LUNDS
UNIVERSITET
Lunds Tekniska Högskola

Exploratory testing with the help of a test tool

-Can a tool show what a tester have been testing?

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg

Bachelor thesis:

Karl Larsson

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2011

Abstract

Exploratory testing (ET) is a test technique used to test software. It is a free way (compared to scripted test) of testing, where the tester takes notes during the test session. The tester has the option to select his own way when testing the software, he learns and tests at the same time. Notes are the result from the test session and are used to see what has been tested, e.g. the outcome and the test data. The notes also become the test case.

Because the notes are important, the idea was to find a tool that can help the tester taking notes, but not to replace the tester. The tool is meant to make the notes better - more information without increasing the work for the tester. The software in this case are web-pages. A web-page can show information to an external user but today more and more web-pages are interacting with the user. It could be products to buy, booking a trip or online banking. The first step was to find and investigate tools that already exists on the market. Specific criteria for the tools was set during the investigation and they were meant to make exploratory testing stronger, to way up for the claimed weak parts of ET. The conclusion was that none of the tested tools fulfilled all of the criteria, the missing part was how to collect information from the testing without making the tester write everything down. To get a tool that can automatically collect information, a Firefox add-on was implemented. The Firefox add-on can collect information from the testing automatically in order for the tester to concentrate on the actual testing, learning and validating the software.

The Firefox add-on that was implemented is a prototype to test the concept of logging in- and outputs from a web-page during exploratory testing. ETnote is the work name for the prototype.

Keywords: Exploratory Testing, Tool, web-pages, firefox add-on

Sammanfattning

Utforskande testning är en teknik för att testa mjukvara. Det är ett fritt sätt att testa (jämfört med testfall), där testaren tar anteckningar under testningen. Anteckningarna är en del av resultatet efter en testomgång och påvisar vad som har testats, till exempel resultatet och testdata, och blir sedan teststegen. Kunskap av produkten är ett annat resultat.

Eftersom anteckningar är så viktiga är tanken att hitta ett verktyg som kan hjälpa testaren, inte ersätta honom. Verktyget ska göra anteckningarna bättre - innehålla mer information utan att öka arbetsbelastningen. I detta fallet är mjukvaran en hemsida. En hemsida kan visa information för besökare men det blir vanligare med hemsidor som samverkar med besökaren. Det kan vara att handla, beställa en resa eller sköta sina bankärenden.

Första steget var att undersöka verktyg som finns på marknaden redan idag. De jämfördes mot kriterier som togs fram för att göra utforskande testning bättre samt för att täcka upp några av de svagare sidorna. Slutsatsen blev att det idag inte finns några verktyg som uppfyller kriterierna, det som saknades var möjligheten att automatiskt spara in- och utdata från systemet. För att kunna lagra in- och utdata byggdes ett Firefox tillägg som skulle kunna användas vid testning. Verktyget byggdes så att testaren ska kunna lägga fokus på systemet, kunskap och testning.

Verktyget som byggdes är en prototyp för att kunna testa konceptet med att spara in- och utdata under utforskande testning. Arbetsnamnet för prototypen är ETnote.

Nyckelord: Utforskande Testning, verktyg, webbsidor, Firefox add-on

Foreword

First and most a thank you to my girlfriend Jeanette for reading this thesis.
I also want to thank Emile Engström for her comments and to Christin Lindholm who has been the supervisor for this thesis.
Thank you Gustaf, Martina, Sophie and Håkan for answering my questions and giving ideas for the work and Marie for the help with contacts.

List of contents

1 Introduction....1
1.1 Can a tool help?....1
1.2 Limitations....2
1.3 Related work....2
2 Methods....3
2.1 Interviews....3
2.2 Survey....4
2.3 Implementation of a tool....5
3 Background....5
3.1 Introduction to Exploratory Testing....6
3.2 Exploratory Testing; used by many, mastered by few....6
3.3 Different approaches, not necessary Exploratory Testing....7
3.3.1Ad-Hoc....7
3.3.2Session based [6]....7
3.3.3Other approach to exploratory testing....8
3.4 Benefits....8
3.5 Drawbacks....9
3.6 Notes from Exploratory testing....10
4 Tools....11
4.1 Why using a tool....11
4.1.1 Requirements for a tool....11
4.2 Different options....13
4.3 The tools....13
4.3.1 Key logger....13
4.3.2 Wink....13
4.3.3 Rapid Reporter....14
4.3.4 Session Tester....15
4.3.5 Selenium....15
4.3.6 Tools and other options that was rejected....16
4.3.6.1 The key logger....16
4.3.6.2 All the tools that cost money....16
4.3.6.3 James Bach's own developed tool....17
4.4 Testing off the tools....17
4.4.1 Summary list containing the results after testing the tools....17
4.5 The tool created....18
4.5.1 The goal with the tool....18
4.5.2 How to build a tool, solution....19
4.5.3 Why a prototype?....22

4.5.3.1 Firefox and add-ons.....22
4.5.3.2 Inputs to a browser.....23
4.5.4 Etnote during implementation.....23
4.5.4.1 How to access ETNote from the browser.....25
4.5.4.2 The output file.....26
4.5.5 Improvements for ETnote.....28
Short version for improvements.....29
5 Conclusions.....29
5.1 In the future.....32
6 References.....33
A Appendix – JavaScript file.....36

1 Introduction

A web-page is today an important source of information to new and old customers. Most companies have a web-page and they get more and more complicated. The customer could for example buy or sell stocks, transfer money or search for a product or information. Many times the web-page is the first contact the customer has with a company.

When a web-page gets more functions and becomes a bigger part of the business, more money is spent. The code is more complex and the system in the background supports more functions than before and over all it gets more complicated. When the system gets more complex and expensive the need for testing increase.

Exploratory testing [1] is based on the idea that testing can be done more effectively. When it comes to finding important bugs testing takes less time using ET compared to scripted test cases (writing test cases takes a long time). During exploratory testing, the notes are created by the tester. They are the outcome from the test session and shows the test steps, test data and web-pages visited. With ET it is the way to make small variations in the testing, for example to take a different way or use a different product, that is the benefit of using this test method. Defects that otherwise wouldn't be discovered can with ET quite easily be found. To log the variations in the testing, notes are taken by the tester but it can be hard to log everything. It is not easy to know if something turns out to be important later.

1.1 Can a tool help?

To get help with the testing, tools are available on the market. There are both expensive tools from large companies but also free open source tools. The goal with the thesis is to find a tool that can help the tester and the test manager to collect more and better information about the test steps and test data with ET. A tool (ETnote) will also be implemented to see if a tool that can automatically collect test steps and test data would be to any help.

The thesis will try to answer:

“Can a tool show what a tester has been testing?”

First the thesis will discuss ET, the benefits and drawbacks, and then shortly it will discuss the interviews. After that a few tools, and the one that was implemented, will be tested from specific criteria. The report will finish with conclusions and future work.

1.2 Limitations

This thesis will not look into the matter if and how a tool can be used in large IT projects. The tools that was tested and ETnote that was implemented are to help a tester to create good notes. ETnote and the tested tools have been tested during a short time frame, not during a long period of time, which means the long benefits or shortcomings have not been investigated.

The tools that was tested are all free and open source. Tools that cost money are not included because:

- a tool that cost money does not need to be better
- these tools has many functions but just one or two are suitable for this project and ET. To pay a large amount of money to help the tester with notes are not always feasible.

1.3 Related work

Books, papers, articles and seminaries can be found that covers all parts off ET, how to test and how to manage ET as a test manager, for large and small projects. In literature it is easy to find benefits with ET but evidence for the claimed benefits and drawbacks are harder to find. "*Session-Based Test Management*" [6] is a great start to ET when used in projects. The paper explains how ET is managed in a project, it uses a technique called "Session based".

If one is looking for an introduction to ET in a smaller scale and the ideas behind ET, one can read "*Exploratory Testing Explained*" [2]. Both "*Session-Based Test Management*" and "*Exploratory Testing Explained*" are written by people that have influenced ET and the articles are more of a guide book then a research paper, everything about ET comes across as good and nothing bad.

Michael Bolton's article "*An Exploratory Tester's Notebook"* [11] has done some great work with explaining how notes are taken during ET. This article was used as base when looking into the difficulties and benefits with notes and ET.

The paper *"Exploratory Testing: A Multiple Case Study"* [7] has done interviews with testers and test managers that are using ET. It describes how they use ET and which the benefits and drawbacks are. The authors had some difficulties in finding drawbacks in literature. During the interviews in the paper they found problems with how to show test converge.

"Defect Detection Efficiency: Test Case Based vs. Exploratory Testing" [18] compares a number of defects and describes how severe they were when they were found. In this article "Test case based testing" is compared to "Exploratory Testing". They used 79 students to test an open source software. The students were not experienced testers so the value of the result in the paper can be questioned. The paper found that more research needed to be done on "manual testing".

What kind of people makes good exploratory testers? "*An Empirical Evaluation of the Influence of Human*" [8] is trying to answer that question. The authors behind this paper have done a study on intelligence and what kind of personality type makes a good exploratory tester. The paper has not done any in-deep interviews or observations during testing, it just uses questions and tests as basis for their findings.

The book, *Guide to the Software Engineering Body of Knowledge,* shows and explains different test techniques. A small part of the book is about ET but it is unfortunately missing the point with this test method (see next paper). The book shows how ET is being seen by parts of the testing community and what the difference is between add-hoc and ET. *"Exploring Exploratory Testing"* [19] discussed the book just mentioned and this paper does not have the same view as the book, the point is that ET is based on knowledge and not random clicking.

2 Methods

In this chapter the different methods used will be short descried. A number of different methods was used during this thesis to get a broad knowledge base.

