# How to perform Exploratory Testing by using Test Charters

By

ENEA

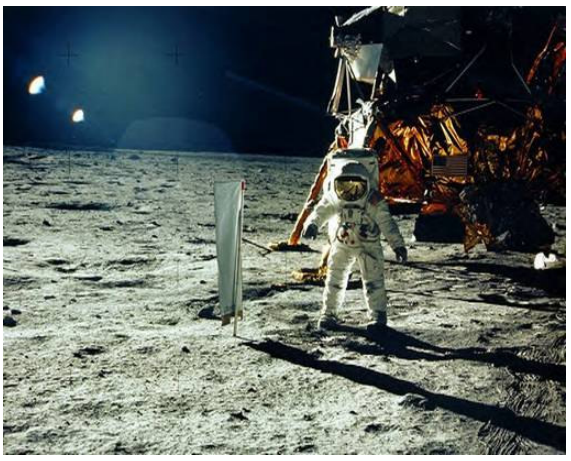
Anders Claesson
ancl@enea.se





## Contents





test Obsessed

# Introduction

- This presentation will give you an overview and understanding of the concept of **Exploratory Testing**.
- You will also learn how to use it from the examples and templates included in this material.
- Other Test Methods are also explained to get an overall picture where Exploratory Testing (ET) can be combined and mixed with other test methods.
- ET and the test charters are used to guide the tester in their exploration, which also have the positive effects of decreasing the required (traditional) test documentation (TS's) by at least 90% without losing any significant information of what is to be tested.
- Logging and test notes are very important, to know what have been covered by the tests.

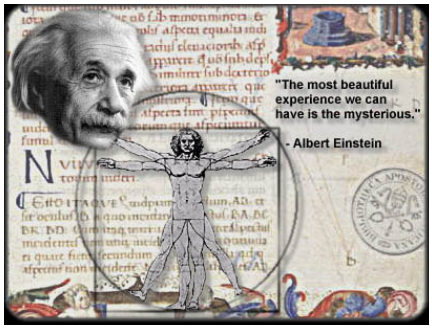





# Exploratory Testing definition

## Concurrent Test Design and Execution

- The most simple definition of exploratory testing is: **test design and test execution made at the same time** where the outcome from your most recent test guide you what the next test will be.
- The starting points for your exploration are the specified test ideas and mission/purpose defined in the used test charter.
- ET is the opposite of scripted testing (predefined test procedures, whether manual or automated).
- Exploratory testing is not an unstructured activity. **Test charters** on a higher level **are used instead of detailed test specifications**.




See www.satisfice.com for more information

# The purpose of testing

Testing = Questioning a product in order to evaluate it.

**Find the most important problems in the system before the customer do.**

Assure all requirements, needs and expectations are fulfilled.

Assure all important quality factors are satisfied.

Identify and mitigate all critical risks that may threaten the value of the product.

Leading to a satisfied and happy customer. →





# Find out what the customer value the most:

| User/Customer Value Analysis | | | | | |
|---|---|---|---|---|---|
| Quality Attribute | Importance Weight % | Customer Satisfaction | Priority IW / CS | Risk/Cost of failure | |
| 1. Reliability | 30 | 0,9 | 33 | High | Critical |
| 2. Availability | 15 | 0,8 | 18 | High | Critical |
| 3. Usability | 20 | 1,1 | 18 | Medium | High |
| 4. Performance | 20 | 1,1 | 18 | High | Critical |
| 5. Service Level | 10 | 1,0 | 10 | High | High |
| 6. Maintainability | 5 | 0,8 | 6 | Medium | Low |
| Summary: | 100 % | X=0,96 | | | |

Supplier costs

Customer costs

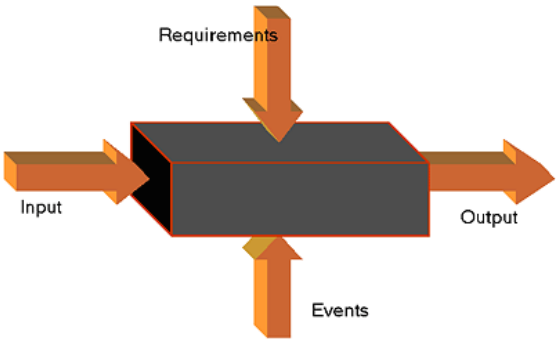
Also analyze the complexity of the system including all valid and prioritized combinations of features and platforms.

