entire development life cycle”

2. Kaner, Cem; Bach, James; Pettichord, Bret; 2002: “Lessons learned in Software testing: A context driven approach”

3. Ammann, Paul; Offutt, James; 2008; “Introduction to software testing”

4. Bach, James; 1998; “A framework for good enough testing” (Last checked: April 1, 2011) http://www.satisfice.com/articles/gooden2.pdf

5. Bach, James; 2002;”Exploratory testing explained” (Last checked: April 1, 2011) http://www.satisfice.com/articles/et-article.pdf

6. Bach, James; 1999; “Heuristic Risk-Based testing” (Last checked: April 1, 2011) http://www.satisfice.com/articles/hrbt.pdf

7. Bach, James; 1999; “Risk and requirements-based Testing” (Last checked: April 1, 2011) http://www.satisfice.com/articles/requirements_based_testing.pdf

8. Robert Bornelind; 24 February 2011; Presentation på SAST; (Last checked: April 1, 2011) http://sast.se/q-moten/2011/stockholm/q1/2011_q1_bornelind.pdf

9. What you need to know about BVT; (Last checked: April 1, 2011) http://www.softwaretestinghelp.com/bvt-build-verification-testing-process

Numbers 1 and 3 have been a big source of reference in regards to what steps there are in the test process and how they are connected to development. They differ in context but present the same basic idea which I have summed up in part 2.

Many of the articles written by James Bach have been used to write this thesis. He presents a lot of methodologies and techniques that where very relevant in this thesis.

Acknowledgements

Thanks to everyone in the IT-department at the bank for making this thesis possible. The terrific test team at the bank has given me strong support and insight in every aspect of this thesis.

Thanks to Pär Johansson who supervised my work and ensured that it ran according to plan. His insight and expertise was essential to the development of this thesis.