2.1 Interviews

To investigate drawbacks and benefits with exploratory testing (ET), test procedures and work routines interviews was conducted. Interviews has been done with members with different roles in a project, one testers, two test manager and one project manager.
The questions was formulated different compared to the role they have in a project, the same questions was not asked to a project manager and a tester. This was done because a project manager and a tester do not have the same role. The interviews was done during 30 minutes slots, person to person. They where some headlines during the interviews, areas to be covered with open questions. With more open questions new areas can be disused that was not part of the interview from the start, a more interactive conversing was possible. Even that the questions was different the area regarding the questions was the same, the fooling areas was covered:

ET overall when it was used and how

The project manager did not have any specific opinions when or how ET are best used. As long it gives the same benefits as test cases when it comes to test coverage.

The test manager (A and B) liked to mix test cases and ET. The had different exampled, A used ET to test a outsourced project deliver. This was because the delivery comes complete with test cases that has been run and passed. A uses ET to find defects that test cases normally misses, e.g. error messages and unexpected behaver from the user. B have in previous projects used ET as a complement to test cases. Time have every week been

dedicated to ET. It was done for two different reasons, the testers find it fun and to find unexpected defects. Both A and B use ET as support to test cases and to find the unexpected defects.

The tester has been testing with ET for several years. He uses ET as a compliment to scripted test cases. Hi founds ET suitable when preforming GUI test, regression test, smoke test and during functions test. His office is a open landscapes office and during testing it is easy to get disturbed, work colleges and questions can lower the concentrate.

Notes from ET

The tester normally add his notes after a test session and not during. He finds it better to do after testing because during the test session he wants to concentrate on the software he tests. He normally has short test sessions and can remember the test steps afterwards. The test managers using the notes after a test session in a smiler way, the test managers look for defects, “on track” vs “off track”* and test coverage. The test managers also use the notes for test ideas for further testing. One off the test manager also raised the point that it is importunate that all test steps and test data are in the notes, the notes are the test case.

* When testing with ET a target can be set for knowing the purpose with the test case, this are then used for show test coverage. When “on track” are used it means that the testing are according to the target, with “off track” means that the tester have left the target and start to test something else. Just because the tester has gone “off track” do not mean he go back to “on track” (in the same session).

Can a tool help?

When asking the tester and test managers if they can find a use off a tool that can help the testers during ET the answers was not to positive. The tester could see a use for a tool that can save the URL but are not interested in a “glorified notepad”. The test manager did not see a use for ET tool.

Non of the persons that was interviewed use any tools for support for ET. They did not neither know of any tools that can save inputs or help the tester with the notes.

The fact that the persons that was interview did not seen a benefit with a test tool that can help the tester with notes was taken under consideration. The destinations was still that a tool that can help the tester with the notes may give a benefit needed to be investigated. This decision was based on the on a couple off different things.

- if not anyone had used it, how can anybody know it gives any benefits?
- testers that add notes after the test session may missing something
- test manager wants everything or large part of the test session documented, if a tool can help with that it would be a benefit

2.2 Survey

This thesis has done a survey that can be divided up in to two, the first one was to find criteria for the tools tested and the second was during testing of the tools. During the first survey where the focus on finding suitable criteria for a tool. This was done with the help of interviews, literature study and search the web for views and opinions on ET. The opinions was regarding strong and weak parts when using ET. From the weak part a number of areas was located that can be help with a tool. It was also important to find criteria that can help the tester with ET. Criteria was also set upon the usability for the tool, if it was easy to get started with and if it was easy to work with, it had to give a plus value for the tester. A couple of the criteria was more important then other, **free** and **easy to maintain.** They where set so companies and a tester can start to investigate the tools without any problems, it has to be easy to get started.

The second part of the survey was done during the testing if the tools located on the market and on ETnote that was implemented for this thesis. The survey used the criteria as base for the testing.

2.3 Implementation of a tool

That it was necessary to implement a tool became clear after testing the tools that was located on the market today. ETnote was devolved for fulfilling the criteria that was not full-filed (mostly do to the lack off automatic collecting of test data) buy the tools tested earlier in the thesis.

The implementation was done in three different steps, the first was to try different languages, this was a part to investigate the different options for how to build a tool that can fulfil the desired functionality. The second part was to implement a basic firefox add-on, to learn the language and understand the functionality. The last step was to build a add-on that are easier to use and understand and remove code that are not useful any more (code that was used when looking in to different options and for help during implementation) The last step also included testing of the tool in small scale. The tool was used for testing functions at a web-page, same testing as the tool was intended for. The testing reviled some findings and changes was done to improve the tool.

3 Background

In the best of worlds testing would be a natural part of the development of a product but with tight deadline that is not always possible. The problem can accrue in most project modules and it is not easy to fix, it is not possible to test a product when it is not developed. Testing can be done on part of the product or background system but when testing a finished product new questions and problems will always occur.

ET can be used both for background systems, during a sprint or on the finale product. When ET are used do not really matter for this thesis, what matters is how can a tester show what they have tested.

When taking to testers and reading literature one of the more common things that are lifted as a problem with ET is that the traceability during testing are a problem. They is already work routines and tools for helping the test manager with how to show test coverage [1]. But tools for helping the tester are missing or not used.

Companies are also using outsourcing as a business model to lower the costs, this also put special requirements on the tester. Difficulties with testing a new product can many testers relate to, how to learn a new system while testing are ET. The outsourcing partner can also be late with their delivery and that increase the time pressure.

With this factors combined the tester has a heavy work ahead. If a tester can get help with a tool that can be greatly beneficial. To understand ET and why a tool can help please continue to read chapter 3.

3.1 Introduction to Exploratory Testing

“Exploratory testing is simultaneous learning, test design, and test execution.”[2]

James Bach is a person that has influenced exploratory testing maybe more than anyone else. He is also the person behind the quote above and it “has emerged as the all-round favourite” [2] definition for ET. This quote will also be used as base when describing and thinking ET in this thesis.

There is a difference between Ad-hoc and ET, where Ad-hoc is “too often synonymous with sloppy and careless work” [4] whereas ET is a more strict approach.

Exploratory testing has a long story, 27 years ago Cem Kaner [3] used the phrase “Exploratory Testing” for describing something all testers do. During the last 27 years a lot has happened and today most testers seems to not use ET but are going for Ad-Hoc instead.

The definitions of exploratory testing are many and they are a bit confusing some times, see chapter 3.4 for more details.

No matter which definition is used there are some basic facts.

- It is not script based testing
- The tester explore the software
- No test run is the same
- ET is an approach not a technique [1]
- The tester learn the software while testing

3.2 Exploratory Testing; used by many, mastered by few

Short description for James Bach quote, divided up in key words.

Learning

During testing the tester will learn how the the system works, and use that knowledge to test better and find more bugs.

Test design

When testing using ET as the approach the outcome will be notes. The notes are essential to ET, they are the test design. They will reveal test steps, data and outcome.

Test execution

During learning and design, testing will be done. The tester is doing all three things at one time, that's the idea with ET.

When testing using ET, a tester needs to use his/her brain, it's not just random clicking or typing and see what happens. It is when the tester sits down and tries to find a bug with a plan that exploratory testing works. The plan can be different, it can be:

- Based on defects found
- Based on tours, see James A. Whittaker book "Exploratory Software testing"
- Against a standard, see James Bach Article: "Exploratory Testing Explained" chapter: ET in Action.

The outcome is the same though, information about the system that consists of notes taken by the tester.

3.3 Different approaches, not necessary Exploratory Testing

In this chapter we will look in to different test approaches.

3.3.1 Ad-Hoc

Same as for ET, Ad-hoc has different explanations, but some see it as a more sloppy way of exploratory testing [4]. The large difference is that Ad-Hoc is not based on creating test cases during testing. It can also be seen as "ET is Ad-Hoc but not all Ad-hoc testing is ET" Ad-hoc may be the most used test technique today [5].

3.3.2 Session based [6]

A test technique created by the brothers James and Jonathan Bach.

When testing pre-scripted, companies can count the number of scripts executed. On a company level this has large benefits, something ET is missing. With "Session based" the testing is divided in time slots (instead of scripts). This makes the company see the time

spent on testing compared to other things, for example meetings, training and so on. It will also give the company a good overview of what have been tested, if the tester has started to test or if they are waiting for a product and costing money while waiting.

For every test session a target was decided, a session can be seen as a mission with a specific task. During testing the tester also needs to see how much of the testing was on target and time spent on testing other things. This is to track the time spent on the specific task, adding bugs, notes or testing something that is interesting.

A test session is about 90 minutes long, give or take, and the tester is not supposed to be interrupted. No phone, mail or breaks, was the goal but of course that was not always possible, but that is the goal when testing "session based".

After a session, a day or a week the tester meets with the test leader/manager and has a short debriefing how the test results look.

"Session based" is created with large test teams and companies in mind. It is used by many companies and have influenced the ET community.

3.3.3 Other approach to exploratory testing

Why not take a tour?

- The Saboteur
- The All-Nighter tour
- The supermodel tour
- The back alley tour

James A. Whittaker is a writer and a tester who has written about the "tours" in "Exploratory, Software Testing". It is a way of thinking when testing software and the different tours is how to test. "The Saboteur " is all about sabotaging, disconnecting the internet connection or close the supporting programs during the test.

"The back alley tour" looks deep into the system and test things that are not the first thing a user see. They "The All-Nighter tour" his how long can you stay out? When will the application crash do to "out of memory". How many evenings hot-spots (bars) before something goes wrong?

"The supermodel tour" are about how do the application look and feel. Like a fashion show, it is all about the "surface".

3.4 Benefits

When looking for benefits of exploratory testing the thesis concentrates on things that are applicable for this case.