X < 1: Less satisfied
X > 1: More satisfied

## Context driven testing

[http://www.context-driven-testing.com/]

1. The value of any practice depends on its context.
2. There are good practices in context, but there are no best practices.
3. People, working together, are the most important part of any project's context.
4. Projects unfold over time in ways that are often not predictable.
5. The product is a solution. If the problem isn't solved, the product doesn't work.
6. Good software testing is a challenging intellectual process.
7. Only through judgment and skill, exercised cooperatively throughout the entire project, are we able to do the right things at the right times to effectively test our products.

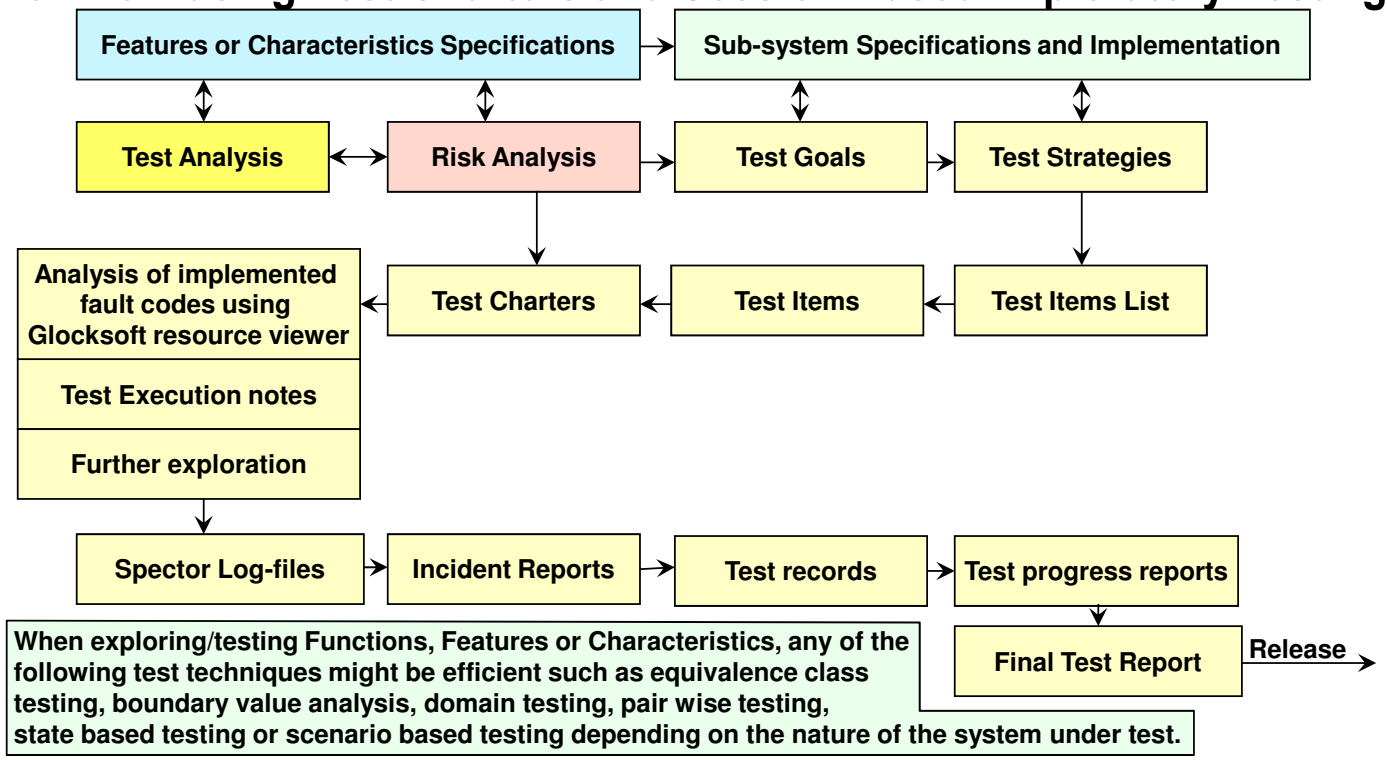



*A Black Box view of a system under test.*





## Workflow using Test Charters and Session Based Exploratory Testing



# Test Methods: Risk Based Testing

**Objective**: Risk Based Testing is a method of identifying potential problems early (by a technical Risk Analysis) and focus testing in these areas. The risk based approach gives the tester the opportunity to discuss the problems before they occur and possibly prevent them. The amount of Test Cases to run can be decided based on the current risks. The test effort can be continuously changed and re-focused based on the main problem areas found during the progress of the project and the test execution.

**Limitations**: It takes a lot of experience, skill and preparation to find the risks in the first place. It might be difficult to get the key persons participating in the risk analysis meeting and there might not be enough time to discuss the potential problems in detail. After the risks have been identified people often don't know what to do with them, or they tend to '"forget" their analysis because they are so busy trying to get their things to work in the first place (or just continue working as they are used to).


Note that it is much better to, at least, attempt finding the problems before they occur to be able to get the designers prevent some of them before they hit the system and a lot of costly correction work have to be done. Testers can also focus their test effort on the most critical areas from the beginning of the test period. Low risk test cases can also be removed completely.





# The Risk Based Testing process

| | | |
|---|---|---|
| 1 | Scope | Define the scope of the risk analysis, e.g. robustness. |
| 2 | Input data | Identify and collect all input information needed for the risk analysis. |
| 3 | Prep. meeting | Present all input information for the risk analysis participants. |
| 4 | Risk analysis | Identify the risks, prioritize, suggest actions including the name of the action responsible. |
| 5 | TC selection | Select test strategies, test ideas to evaluate the risks (consider other aspects as well). |
| 6 | Follow up | Have weekly follow-up of the progress of the risk evaluation (testing) and mitigation. |
| 7 | Evaluation meeting | At the end of the iteration, go through how much of the test object was actually covered, how many test charters/ideas that were executed, the risks that were covered, how many faults were found (any stopping or severe faults), leftover Test Cases, if the initial test strategy worked or not, and why. This learning process is very important in order to improve in the next iteration! |

For the next iteration, start the process over again with the new features/ functions being delivered from design. The second run usually goes faster when the risk process has been learned by the team.

# Risk example

| Prio | Risk description | P*C, T | Actions |
|---|---|---|---|
| 1 | **Risk:** Unstable system platform. The interconnections and interactions between the functions in the platform may not work.<br>**Why:** Purchased platform components for this system developed by 10 independent, third party vendors. Components were not tested together by vendors.<br>**Why:** Application parameters are not set correctly.<br>**How:** May cause failures in the communication and the functional behavior of the application functions.<br>**When:** Happens when the application starts up and when user data is entered into the system.<br>**Where:** Problems may be visible in the interface between the application and the middleware.<br>**Which part:** The faults reside in the functions of the HP Unix server operating system. | 5 * 3 = 15<br>T = 2<br>P = Probability<br>C = Consequence<br>T = Testability | **Risk reduction and evaluation strategy**:<br>Ask 3PP vendors how they have tested their products.<br>Perform platform tests to evaluate the interactions between components.<br>**Test items**:<br>Platform test of system xxx.<br>**Test ideas**:<br>- Test of individual important functions.<br>- Tests of related function groups, bottom up to the application interface level. |





# Exploratory test preparation - selection and prioritization of test ideas

**1 Identify test items (charters covering a Use Case or a feature, function or char.)**
- Identify which test items/charters to include based on the test scope and the test goals for the current area. A Test item could be a function, feature, Use Case, characteristic, etc.

**2 Identify the number of required test ideas based on your ranking below.**
- How much testing is needed to cover the areas within your test charters?