To find scientific benefits with ET is harder than it sounds. The benefits that are given with ET without a paper or report to back it up are:

- Founds more defects faster
- Possible to provide fast test results
- The tester use his knowledge
- The tester have more fun
- Possible to test without a large requirement mass.
- Possible to do more testing with less preparations

The opposite of ET is scripted test, so many of the benefits with ET stand in relation to the draw backs to scripted testing. A common scenario is that it takes less preparation to start with ET, but this is not always true. ET can be used for most testing, so listing all benefits is almost impossible but the ones this thesis and case will use as a ground for testing is that it enables the tester to find more defects faster. This benefit is mostly backed up by personal anecdotes from the community in ET.

A case study [7] with interviews showed that many of the things stated by the ET community have support, at least in this study. It also showed that:

- It is “difficult, laborious, and even impossible” to cover everything with scripted tests
- Works well when testing from a user view
- ET relays on good testers

The last point is that ET relies on the tester and this is not the only thesis that has this point, the tester needs to know what he/she is doing. A paper [8] investigated what makes a good exploratory tester and got to the conclusion that it was the tester’s personality. It found that a good tester needs to have an “extrovert personality” [8].

3.5 Drawbacks

All drawbacks have been discussed in other works and this thesis will not go in to the deep with answers and will instead try to find a tool and work routines that may help to overcome some of them. And they are:

- Traceability
- Areas missed
- Document the details of testing
- Difficult to replicate a failure

Traceability

Traceability can mean two different things,

1. hard to trace a test run to a requirement
2. hard to trace what and how something have been tested

The first one can be a problem due to the fact that a test run can test several different requirements, or no requirements. This is a problem because it's hard for anyone to know what has been tested and why it has been tested. The second one will be covered in the others.

Areas missed

When testing with ET and the tester goes on tours, finds defects and explore the software there is a risk that important areas are forgotten or missed. If this happens on tested areas of the product it can be devastating.

Document the details of testing

A scripted test shows all test steps and test data that were documented before. ET works in the opposite way many times. Test data and test steps are written down during the testing and if it's not done correctly, they will probably be lost.

Difficult to replicate a failure

This is connected to the previous point, documentation during the testing. A defect that is not possible to re-create is a problem in many ways. For example:

- finding the defect can take just as much time as when found for the first time
- maybe it's in fact not a defect, but it will take time trying to verify that it's not
- a defect not found again is a defect that will be found later and cost more to fix
- a defect can occur from steps and settings that are forgotten or done without intent.

These are some of the of problems when defects are not documented correctly.

3.6 Notes from Exploratory testing

During ET notes are the most important factor, and many times the only result from the testing. The notes will have all the important information [9] such as:

- the outcome
- defects
- notes
- issues
- areas for more testing and test ideas.
- test data

- settings

It is impossible to remember all these things a week later, or the next hour during a busy day. So notes needs to be used. The notes are also the information given to the test leader / manager.

But does everything need to be documented? Well, everything that is important.

How will I know what is important? That is hard to know before you need it. Experience is one thing that will help the tester to know what is important. But even the best tester can miss things and things that don't seem important, may have an effect on the test result.

The results from scripted test cases are a way of showing test coverage, areas and which requirements that have been tested. When using scripted test cases it's easier to understand if the test case has "passed" and it works, or if it "failed" because it has one or many defects. When comparing ET to scripted test cases this will always be seen as a drawback.

To get at better overview and structure a standard template [10] can be used for the test notes that are for reporting. For drawing, a paper is probably better, due to the fact that drawing pictures in a computer is always more time consuming. This thesis will look into tools that can help with the notes for reporting.

If you want to learn more about ET and how to take good notes please see Michael Bolton power point and presentation: "An Exploratory Tester's Notebook" [11].

4 Tools

One of the drawbacks with exploratory testing can be formulated into this question: "How can I show what I have been testing?". Can a test manager after some time see what has been tested, how it was tested and what the outcome was? The answer is yes he can, if everything has been correctly done by the tester. The tester manager also has a large responsibility with training and providing information about what needs to be done to the testers.

It can be a hard balance between efficiency and thoroughness when taking notes. It needs to be done in a way that explains the testing but do not steal time from the testing. It is also easy to forget to add small changes in settings that can make a large impact for the test result.

Today most testers rely on notepad, the classic that comes with windows, or even more primitive, pen and paper. (Pen and paper has its advantage, the tester can draw to explain his steps or system better and if the system crash nothing get lost)

4.1 Why using a tool

A tool can help the tester to catch the small difference regarding both settings and variation during the testing itself. It can give the tester, the developer and the test manager

a better understanding of the testing and the outcome. In the long run using a tool can give a better overview when defects are found, missed and why.

The tool is not supposed to replace the note taking. It is still important to write down things that need to be investigated more, ideas, deviations and so on. A tool cannot replace the tester's brain but it can help with logging and repetitive note takings.

4.1.1 Requirements for a tool

The requirements are collected from interviews and from other literature, see section 3.5. The tool will hopefully decrease or remove draw backs with exploratory testing.

Requirement	**Rank**
Free	Required
Low maintenance	High
Easy to use	High
Save the URL	Medium
Not a glorified notepad	Medium
Possible to run the test case at a later time	Low
Save inputs from keyboard (Key logger)	Medium
Screen dump	Low
Support the most commonly used browsers	High

A closer view for each requirement

- **Open source / Free** – A tool is not necessarily free because it is open source. The companies require that the tool is free. They are not willing to spend money before they know values will be generated – money or quality.
- **Low maintenance** – The tool will be used by a small test team and to maintain a tool can cost both money and time.
- **Easy to use** – if a tool is not easy to use, it won't be used. It needs to give the tester an extra value.
- **Save the URL** – a URL can say a lot about the testing. It will let the user know about the environment that is tested, links clicked and the web-pages visited. It can be easy to see the test flow. An auto save function or just a click away can be helpful.
- **Be more then a notepad** – if the tester needs a notepad, he will use it. The tool needs to provide something extra.
- **Possible to run the test case at a later time** – this is a great feature but it requires a larger program.

It is not the main purpose with an acceptance test, to be run at a later time, the main purpose is to control the delivery.

- **Save inputs from keyboard (Key logger)** – small changes to input can create something completely different in the end. Defects can turn up in the most interesting situations and to recreate can be difficult.
  Another benefit is the possibility to get statistics regarding words or characters often used, so the tester can start using other ones.
  One problem with this is the possibility to end up with large amounts of data that is hard to use and/or to get an overview over.
- **Screen dumps** – a picture say more than a thousand words. But how often do we want one taken? There are some great tools on the market already which is why this in comparison seems somewhat unnecessary.
- **Support the most commonly used browsers*** – testing will be done in different browsers and operating system. The most commonly used browsers are Explorer and Firefox which means that this requirement will be ranked higher if the tool supports both. If it doesn't support Apple products it will not effect the ranking much, testing and most external users are using windows environments [12].

*”most commonly used” means browsers that are used most often by the users for the homepage. This is always changing but it is around seven (7) different browses versions. [16]

4.2 The tools

The selection of tools was based on how close to the requirements they were. The tools also have difference between them, one is close to a notepad, one is more a screen dump tool and some are a bit of a mix. “Rapid Reporter” and “Session Tester” are influenced by James Bach test approach [10].

4.2.1 Key logger

A key logger could help to capture all things a tester does during a test session. A key logger monitor the keyboard and the mouse to see what is done, which programmes that are opened, the text that's added and links that are clicked on. This thesis works with the “mindset” that everything can be saved during a test session. Everything can go through, be evaluated and decisions can be made based on that information.

The information sampled by the key logger can be used to find out if the testing has been done with the JavaScript turned off temporarily. If that is a fact, the testing needs to be redone with JavaScript turned on.

The key logger logs everything during a test session and could therefore create a very large amount of information. The question is whether it's possible to control the data and to be able to make any sort of decision from it. The answer, without having a scientific ground, is: No. The information will simply be too much.

Key loggers are often run in the background, in many cases as spy-ware. This will create another problem, which is how to control what the key logger captures. This will make most companies nervous, not just legal, but how will the company know what is logged and not logged?

A key logger has many draw backs but the basic idea is good. It helps with the problems to document the details of testing and with the difficulty to replicate a failure.

4.2.2 Wink

Winks main feature is the possibility to take screen shots during the whole test session. The user can decide how often the screen shot will be taken, if the screen shot will be trigged on events like a mouse click or from the keyboard. The tester can choose to save all pictures or just the ones that is important and everything can be reviewed after the test session.

If design features are tested, winks can be extra helpful to see how the product looked like some time ago and then compare that to a more recent version. During design testing the tester can also go back to see if he/she had missed any old defects, or if they are new.

When testing wink's function, it didn't work trying to trigger the shot by a mouse click or the keyboard (tested on a computer with windows 7 as operating system). This takes a lot of the positive things away with the tool and means that the tester ends up with a large number of pictures which could be hard to handle. Testing during 30 minutes can generate three hundred screen dumps. This is a large amount, both to go through and to save (large data quantities). Wink is also missing the notepad function which means the tester needs to use a separate tool for notes, not very efficient.

4.2.3 Rapid Reporter

This is a light tool, no need for installing or set-ups. It's just to download the tool and to start the program. The tool occupies a small part of the screen and is always at the top (it's placed above other opened programs). It is easy to access during the whole testing. It has two buttons, one for screen dumps and one for taking notes. When adding notes a new window opens that consists of a simple notepad with the only extra function that makes it possible to style the text a bit. Before testing can start the user needs to fill in his name and the name for the session. Here it's possible to “Clear text” and “Save and hide”. The buttons are located side by side and if the tester is too quick with the mouse it's easy to erase instead of saving. Although a “crlt-z” makes the text comes back.

Every time a text is saved, a new file is created. If saving multiple times the tester ends up with a new file for every time saving, meaning too many files in the end. The tool also auto saves the files without creating a new file, but to rely on that function would in many opinions be foolish considering how often soft- and hardware breaks.

The files are saved in the same folder as the program is saved. This means the user needs to move the files from the folder where the program is stored. A more easy way is actually

to create the folder structure before and then move the program to the right folder. With that procedure all screen dumps and notes gets saved in the correct folder from start. The program generates two files and the screen dumps.

The notes during a test session are stored in one file. The file is then stored automatically or could be saved manually, as a new file.

The screen dumps are also saved in the same folder. There is two different options here, to click on the button and the file is saved (the tester doesn't get a confirmation that it's done though) or to click on the button and at the same time press ”shift”. If this is done window's program “paint” opens and it's here possible to cut and/or mark the screen dump. This can help in a later step to identify why the screen dump was saved. Of course it's also a great way for the developer to understand the problem better.

This tool was the only one that supports both screen dumps and notes, but the support was not complete or great, it was about average for both functions.

4.2.4 Session Tester

The first impression of this tool, when starting the program, is that it's a complete design with session testing in mind. The first view is the basic setup for exploratory testing: tester, mission and length of the session. The tester can also select if he/she want reminders when the mission ends (get a “warning” when the time is running out) and/or what the purpose is for the testing.

To get a small reminder may some testers find very useful while some will never use it, but it's a nice extra feature. Another feature is the button “Prime me”. Using this, the tester can get helpful tips when it comes to exploratory testing, it's a fun function and you can always learn something new.

When the testing session starts the tester can see a notepad with a drop down. The drop down is filled with useful notes that can be inserted in the notepad during the testing, some more useful than others. “Bug”, “Issue”, “Environment” and “Area” are some of them.

All the notes from the drop down starts with an '@'. This will help when making reports. The tool has a built in function to generate html reports. The design of these reports is not the best but definitely fulfil the purpose. It is also possible for the tester with some programming skills to write a report program by him-/herself to get the information he/she is interested in.

The conclusion is that the session tester is a form of notepad with some extra functions, it does not stand out in any way but it helps the tester with the basic things.

4.2.5 Selenium

This is the wildcard within the tools tested. Selenium is an open source tool designed for creating automatic test cases. It is an add-on to Firefox in the basic form (Selenium has

more to offer than the first view, see chapter 4.3.4.1). It's easy to get started, just download the tool, start Firefox and install it. After this you open the add-on, press record and start building your test case, in this case, start with the exploratory testing.

The result is a file that can be run or viewed. If viewed the tester can see what links he pressed, what happened and so on. If the tester decides to run the test (which is the primary purpose of Selenium), Firefox will run the test set again and present a result. These test cases are also great to run as regression testing when needed.

The tester has now created a test case that can be run when needed and has a complete log containing everything he/she tested. This sounds perfect but it has some draw backs.

When testing the product, the design is important. To catch design defects with Selenium controls needs to be added to the test case. This is done by coding, so to get a complete regression test hours need to be spent on programming, not testing.

The file that is generated from a recorded test is not in the most reader friendly format, it requires trained eyes to know what has been tested. Selenium is also missing the notepad functions, which means the tester has to have one more program open for notes. After a test session the tester and the manager have one thing, an exact test case consisting of what he/she did that can be recorded.

Selenium add-on is for Firefox, no other browsers are supported. This is a clear disadvantage.

4.2.6 Tools and other options that was rejected

Other options was sometimes excluded from the report early, some was investigated carefully. Here are some of the options for tools that was excluded, early and/or late.

4.2.6.1 The key logger

This is not a safe option. Many companies will feel uncomfortable with installing a key logger downloaded from a random site. See chapter 4.3.1

4.2.6.2 All the tools that cost money

It's possible to find many good tools on the market that can help with exploratory testing. In the last year, more and more companies have started to use exploratory testing which means that a large company can start to develop and sell large, expensive and good tools. Microsoft, for example, has included it in the latest visual studios 2010. If the user wants it, he/she needs to pay thousands of Swedish crowns (the price is always changing and with licence costs and different offers it's hard to say what it will cost a company exactly). To run the visual studies, the test manager also requires a “team foundation” server, one more program that needs to run the test program.

The set up is complicated and it takes several hours to get the program up and running. When it's open and running correctly there is plenty of features to help with the testing,

but it takes hours before the tester is self-going with the program. Before any testing can start, days have been used to set up and to learn the program.

Another company that has been developing test tools for years is HP. They have released a new version of their program Quality Center (Quality Center 11). Between 9.2 and 10 it has just been small changes, but 11 introduces many new features.

Quality Center 10 has support for testing, creating and managing test cases. It gives support for defect handling and requirements that can be connected to a test case, with this feature the project knows which requirements that are fulfilled. It also has support for more “advanced” testing and it has automatic testing and performance testing. In many ways it's very similar to Microsoft Test Manager 2010.

Some testers are using Qualtiy Center during exploratory testing because the test case is created during the actual testing. This way the tester ends up with a test case and the test results, this could work and the thesis will look into this option later.

With Quality Center 11 (the latest version) exploratory testing functions are standard (same as in the latest Test Manager). The tester does the testing and the program builds the test case. It's also possible to build test cases from it, run in multiple browsers in the same time and get the result directly. The design is new and it looks good, it has all the functions a tester wants, needs and more that will never be used.

Quality Center and team foundation can be very good for a large company with many testers in different projects. But for this thesis they have not been suitable, largely due to the fact that they're not fulfilling the most important requirement: the software needs to be free. It's also clear that these tools have been developed with large companies in mind, size and price.

If this thesis was dedicated to find a tool for a large test team, more time would have been spent on testing and exploring tools that cost money. To invest the money a tool costs the companies need to know it will give a return. Large companies can also have complicated routines for signing a new software partner.

4.2.6.3 James Bach's own developed tool.

This tool is more suitable for managing big projects. It makes it easy to keep track on large amounts of test cases generated by exploratory testing.

This tool was not chosen to be tested in this thesis because of the way to fill out test reports. The reports are templates and for that purpose, a simple notepad could be used. The tool will therefore not give any extra value to one single tester or a small team.

4.3 Testing off the tools

The tools were tested during an acceptance test. During the first test sessions the product had too many errors; most of the test time had to be spent adding new defects. It was

decided to wait with the exploratory testing, until a more stable environment was achieved and less defects was found, this to get a better flow when testing.

The testing is not based on how many defects a tool finds. It's rather based on the requirements and the general feeling of the tool, if it's easy to work with, benefits and draw backs.

4.3.1 Summary list containing the results after testing the tools

Requirement	**Selenium**	**Rapid Report**	**Session Tester**	**Wink**
Free	Yes	Yes	Yes	Yes
Low maintenance	Medium	High	High	High
Easy to use	Low	Medium	High	Low
Save the URL	Medium	Low	Low	Low
Not a glorified notepad	High	Medium	Low	High
Possible to run the test case in a later time	High	Low	Low	Low
Save inputs from keyboard (Key logger)	Medium	Low	Low	Low/ Medium*
Screen dump	Low	High	Low	High
Support the most used browsers	Low	High	High	High

*If the program catches a screen shot in the right moment.

- “High” is the highest grade, it fulfils the requirement
- A “medium” means that the requirement can be attained with some work from the tester.
- “Low” means that the tool has little or no support.

4.4 The tool created

After researching the tools that can be found on the market today none of the tools was found to meet all the requirements. The area that was lacking most, according to the research, was the possibility to log or save the test steps and in-data to the system.

4.4.1 The goal with the tool

When starting the development, the goal was to build a tool that could:

- log
- collect in-data
- show test steps

But most importantly, it was to be automatic.

The basic idea was to save all inputs that the tester gives the system. When that's done, a large amount of data will be collected so it's also important that it's possible to select the type of input the tool collects.

The large benefit when this is done automatically is that all information is saved, nothing is missed. It also creates the opportunity for the tester to concentrate on the system instead of how to document the testing.

The goal was also to create a tool that is easy to use and where the tester has the control over easy-to-use-settings. The tool needs to be fast and when the testing is started, the tool needs to run in the background without disturbing. It was also required to build a program that companies can use, which means no spyware or key loggers. The program will just collect information that is meant for the system that is tested.

During the research it was found that no open source program on the market had this features. To be able to see if the program can meet the goals a prototype was implemented.

4.4.2 How to build a tool, solution

After the first investigations, there were two different options that could resolve the objectives:

1. building a version for every common used operating system, complex language
2. building a version for every common used browser, scripted language

Short explanation to complex and scripted

This two names are just valid for this thesis, they should not to be seen as a true.

This names is to categorise the different solutions investigated.

In this thesis a complex language can be seen as a high-level language or a language that need to be complied to work. It can also be seen as a language that have more built in functions for implementing large system and with graphic support (e.g. Java, .NET with C#). It is not the same as the language that just work on a high level without access to registers in the operating system.

A scripted language are seen as (in this thesis) smaller language, not in the number of users but more as what it can be used for and functions. It do not need a complied to work

(e.g. javaScript) and can run direct in a browser as a add-on. This is not the same as a “scripted language” are less difficult to master then a “complex language”.

Complex language

When building a tool with a complex language that can be used for operating systems without depending on what browser the system uses, a programming language needs to be used that can access the register in the computer. The support varies and here are the ones investigated:

- **JAVA** is a program language that supports most operating systems because it's run on a Virtual Machine. The problem with JAVA is that it has a limited low-level access to the operating system, which makes it hard to access registers. It needs to be installed on the computer but it's commonly used (1.1 billion desktops have it) [15].
- **.NET** is a framework from Microsoft. It's well integrated with the operating system Windows but it has a couple of problems. The computer needs to have .NET installed in order for the program to work; same as JAVA but they both need the correct version (if any specific functions are used). The benefit is that it has easy access to all registers on the windows machine.

Even when building a tool specifically for windows, changes need to be done in order for it to fit different versions e.g. windows 7 and XP requires different code.

The tool is not supposed to be a key logger, which means it will still require different code for the different browsers in some way. Chrome and Internet Explorer don’t use the same part of the register, so the tool first needs to know which type of browsers that are tested if they're going to be able to pick up the correct inputs.