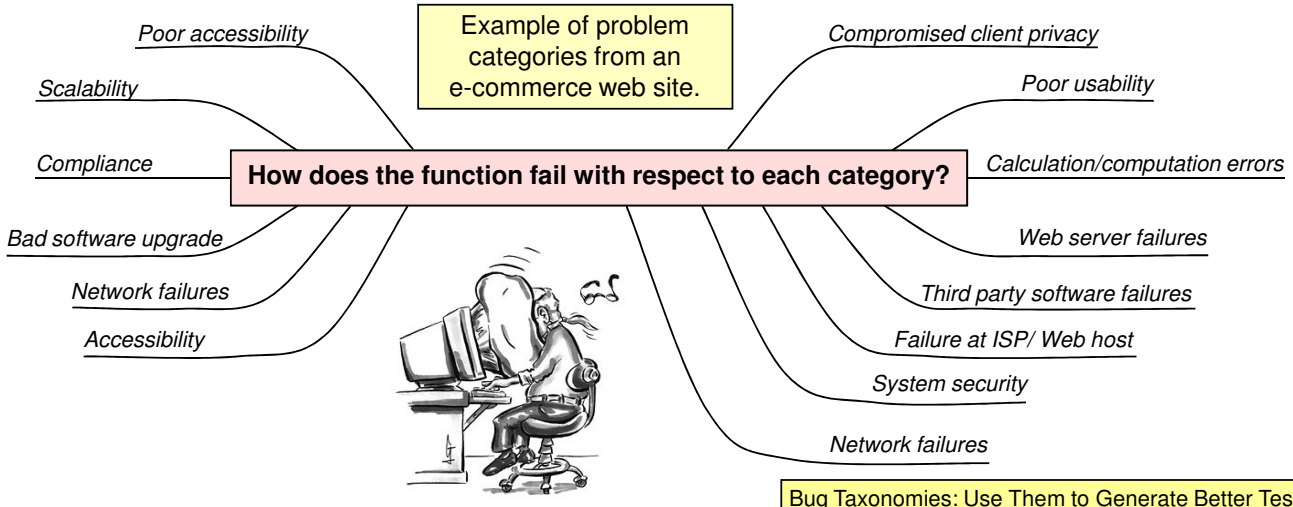
**3 Select the most important test ideas based on the following attributes:**

| No | Attribute | Description | Ranking |
|---|---|---|---|
| 1 | **Risk** | Risk level for the area (Feature, Characteristic or system part). | **H, M, L** |
| 2 | **Required** | Priority of the requirement (from the customer and/or product management). | **H, M, L** |
| 3 | **Complexity** | How complex the feature, product or characteristic is with regard to internal logic and external connections to other components, systems, interfaces and/or amount of units/features (3PP, multiple usage of features, simultaneous users, background load/traffic mix, etc.) to be tested. | **H, M, L** |
| 4 | **Value** | How much the customer value the current area (Feature, Characteristic or system part) in their business/operation. | **H, M, L** |
| 5 | **Importance** | Importance of the system part, interface or function to achieve a stable, responsive and capable operation of the system (e.g. restart functions). | **H, M, L** |
| 6 | **Failure** | Severity if failure occurs in the area to be covered. | **H, M, L** |
| 7 | **Probability** | Probability of how likely a failure is in the area. | **H, M, L** |
| 8 | **Cost** | Cost of failure (both internal and for the customer). | **H, M, L** |
| 9 | **Visibility** | Failure visibility for the user. | **H, M, L** |
| 10 | **Tolerability** | Tolerability of failure, i.e. how tolerant the customer/user is of a failure occurring. | **H, M, L** |
| 11 | **Human factors** | How usable the system is in the current area. | **H, M, L** |

For more information about how to select Test Cases, see the book: Software Testing Fundamentals by Marnie Hutcheson
http://www.ideva.com/

# Exploratory test preparation

- Create a **Bug Taxonomy**: Categorize common types of faults found in previous projects and in operation (including Root Cause Analyzes), feedback from maintenance is very important. The taxonomy is a good input for the identification of risks and selection of test ideas. Look for **systematic faults** your designers usually do in every project.

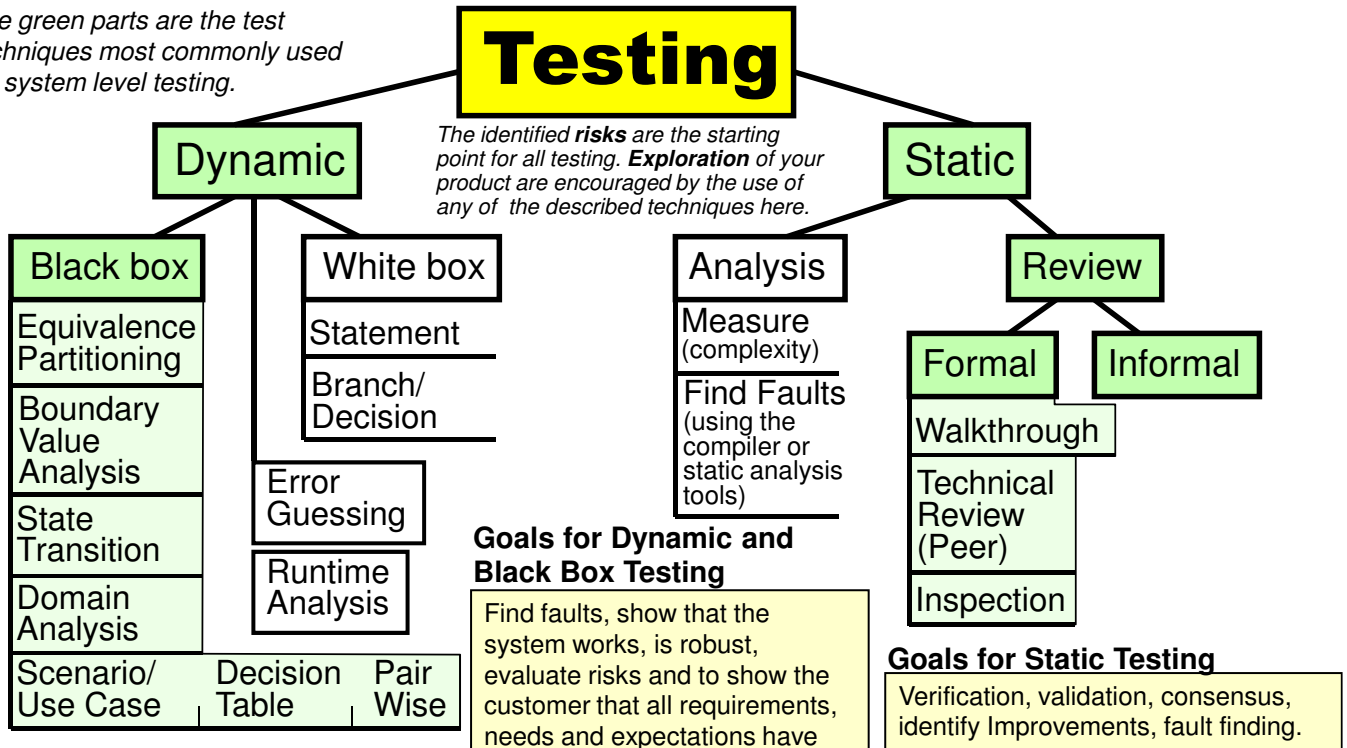



Bug Taxonomies: Use Them to Generate Better Tests [http://www.testingeducation.org/a/bugtax.pdf]





*The green parts are the test techniques most commonly used for system level testing.*




*The identified **risks** are the starting point for all testing. **Exploration** of your product are encouraged by the use of any of the described techniques here.*

**Goals for Dynamic and Black Box Testing**

Find faults, show that the system works, is robust, evaluate risks and to show the customer that all requirements, needs and expectations have been met.

**Goals for Static Testing**

Verification, validation, consensus, identify Improvements, fault finding.

# A test charter template for Exploratory Testing

**Test Charter id and unique name**

**Actor/Theme:** <The user role you as a tester is going to play when using the system or theme when testing system characteristics>

**Purpose:** <The purpose of the charter including what particular objectives the actor is supposed to achieve or what the expected outcome of the theme exploration should be>

**Setup:** <What needs to be in place in order to start using the intended function or characteristic , i.e. the precondition to be able to test (configuration and the initial system state, e.g. a start menu should appear on the screen with login fields or what background load scenarios should be up and running)>

**Priority:** <High/low, based on what the priority of the requirements are, or other considerations (risk level)>

**Reference:** <Specification (e.g. Requirements Specification), risk or other information source>

**Data:** <Reference to a data file with input data to be used, e.g. an Excel sheet. Variations of data to be made based on the initial data can also be specified, e.g. use boundary values of valid and invalid input data for all parameters, use pairings of input data to catch combinatorial problems>

**Activities:** <What the actor should do with the system, e.g. Log on to the system as a super user:
- Add a new user.
- Attempt to delete a non existing user>

*Test ideas*

**Oracle notes:** <Things to pay special attention to regarding system response and what is expected as an adequate/correct outcome, e.g. Check that correct error messages are given that informs the user about what mistake he made and how to do it correctly next time. It could also be a reminder to the tester to check for buffer consumption or other important resource usage>