The problem with a low-level solution is that it makes it harder to pick up elements on the browser that haven’t been an input from the tester, for example if a button is located on the browser but was never used.

A complex solution has the benefit that it can pick up mouse and keyboard inputs, which a scripted solution can´t (this is supported more or less in different versions).

Scripted solution

With a scripted solution, a different tool needed to be written for each browser. The most used browsers are Chrome, Firefox and Explorer [16]* and all have support for add-ons (which is a “selling point” for the companies that build browsers). Some code can also be reused.

A browser knows all the elements in the browser, also the ones that are in use by the tester. It knows internal id, the value and elements presented for the user and all inputs. The only thing it doesn't know is how the inputs are given, is it an “enter” or a mouse click?

The browser can also give too much information, a web-page can have loads of different elements so selecting what elements to save and what is important will be a challenge.

One problem with a browser is that the information is just collected when the web-page reload. This means that if the web-page doesn't reload according to the browser it will not read any information or what happens between the loading of the pages.

*Statistics for browsers give always different results but this three are always the top three.

Decision time

The first step was to start developing and to see how much support there is for the different goals for the target that was stated. Before starting developing a quick search on internet was done to see what can be done with different languages. But without working with the solutions it is hard to get a good understanding of how big or good the support is in the different languages. The decision was made to try a couple of different languages and then to make the final decision based on that.

The developing started with using a complex language.

- **Java** runs an virtual machine (JVM). It has large benefit but in this case it was a drawback. To get the needed access a .dll libery togther with C++ code had to be used. The developer environment (Eclipse) was also missing support for accessing the the loxer level in the operating system. Java was not a subtitle language for this task.
- **.NET with C#** has a much better support for accessing the lower-levels in the operating system. It has also a good support in the developer tool (Visual studies) for accessing the register, and to see the value in the register. It is also easy to create a GUI where it is just drag-and-drop to put the elements right.

  But is also had some problems. It needed to have different code for different operating system. Browsers are also use different part off the registers so different code was reacquired. A benefit is that a tool develop with .NET can locate all inputs from mouse and keyboards, nothing was missed, it can log things that happens between the reloads of the pages.

The next step was to investigate the benefits and drawbacks with a scripted language.

The first step was to see it the tool can collect all the elements on a page and that was not a problem. The problem on the other hand was that it collected too much. The second

problem was when to collect elements. This was done possibly by an event handler, more about that latter.

After the same time spent on scripted language as a complex language the decision was easy. Much more testing had been done with a scripted language why this was chosen. The scripted language still has its drawbacks but overall the time factor was overwhelmingly positive.

After selecting the language, a scripted solution was the best option to build a prototype to test the concept. For the finale product the next step was to select the browser to build it in. All the large browsers have support for add-ons and the set-up is similar, the thing that made the decision easy was the support, documentation and books which all worked in the favour of Firefox.

4.4.3 Why a prototype?

For all programmes that are built, there should always be a prototype first to test both the concept of the program and the usability on (this is a personal opinion though). To test a concept the tool needs to be used as it is intended over some time and the result to be analysed. This investigation can be done on a prototype. If functions need to be added, removed or changed it's easier and cheaper (time is money) than changing an already complete product.

For this thesis it was also more suitable with a prototype, both due to reasons just discussed but also to finish in the time-frame for this thesis. The prototype got the workname: "ETnotes"

4.4.3.1 Firefox and add-ons.

A Firefox add-on is built with CSS, JavaScript and XUL.

- **XUL** is a version of XML. The syntax is the same and have the same function. The XUL holds the information on buttons, text boxes and the position. It's also in the XUL where actions from the user (e.g. Click on button) are connected to the JavaScript.
- **CCS** works the same way as HTML [14]. It's in the CCS file where all the styling happens. Colours, paddings and text style are control. If the goal is to build the best looking add-on, the CCS file (files) is the place to work in.
- **JavaScript** is where all the action happens. JavaScript is a script language that is often run in the client (the browser). If it's used together with HTML it makes a homepage look better and it interacts with the user. When used in an add-on, it could be viewed as the brain.

In the XUL file there is a button, which could be a picture or have stylings from the CCS file. When the user clicks on the button a function is triggered in the JavaScript. The function is run and displays the result for the user. JavaScript is the only part of the program that works when it's up and running.

A Firefox add-on also has a “start-up” file, a XUL file where it's possible to add extra buttons to the browser and a file where data can be stored when the browser is turned off (or if the computer is). It is possible to add toolbars and buttons on the browser or open a separate window that is connected to the browser.

The large benefit with an add-on is that it has direct access to the browser and therefore all inputs can be collected without interfering with private inputs. One drawback is that it's not possible to see where the inputs are coming from, e.g. “enter” or a mouse click. This can probably be fixed but it was not focused on during writing this thesis.

A problem with a JavaScript solution is that if the tester turns off JavaScript (in the browser), for any reason, ETnote will not work. Today most users on the internet have JavaScript and most web-pages need it turned on to work. More than 90% of the users have JavaScript enabled [16].

4.4.3.2 Inputs to a browser

Most inputs to a browser go through HTML where a type is called “input”. HTML is based on different elements with different names, that is how the browser knows what to show for the user. A button is one, it is called “button” and the browser knows it is an area where the user can press and something may happen. A text file is another element and it's called “text”. It is where the user can write something, for example a login id or search for something. All these inputs are then given to the system and a result is then presented. It could for example be a new web-page that opens up, a search result presented or the user could have logged in to a system. Often the result after an action is presented by the back-end of the system, e.g. accessing a database with login information or search results. “Select” is another type of input which can be used for drop-downs for example.

It is also possible with JavaScript to get the URL where the testing is performed and this will help with the test steps and the test environment.

Here is an example of a normal input field and select element:

<select id="colordropdown" class="texts" name="color">

<input type="text" value="text field 1" name="query" id="search">

The first word indicates the element's type and for the input it also have a “type” that indicates it's a text file.

4.4.4 Etnote during implementation

During the implementation of ETnote, the following was focused on:

- to create a tool where the user can select the type of inputs the tester wants to save
- that design doesn't matter, as long as the tool can be used
- that the tool will not effect the testing while running
- to save the output in a text file.
- to save the id of the HTML element. Not always used, but when it is it can tell the tester from where the data is collected.

When a web-page loads or reloads on the browser a EventListener can be used for catching the event. The EventListener syntax:

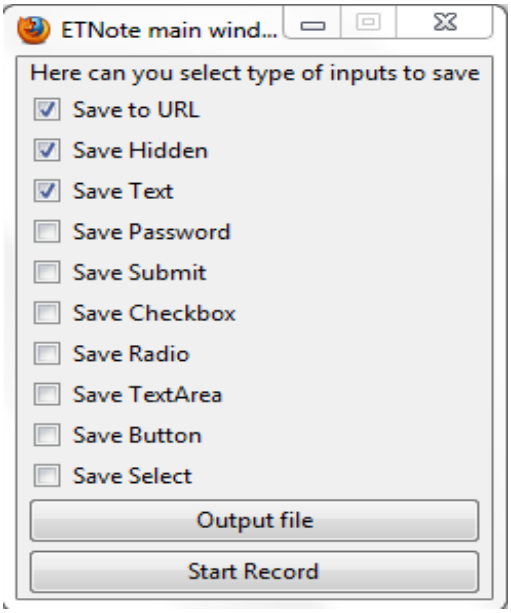
window.addEventListener("pagehide", OnPageUnload, false);

The “pagehide” is what the browser listens for and “OnPageUnload” are the javaScript functions that are runs. The “pagehide” reads from the “old”page, not the “new” page. Because it reads from the old web-page all the values from that web-page are saved, it reads indata to the system e.g. if the tester search for a product and when the web-page reloads ETnote captures the search term added in the first web-page.

For collecting the elements:
var array_Elements1 = content.document.getElementsByTagName("input");
var array_Elements2 = content.document.getElementsByTagName("select");
“input” and “select” are the types that are fetched in this example (same as in ETnote).

If a “*” is used instead, all the element types are fetched, which could be well over a hundred different ones on a commercial web-page (this includes all HTML elements).

HTML “inputs” have different names, therefore a window with check boxes was created so that the tester can select the type of data he/she wants to save. By selecting the type of data the tester wants to save he/she has complete control over the elements that are saved to the output file.



Picture 1

The different boxes in the above figure are described here:

- **Hidden:** A web-page can have a hidden values that is just visible in the source code, and not for the user. This can hold important information.
- **Text:** Small text areas, commonly used when adding a user name, address file or a search area. Used when the entry is short, one row.
- **Password:** The program picks up the passwords and show the password in the output file. The complete password is shown so if the password is private it's recommended to have this turned off.
- **Submit:** When submitting a complete form “submit” is used.
- **Check box:** This can collect the value in a check box, true or false
- **Radio:** Used with drop downs.
- **Text area:** Used for longer text inputs, for example when a user wants to send a question to a company from the homepage.
- **Button:** A button the user can click on. This may seem unnecessary because the button it self can’t store any data but it can show if the web-page had an attribute that now is missing.
- **Select:** The “select” is not an input type. It has it's own type and can be used for drop downs.

  The two buttons are used for:

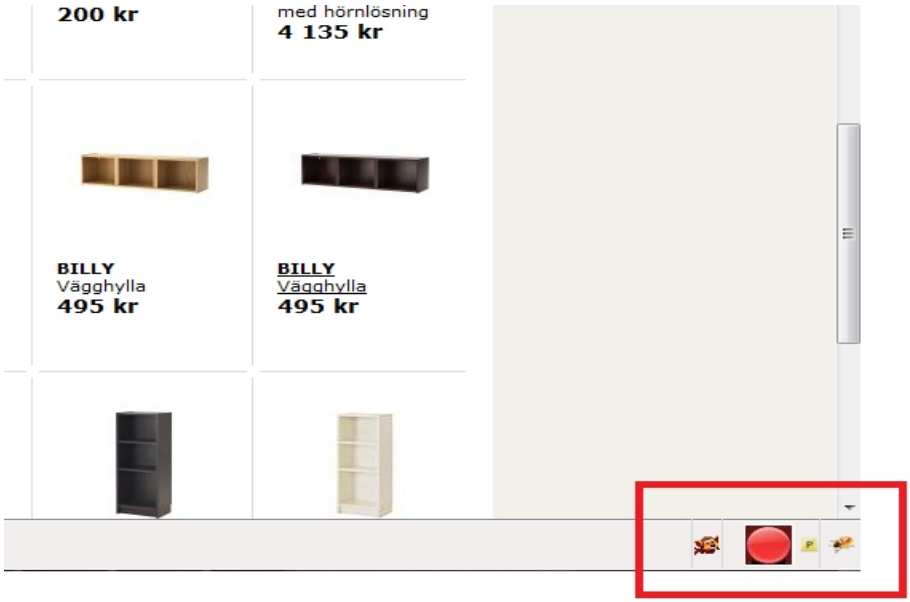
- **Output file:** Here will the user select the name and where to save the file. The name can be important because it can tell the tester what he has tested.
- **Start record:** This button starts the recording, when pressed the text on the button changes to: “Stop recording”.

The “page hide” runs when the web-page have reloaded so that settings from the box will take place directly. This means it's possible to change the values on different web-pages depending on what is required from that web-page.

The program has two different record types, one is just for the URL and the other is for HTML data collected on the browser. With this set up it's easy for the tester to test specific features or web-pages and it can also help during implementation.

4.4.4.1 How to access ETNote from the browser

The button for ETnote are located in the lower right corner, it is a small button that takes the user to the main window. Here he can select type of input to save, file to save to and start and turn off the recording (see picture 2).



Picture 2

4.4.4.2 The output file

The file generated is saved as a text file. The location and the name of the file are selected before the testing starts.

The .txt format was selected because it's the simplest format. All text editors support it; it is easy to work with. To add a new line or space between inputs is easily done with JavaScript.

It was a balance between saving too much or too little information. One way off limiting the amount of information was to select the type to be saved, e.g. check boxes.

It was also important that the information that was saved could be used later on. To make it more clearly from where and how the saved information was used in the system the id from the element is saved in the file.

An id on a web-page is an identifier for an element so other elements can use it. For the id to work, it needs to be unique for every web-page. Using this id, a tester can look on the source web-page to find out exactly which element it was. The id is not mandatory in any way, but needs to be added if any other element needs to use the information.

The id can also be declared higher “up” in a parent “div”. When the id is stored in a div, ETnote can´t find it. e.g.

<input type=”button” value=”Send”>

The type of the element is also saved. All HTML elements have a type (not as an id) and it gives the tester a quick view what the data has been used for. For example, the tester can see if it's a text field, a drop down or a text area. This will help the tester to quickly see what kind of element it is without having to go to the source code to find the element and its type.

Here is a short example for how an output file can look like: (the numbers are not from the file, it is to help any reader of this report)

1. select-one ID=ikeaStoreNumber1 020
2. text ID=search
3. button Add to shopping cart

Short explanation:

1. The “select-one” indicates that more than one option can can be selected. “ID=ikeaStoreNumber1” is the id of the specific element and “020” is the value. In this case it's an internal id for the option selected in the “select” field.
2. “text” shows it is a text field with the id “Search”. The field is empty when the web-page was reloaded, e.g. the user have not typed anything.
3. This is a button with the value “Add to shopping cart”. For a web-page this can be very important information and helpful for a tester. It doesn't have an id that is visible for ETnote but it lets the tester know that this product is available for purchase. If this changes it could indicate that something's wrong. “Add to shopping cart” is the value that is displayed for any user.

During testing the text file gets updated as soon as a web-page are reloaded. So if the computer or program crash during testing the notes will not get lost. To write to a file in javaScript a couple of things need to be done:

1. Create a stream:

foStream = Components.classes["@mozilla.org/network/file-output-stream;1"].createInstance(Components.interfaces.nsIFileOutputStream);

2. Set the correct settings, in this case we append to the end of the file

   foStream.init(file, 0x02 | 0x10 | 0, 0, 0);

3. Get ready:

   converter = Components.classes["@mozilla.org/intl/converter-output-stream;1"].createInstance(Components.interfaces.nsIConverterOutputStream);
   Setting the correct format, UTF-8:

   converter.init(foStream, "UTF-8", 0, 0);

4. Write to the file and then close (stringToAdd are a variable with a string):

   converter.writeString(stringToAdd);
   converter.close();

The text have been added in the correct format. This will be done every time a web-page reloads in the browser.

4.4.5 Improvements for ETnote

ETnote was built as a prototype so it was never the plan to make it a complete and fully working product.

The design needs to be better. At the moment it has a very simple design with check boxes and a few buttons, this could be done better. With a better design the usability may increase and help the tester to better understand the tool.

When locating an id for a HTML element and the id is located in a div instead of the element itself, ETnote can't find it. This can be fixed by looking in the parent element and going up through the elements until the id is found. This is a solution that can work but during this time frame, it was not feasible.

There are also other types of elements that can be interesting for a tester. This can be added without a large amount of work or time but ETnote is in the prototype stage and the most used input types are covered as it is now. For a tester with some programming skills they can add it themselves.

The output file is now in a .txt format. In the long term it may be better to use a XML format, specifically if a program is developed for analysing test results or comparing to old test sets.

A XML syntax can also help if developing a program that can run the test sets again automatically, which is similar to Selenium (please see chapter for tools already on the market). This can help the tester with regression testing.

A notepad function is not included in the program for the time being. With exploratory testing, taking notes is the key to a good test result. Today it needs to be done separate in a

notepad or using notepad++ [13] and the tester needs to write in the file that stores the output result (can´t be done with the notepad that is standard on windows).

If the tester opens the test file in notepad++ it will automatically update the text and the tester can add his/her notes during the test session at the same time as the file gets updated by ETnote.

When implementing the notepad function the idea is to add the information on the web-page the tester took his/her notes. This will help with organising the notes and to put the notes into the perspective to the task performed when the observation was done.

That the notepad functions that are a important part of ET are not implemented can be seen as a large part of the functionality are missing. But the purpose was not to build a notepad; it was to see if the function that the ETnote gives can help the tester and test management.

Other things that need to be added are help texts – how the ETnote work. It can be one large help text that covers all the different parts or one small help text where it is needed.

Short version for improvements

This thesis has been great to write and the outcome is the best one yet, but there are still some improvements that need to be done.

1. Complete the tool with notepad functionality
2. Usability test the tool and make the modification needed
3. Investigate in deep if the tool can give any benefits for a large company

The first point is a question of time and effort. It needs to be investigated how this notepad function will work but the implementation part is not overwhelming.

The second point requires some work and it can be done in a couple of different ways. It can be used by testers and one can simply observe how to interact with the tool, this can be done in a small scale and with testers that are familiar with exploratory testing. This can also be done with large focus groups.

The third point is the hardest one. To investigate benefits it requires projects run in a professional environment. It needs projects that are using the same kind of testing but without the tool. Interviews from both testers and management how they have been using the tool, if they have seen any outcomes that are beneficial for a company and project.

5 Conclusions

When first started working with this thesis the plan was to find a tool that can be used during ET. The criteria for the tool was set after having done interviews, read literature and from own experience. One of the more important things was how to collect information, test steps and test data. During the start of this thesis the focus was how to test an outsourced project during an acceptance test. When preforming an acceptance test on an already tested product, the focus moves from things that already have been tested to areas that may have been missed. Flows, which describes how the users move through the product, and settings, e.g. settings in the browsers or in the software (which make the software behave differently) needs to be tested during an acceptance test. Other things that needs to be tested is to see what happens if the internet connection gets cut off or if the users start to behave in a way that is not expected.

The project that was lined up for this case study required more complete testing, from start to end, and the acceptance test became regular testing. This added up to the fact that other things came into focus when looking for a tool that can help the tester during testing. The focus moved from acceptance tests to regular testing with the challenges that face a tester when using ET. The challenges are many but the focus was if a tool can help the tester to collect information both from the software and the test sessions. A number of difficulties became clear:

- **Open office landscapes.** It can be great for communication and for knowledge sharing but it can also increase stress and cause problems with concentration [17]. When a tester gets interrupted during ET, which happens all the time in open office landscapes, valuable test information can get lost, the tester forgets what he just did.
- **Changing focus.** It is always a problem but when different parts of the project are ready for testing the tester may need to change his focus to something else. A defect that previously was blocking the testing is now fixed and it has higher priority - so the tester needs to change his focus with the risk that previous test ideas and steps get lost.

These two problems (open office landscapes and changing focus) increase the value of collecting information during the test session. This will help the tester to become more efficient because he just needs to take notes on data and behaviours of the software that are not captured by the tool. Depending on what the tool captures the tester only has to write down very little in-or outputs and can focus mainly on writing down test opportunities and ideas. The problem with collecting large amounts of data is that most of that data will never be needed and to find the important and valuable information from data stored after a test session will be complicated. The overall benefit is that with less note takings, switching to pen and paper or using the computer notepad, there is more time for testing and less information about the software will get lost.

The first option was to find an already implemented tool on the market today. The criteria was set with the background that the tool was to help the tester during ET but also to make it easy for companies to get started. The selections of tools was based on:

All tools was **free** and therefore suitable for small companies but also for large companies with complex procedures for buying new software or products where contracts need to be signed. In order to be suitable the tools were also required to be **low maintenance** and **easy to install.** Testers or companies do not what to spend time or money on installing and maintaining a tool,.