**Variations:** <Additional system or user events that could occur at anytime the normal activities are performed as described above under Activities, e.g. another super user logs on to the system trying to manipulate the same data as the first is working on, or the network connection suddenly fails>

For more information about **test charters** with examples see: http://www.satisfice.com/rst-appendices.pdf, http://www.satisfice.com/rst.pdf





# The test charter

- A typical test charter includes the work of **a pair** of testers working together for a 90 minutes period of **uninterrupted** time, i.e. time boxing (cell phones shut off, no email pop-ups active, etc.).
- The test charter should suggest what to be tested, how it should be tested and what problems to look for (OBS, not a detailed step-by-step instruction).
- Chartered testing is mainly used for exploratory testing and require more system and testing knowledge than scripted testing (but the fault finding efficiency is 2 - 4 times higher). Try to play chess with someone using a predefined strategy without consider changing the strategy based on what moves your opponent makes.
- Chartered testing encourage testers to react on the response from the system and think for themselves what a correct outcome should look like and also use their creativity how to break the system. The tester will also be encouraged to look for different ways how it might fail to meet the users needs expectations and requirements (e.g. usability problems).

# The test charter

- A test charter can be seen as a **mirror** of a Use Case. The Use Case represents the design view, while the test charter represents the test view, i.e.:
  1. What data, test environment and configuration do we want to use when testing this Use Case or characteristic?
  2. From which actor or theme do we want to give input and observe the output when exercising the specific paths we want to cover in the Use Case, or quality attributes for the characteristics.
  3. The test ideas are the starting point for the exploration and are similar to Test Cases, with the **main difference** that it is **not** a step-by-step description of how to perform the test, it is a one line statement of what might be a good starting point regarding what to do with the feature to cover the Use Case, its paths and to **find the most serious problems as fast as possible**. If the tester can't figure out how to use the feature or evaluate the characteristic for him/herself, how could we expect the users to understand how the feature should be used?
- This way of working with exploratory and chartered testing is a **completely different approach**, which might need some discussions and acceptance on management level. Instead of reporting how many Test Cases were executed the testers report how many charters and how much test effort have been subjected to a specific feature or characteristic based on the current risk levels and other factors.



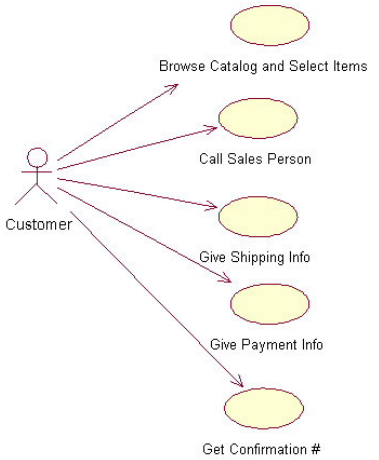


# The test charter

An illustration of the **two views** comparing Use Cases and test charters:

A **use case** is a single task, performed by the end user of a system, that has some useful outcome. I.e. **WHAT** the user should be able to do with the system.

A **test charter** is a description of **HOW** an end user is **supposed to use the system** in a real situation by executing specified tasks in different ways. The intention is to execute what may be useful for him/her or any other system actor, which may show weaknesses in the way it is intended to be used considering what the user wants, needs and expects.





**Role description** of the actor and the **theme** for the exploration, i.e. what do we want to find out about this feature, actor or characteristic within the scope of this charter?.

**Test setup**, **data** and suggested **test methods/techniques** to be used. Try to use a customer like environment to catch any compatibility issues. Also outline an overall test strategy/ approach to be used in this charter (in the purpose section).


**Activities** described as test ideas how to perform the different parts the actor should be able to do. Test ideas for negative testing should also be included.

Things to **pay special attention to** when executing different parts (known weaknesses or checking error messages for relevant or misleading information). Find useful test oracles.

**Suggested variations** when executing the different parts of the Use Case, e.g. interruptions, interactions with other Use Cases, etc. that can happen anytime during the use of the feature.

# Example of a test charter for a 90 minutes test session

**Test Charter 1: Analyze the copy/paste function of pictures**

**Actor:** Normal user

**Purpose:** To evaluate if the copy/paste function of pictures works in our word processor together with the most commonly used word processors on the market (Word, Power Point., etc.) and other programs where pictures can be inserted in the copy buffer for copy/paste operations in the PC. The purpose is also to see that no information is lost or corrupted.