Other criteria that was set was related to the testing and were:

The tool had to be **easy to use.** "Easy to use" is many times something that comes from experience, something a user feels when working with the product and in this thesis "Easy to use" will strictly be from experience. When testing and working with the tools selected for further investigation the tools had to be easy to start with (that did not require one to read a long manual), they were not to take up more time than was gained and had to give a natural flow when working.

The other criteria was more based on **functions**. The function criteria was to decrease the manual work required by the tester and to help the tester to create better notes. The functions had different values for the tester and some was seen as more important than others.

The first step was to find the tools that was going to be more carefully investigated. During this search all kinds of programmes were investigated from different areas, not all tools was made for testing. For example, tools that record desktops for training and presentations, in this case the data storage, become the problem. Testing and recording three hours per day ends up with large amount of data. The tool had to be safe for testers and companies to use e.g. key loggers can be seen as a security threat. During this selection a problem became clear, non off the tools was going match all the criteria.

After the first selection four tools (Selenium, Rapid Report, Session Tester and Wink) was selected for further investigation. Selenium is used for building automation test cases, it is free and open source. The other three tools are influenced by James Bach and his session testing. The tools had their strong and week points (see chapter 4.4) but the overall conclusion was that not any off the tools contributed to the notes, the tester still had to take all the notes and did not get any help from the programs with the manual repetitive information.

The idea that a tool can help with the notes had still not been investigated due to no program was found that could do it. The decision was to implement a Firefox add-on that can collect in- and output during ET. ETnote (the workname for the Firefox add-on that was implemented) read HTML in- and output when a web-page reloads and reads all variables the tester has added to the web-page. The output from the tool is saved in a text file at the same moment the web-page reloads.

The tool that has been implemented is something new, it can help the tester with something that was not possible before. The tester can control the input to limit the information but can still get the important things by using the checkboxes available by the tool. The outputs are saved and the tester can search and compare to earlier test cases. It is possible to go back to see the exact route taken during a test case and to see all test data and web-pages visited.

After development and using the tool on a smaller scale it was found to work as excepted. The tester does not need to add all repetitive test data and steps, it is free and easy to install. When starting the test session all that is required is to name the file that will store the data collected and with good naming together with a practical folder system in the computer it will be easy to see what has been tested and what has not. When the testing has started it is possible to select type of input for every new web-page that is visited.

Big companies that build and sell software have now started having support for ET. Both Microsoft with their “Test Manager 2010” and HP with their “Quality Center” version 11 has a great and complex support for ET. This tool has of course much more to offer than a simple add-on but it shows that there is need for help and tools when preforming ET.

When preforming ET the tester is required to analyse the system more compared to fooling a test case. In a test case the tester follows the steps, looks at the result from the system and compares it to the, from test case, excepted result. During ET it requires the tester to think before every step, to learn the system and to document. With ETnote the tester will get help with the documentation (he still needs to think).

This thesis starts with the question: “Can a tool show what a tester have been testing?”

The paper does not leave a clear "yes" or "no" to this question but it shows a couple of different things that are important. The first thing is that using tools when preforming ET is something that is not widely used or has been investigated in a large scale. It also shows that the number of tools that support ET are limited. This thesis has implemented a tool that can help, but for a clear "yes" or "no" a long term study needs to be preformed.

5.1 In the future

To test the tool and the idea from where this thesis started needs a large test effort with a couple of different projects. This needs to be done over a longer time period together with interviews on a deeper level and tracking of defects, when and how they are discovered.

Work routines need to be investigated, how the project can use the data for creating test plans per sprint, week and day. If it is possible to track how the tester has tested it will be possible to take statistics on the web-page tested. It will be possible to find areas that have not been tested that often or if it has not been tested for a long time.

When defects are found it will be possible to know the last time someone was on that specific web-page, if the tester has used that web-page or link before. It will be possible to see what was displayed on the specific web-page for example 4 weeks ago.

6 References

[1] Cem Kaner, *A Tutorial in Exploratory Testing*
http://www.kaner.com/pdfs/QAIExploring.pdf page 143 (May 2011)

[2] James Bach
Exploratory Testing Explained v.1.3 4/16/03
http://www.satisfice.com/articles/et-article.pdf page 2 (April 2011)

[3] Cem Kaner,
A Tutorial in Exploratory Testing
http://www.kaner.com/pdfs/QAIExploring.pdf page 2

[4] James Bach
Exploratory Testing Explained v.1.3 4/16/03
http://www.satisfice.com/articles/et-article.pdf page 1 (April 2011)

[5] *Guide to the Software Engineering Body of Knowledge*, Trial Version 1.00, May 2001
Chapter 5
http://www.computer.org/portal/web/swebok/html/ch5 (April 2011)

[6]Jonathan Bach
Session-Based Test Management
(first published in Software Testing and Quality Engineering magazine, 11/00)
http://www.satisfice.com/articles/sbtm.pdf (February 2011)

[7] Juha Itkonen and Kristian Rautiainen
Exploratory testing: A multiple case study
Year: 2010

[8] Lozina Shoaib, Aamer Nadeem, Aisha Akbar
An Empirical Evaluation of the Influence of Human Personality on Exploratory Software Testing

[9] *Software Testing and Quality Services*
http://staqs.com/docs/sbtm/#5 (June 2011)

[10] James Bach, *Session-Based Test Management*
http://www.satisfice.com/articles/sbtm.pdf (May 2011)

[11] Michael Bolton, DevelopSense
An Exploratory Tester's Notebook
The QAI QUEST Conference in Chicago
April 2009

[12] http://www.w3schools.com/browsers/browsers_os.asp (July 2011)
W3Schools homepage

[13] http://notepad-plus-plus.org/ code editor and Notepad replacement that supports several languages

[14] http://www.w3.org/TR/1999/REC-html401-19991224/ (July 2011)
The World Wide Web Consortium

[15] http://www.java.com/en/about/
The official site for Java

[16] http://www.w3schools.com/browsers/browsers_stats.asp
W3Schools homepage
Web Statistics and Trends and **JavaScript Statistics**

[17]Johanna Höjd & Anna Jaxeliu
Vi trivs bäst i öppna landskap, eller?
En utvärdering av förändringen från cellkontor till öppna kontorslandskap hos Skatteverket i Helsingborg

[18] Juha Itkonen, Mika V. Mäntylä and Casper Lassenius
Defect Detection Efficiency: Test Case Based vs. Exploratory Testing
Helsinki University of Technology, Software Business and Engineering Institute

[19] 2003 Andy Tinkham & Cem Kaner
Exploring Exploratory Testing
http://www.kaner.com/pdfs/ExploringExploratoryTesting.pdf 2011-08-31

A Appendix – JavaScript file

```
var linkTargetFinder = function () {


netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
var prefManager = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
return {
                init : function () {
                gBrowser.addEventListener("load", function () {
                var autoRun = prefManager.getBoolPref("extensions.linktargetfinder.autorun");
                if (autoRun) {
                                }
                }, false);
                },
                run : function () {
                mainWindow = window.QueryInterface(Components.interfaces.nsIInterfaceRequestor)
            .getInterface(Components.interfaces.nsIWebNavigation)
            .QueryInterface(Components.interfaces.nsIDocShellTreeItem)
            .rootTreeItem
            .QueryInterface(Components.interfaces.nsIInterfaceRequestor)
            .getInterface(Components.interfaces.nsIDOMWindow);
                }
}
}
//help klass for devoplemnt
function getFieldsForView (event){
                alert("Vi letar fält i browser");
                var array_Elements = content.document.getElementsByTagName("*");
                var array_Elements1 = content.document.getElementsByTagName("input");
                var array_Elements2 = content.document.getElementsByTagName("select");
                alert(array_Elements.length);
                alert(array_Elements1.length);
                alert(array_Elements2.length);
                alert(array_Elements2[0].type +" type " + array_Elements2[0].id + " id " +array_Elements2[0].value + " value
" +" array_Elements2");
                for(i=0; i<array_Elements2.length; i++)
                {
```

```
                                                alert(array_Elements[i].type +" type " + array_Elements[i].id + " id "
+array_Elements[i].value + " value " +" array_Elements");
                }
}

//Setting the checkbox value for Indata
function setRecordIndateValue()
{
                alert("setRecordIndateValue");
                netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
                var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
                var rec = prefs.getBoolPref("extensions.linktargetfinder.record_Indata");
                if(rec==true){
                                prefs.setBoolPref("extensions.linktargetfinder.record_Indata", false);
                }else{
                                prefs.setBoolPref("extensions.linktargetfinder.record_Indata", true);
                }
}

//Setting the checkbox value for URL
function setRecordURLValue()
{
                netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
                var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
                var rec = prefs.getBoolPref("extensions.linktargetfinder.record_URL");
                if(rec==true){
                                prefs.setBoolPref("extensions.linktargetfinder.record_URL", false);
                }else{
                                prefs.setBoolPref("extensions.linktargetfinder.record_URL", true);
                }
}

function set_input_text()
{
                netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
                var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
                var rec = prefs.getBoolPref("extensions.linktargetfinder.input_text");
                if(rec==true){
```

```
                prefs.setBoolPref("extensions.linktargetfinder.input_text", false);
        }else{
                prefs.setBoolPref("extensions.linktargetfinder.input_text", true);
        }
}

function set_input_hidden()
{
        netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
        var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
        var rec = prefs.getBoolPref("extensions.linktargetfinder.input_hidden");
        if(rec==true){
                prefs.setBoolPref("extensions.linktargetfinder.input_hidden", false);
        }else{
                prefs.setBoolPref("extensions.linktargetfinder.input_hidden", true);
        }
}

function set_input_checkbox()
{
        netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
        var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
        var rec = prefs.getBoolPref("extensions.linktargetfinder.input_checkbox");
        if(rec==true){
                prefs.setBoolPref("extensions.linktargetfinder.input_checkbox", false);
        }else{
                prefs.setBoolPref("extensions.linktargetfinder.input_checkbox", true);
        }
}

function set_input_password()
{
        netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
        var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
        var rec = prefs.getBoolPref("extensions.linktargetfinder.input_password");
        if(rec==true){
                prefs.setBoolPref("extensions.linktargetfinder.input_password", false);
        }else{
```

```
                        prefs.setBoolPref("extensions.linktargetfinder.input_password", true);
        }
}

function set_input_submit()
{
        netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
        var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
        var rec = prefs.getBoolPref("extensions.linktargetfinder.input_submit");
        if(rec==true){
                        prefs.setBoolPref("extensions.linktargetfinder.input_submit", false);
        }else{
                        prefs.setBoolPref("extensions.linktargetfinder.input_submit", true);
        }
}

function set_input_radio()
{
        netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
        var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
        var rec = prefs.getBoolPref("extensions.linktargetfinder.input_radio");
        if(rec==true){
                        prefs.setBoolPref("extensions.linktargetfinder.input_radio", false);
        }else{
                        prefs.setBoolPref("extensions.linktargetfinder.input_radio", true);
        }
}

function set_input_button()
{
        netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
        var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);
        var rec = prefs.getBoolPref("extensions.linktargetfinder.input_button");
        if(rec==true){
                        prefs.setBoolPref("extensions.linktargetfinder.input_button", false);
        }else{
                        prefs.setBoolPref("extensions.linktargetfinder.input_button", true);
        }
```

```
}

function set_input_textarea()
{
		netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
		var prefs = Components.classes["@mozilla.org/preferences-service;1"].getService(Components.interfaces.nsIPrefBranch);
		var rec = prefs.getBoolPref("extensions.linktargetfinder.input_textarea");
		if(rec==true){
			prefs.setBoolPref("extensions.linktargetfinder.input_textarea", false);
		}else{
			prefs.setBoolPref("extensions.linktargetfinder.input_textarea", true);
		}
}

function set_select()
{
		netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
		var prefs = Components.classes["@mozilla.org/preferences-service;1"].getService(Components.interfaces.nsIPrefBranch);
		var rec = prefs.getBoolPref("extensions.linktargetfinder.select");
		if(rec==true){
			prefs.setBoolPref("extensions.linktargetfinder.select", false);
		}else{
			prefs.setBoolPref("extensions.linktargetfinder.select", true);
		}
}

function openWindow(event){
netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
var prefs = Components.classes["@mozilla.org/preferences-service;1"].getService(Components.interfaces.nsIPrefBranch);

		var ww = Components.classes["@mozilla.org/embedcomp/window-watcher;1"].getService(Components.interfaces.nsIWindowWatcher);
		var win = ww.openWindow(null, "chrome://linktargetfinder/content/select.xul", "aboutMyExtension", "chrome,centerscreen", null);

}

function getFieldsInput()
{
```

```
		var array_Elements = content.document.getElementsByTagName("input");
		return array_Elements;
}

function getFieldsSelect()
{
		var array_Elements1 = content.document.getElementsByTagName("select");
		return array_Elements1;
}

function onToolbarButtonCommand()
{
netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
var prefs = Components.classes["@mozilla.org/preferences-service;1"].getService(Components.interfaces.nsIPrefBranch);
var ww = Components.classes["@mozilla.org/embedcomp/window-
watcher;1"].getService(Components.interfaces.nsIWindowWatcher);
var win = ww.openWindow(null, "chrome://linktargetfinder/content/select.xul", "aboutMyExtension", "chrome,centerscreen",
null);
}

function saveFile()
{
		netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
		var nsIFilePicker = Components.interfaces.nsIFilePicker;
		var fp = Components.classes["@mozilla.org/filepicker;1"].createInstance(nsIFilePicker);
		fp.init(window, "Save a File", nsIFilePicker.modeSave);
		fp.appendFilters(nsIFilePicker.filterAll);
		fp.defaultExtension=("txt");
		fp.displayDirectory;
		var res = fp.show();
		if(res==nsIFilePicker.returnOK || res == nsIFilePicker.returnReplace){
				var theFile = fp.file;
				theFile.create( Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 420 );
				setFile(theFile);
		}
}

//Are set by saveFile
function setFile(theFile)
{
```

```
		var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefService).getBranch("extensions.linktargetfinder.");
		prefs.setComplexValue("workingFile", Components.interfaces.nsILocalFile, theFile);
}

function writeToFile(stringToAdd)
{
		netscape.security.PrivilegeManager.enablePrivilege('UniversalXPConnect');
		Components.utils.import("resource://gre/modules/NetUtil.jsm");
		Components.utils.import("resource://gre/modules/FileUtils.jsm");
		var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefService).getBranch("extensions.linktargetfinder.");
		var file = prefs.getComplexValue("workingFile", Components.interfaces.nsILocalFile);
		if(file==null&&file=="")
		{
			alert("The file are null");
			return;
		}
		// file is nsIFile, data is a string
		var foStream = Components.classes["@mozilla.org/network/file-output-
stream;1"].createInstance(Components.interfaces.nsIFileOutputStream);
		foStream.init(file, 0x02 | 0x10 | 0, 0, 0);
		// write, create, truncate

		var converter = Components.classes["@mozilla.org/intl/converter-output-stream;1"].

		createInstance(Components.interfaces.nsIConverterOutputStream);
		converter.init(foStream, "UTF-8", 0, 0);
		converter.writeString(stringToAdd);
		converter.close(); // this closes foStream
}

function write(){
		var prefs = Components.classes["@mozilla.org/preferences-
service;1"].getService(Components.interfaces.nsIPrefBranch);

if((prefs.getBoolPref("extensions.linktargetfinder.record_URL"))&&(prefs.getBoolPref("extensions.linktargetfinder.record_In
data")))
		{
			writeToFile("\n URL: "+content.document.location.href+"\n");
		}
		if(prefs.getBoolPref("extensions.linktargetfinder.record_Indata"))
```

```
{
		if(prefs.getBoolPref("extensions.linktargetfinder.select")){
			var array = getFieldsSelect();
			for(i=0; i<array.length; i++){
				writeToFile(array[i].type+" ");
				if(array[i].id!=""){
					writeToFile("ID="+array[i].id+" ");
				}
				writeToFile(array[i].value+"\n");
			}
		}
}
if(prefs.getBoolPref("extensions.linktargetfinder.record_Indata"))
{
		var array = getFieldsInput();
		for(i=0; i<array.length; i++){
		if(array[i].type=="hidden"){
			if(prefs.getBoolPref("extensions.linktargetfinder.input_hidden")){
				writeToFile(array[i].type+ " ");
				if(array[i].id!=""){
					writeToFile("ID="+array[i].id+" ");
				}
				writeToFile(array[i].value+"\n");

				}
			}
		else if(array[i].type=="text"){
			if(prefs.getBoolPref("extensions.linktargetfinder.input_text")){
				writeToFile(array[i].type+ " ");
				if(array[i].id!=""){
					writeToFile("ID="+array[i].id+" ");
				}
				writeToFile(array[i].value+"\n");
				}
			}
		else if(array[i].type=="radio"){
			if(prefs.getBoolPref("extensions.linktargetfinder.input_radio")){
				writeToFile(array[i].type+ " ");
				writeToFile("ID="+array[i].id+" ");
				writeToFile(array[i].value+"\n");
```

```
                }
        }
else if(array[i].type=="checkbox"){
        if(prefs.getBoolPref("extensions.linktargetfinder.input_checkbox")){
                writeToFile(array[i].type+ " ");
                if(array[i].id!=""){
                        writeToFile("ID="+array[i].id+" ");
                }
                writeToFile(array[i].checked+"\n");
                }
        }
else if(array[i].type=="password"){
        if(prefs.getBoolPref("extensions.linktargetfinder.input_password")){
                writeToFile(array[i].type+ " ");
                if(array[i].id!=""){
                        writeToFile("ID="+array[i].id+" ");
                }
                writeToFile(array[i].value+"\n");
                }
        }
else if (array[i].type=="submit"){
        if(prefs.getBoolPref("extensions.linktargetfinder.input_submit")){
                writeToFile(array[i].type+ " ");
                if(array[i].id!=""){
                        writeToFile("ID="+array[i].id+" ");
                }
                writeToFile(array[i].value+"\n");
                }
        }

else if (array[i].type=="button"){
        if(prefs.getBoolPref("extensions.linktargetfinder.input_button")){
                writeToFile(array[i].type+ " ");
                if(array[i].id!=""){
                        writeToFile("ID="+array[i].id+" ");
                }
                writeToFile(array[i].value+"\n");
                }
        }
```

```
                        else if (array[i].type=="textarea"){
                                if(prefs.getBoolPref("extensions.linktargetfinder.input_textarea")){
                                        writeToFile(array[i].type+ " ");
                                        if(array[i].id!=""){
                                                writeToFile("ID="+array[i].id+" ");
                                        }
                                        writeToFile(array[i].value+"\n");
                                        }
                                }
                        else{
                                        alert("något har försvunnit som inte borde ha försvunnit");
                                }
                        }
                }
}

function openFile()
{
                alert("openFile file");
                var nsIFilePicker = Components.interfaces.nsIFilePicker;
                var fp = Components.classes["@mozilla.org/filepicker;1"].createInstance(nsIFilePicker);
                fp.init(window, "Select file", nsIFilePicker.modeOpen);
                fp.appendFilters(nsIFilePicker.filterText | nsIFilePicker.filterAll);
                var res = fp.show();
                if(res==nsIFilePicker.returnOK){
                        var theFile = fp.File;
                        alert("openFile = fp.File");
                }
}

window.addEventListener("pagehide", OnPageUnload, false);
function OnPageUnload(aEvent) {
                if (aEvent.originalTarget instanceof HTMLDocument) {
    // var doc = aEvent.originalTarget;
                        write();
  }
}
```