**Setup:** A Dell PC with 2 Gb memory, our word processor, the Microsoft Office package professional and the home edition 2003 patch level xx, PDF reader version 7.1, Notepad version 4, Internet Explorer version 7, Opera version 5, Mozilla……etc. (the setup might be common for several charters and can therefore be described and referred to instead of repeating the same information in every charter).

**Priority:** **High**, because this function is used very frequently both within our own word processor, but also between other word processors and programs, the user may want to copy/paste pictures with ours.

**Reference:** **Requirement abc_1** in document Def-07:034 revision R1.1. **Risk number 3** from the risk assessment held on April 4 2007 regarding the copy function, which is documented and followed up in the document Rsk-07:012.

**Data:** A variety of pictures with different resolutions, both vector graphics as well as bit mapped pictures. The pictures could be photos or figures in web browsers for example. Complex pictures are also included and pictures which might be copy protected in some way.

**Activities:**

1. Copy/paste a picture in our word processor from one location to another in the same document.
2. Copy/paste a picture to and from our word processor into/from Word, PowerPoint and Excel.
3. Copy a picture into our word processor from a variety of the mostly used Web browsers.
4. Copy/paste a picture to/from the most commonly used web site building tools such as Dreamweaver from Adobe.
5. Copy a picture from a PDF document using Acrobat Reader into our word processor.
6. Try to copy a write protected picture from the web or other source into our word processor.
7. Try to copy/paste a variety of non readable and some readable ASCII-characters or corrupted pictures.





# A simple example of a test charter, continued….

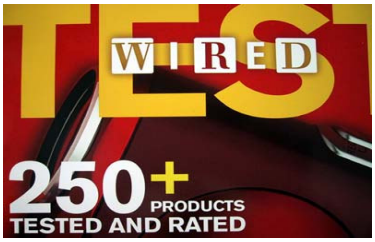
**Test Charter 1: Analyze the copy/paste function of pictures, continued….**

**Oracle notes:**

- Look if the size of the pasted picture changed on the screen (it should not).
- Check if there is any loss in the resolution of the picture especially when you copy and paste from other programs to or from ours.
- Check which is the highest resolution picture that can be copied and how that affect the system.
- Check for memory leaks and how much memory the copy/paste operation takes from the system and how that affects the use of other programs. Other programs should not slow down or be affected in any way.
- Print pasted pictures to see if there are any differences in color, resolution or any other anomaly.

**Variations:**

- Try out (and find) the boundaries of how large pictures are possible to copy/paste.
- Perform copy/paste with a large number of items in the copy/paste buffer in your PC.
- Try to fill the copy/paste buffer to its limit and then copy/paste a picture and see what happens.
- Try the longest file name of the picture you can type before making the copy/paste. You may use the tool Perlclip (download from http://www.satisfice.com/tools/perlclip.zip) to generate file names with a million characters or more if you want.



# Example of a Test Charter for Robustness Testing

**Test Charter 2:**

**Theme:** CEP handling (check of internal node communication).

**Purpose:** To trigger failures in the CEP handling. For the internal communication in the node, many changes have been made in design which may be a high risk. Faults have been reported from the customer and tests made by the Product Line Maintenance organization. If CEP fails in operation, it can not be found at the customer site. Communication is interrupted. Cells can not, for example, be possible to get up and running.

**Setup:** Minimum C-node configuration.

**Priority:** High.

**Reference:** IR <abc_identity> and system documents:
Xxxx Movable CEP in RNC,
Yyyy Node Module MP's in Pool.

**Data:** Traffic transactions through the node (BAT). Load modules interacting with CEP handling (see below).

**Activities:**

| No | Prio | Description |
|---|---|---|
| 1 | H | Module Main Processor (mMP) Plug In Unit (PIU) locking. |
| 2 | H | Perform restarts of mMP or RPU switching during interactions with CEP handling. |
| 3 | H | Restart of LmCell (Load Module for handling cell administration). |
| 4 | H | Restart of UNI-SAAL (CEP resource control program). |
| 5 | H | Restart of RoamFro (xx). |
| 6 | H | RPU switch of RoamFro (xx). |
| 7 | H | Remodule IubLinks. |



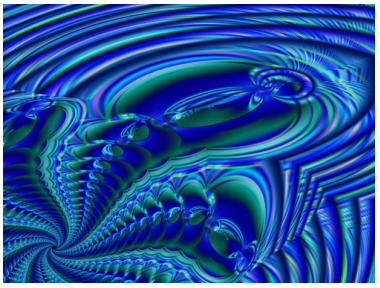


# A simple example of a test charter, continued….

**Test Charter 1: Analyze the copy/paste function of pictures, continued….**

**Oracle notes:**
- Check that UNI-SAAL Termination Point (TP) moves successfully after 120 seconds to another module xx when yyy is locked.
- Check that restarts don't disturbs traffic handling or the system performance.
- Check that it is possible to perform RPU switch back after board restarted, board removed or blocked.
- Check trace logs for error messages.
- Check status of the node/boards.

**Variations:**
- Select a number of load modules that interacts with CEP handling when performing restarts or RPU switching.
- Perform cold, warm and refresh restarts.
- Restart board, block board or remove to trigger switch to redundant board.
- Or other variations that the tester might find useful.



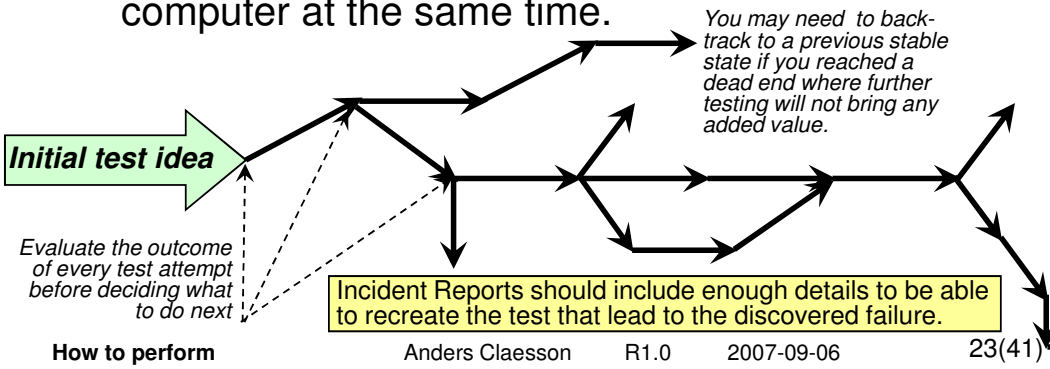
# Exploratory test execution

1. Keep your mission clearly in mind.
2. Keep notes that help you report what you did and why you did it, that support your assessment of the perceived product quality.
3. Keep track of questions and issues raised in your exploration (constant logging, taking notes and debriefing after each test session).
4. To supercharge your testing, pair up with another tester and test the same thing on the same computer at the same time.

***Expect the unexpected***

**You never know what you will find.**

Continue on the paths that evokes your curiosity until you have reached a point where you don't see anything suspicious anymore, and you consider the probability of any remaining faults being reduced to a level where there are no major risks in that that area.





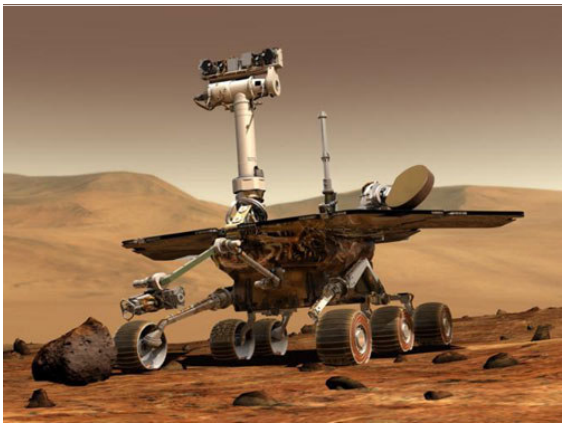



# Exploratory test execution

- The results from the tests you design and execute influence the next test you will choose to design and execute.
- You build a mental model of the product while you test it. This model includes what the product is and how it behaves, and how it's supposed to behave.
- You test what you know about, and you are alert for clues about behaviors and aspects of the product that you don't yet know about. But beware of your biases. You may only see what you expect to see.

The more we can make testing intellectually rich and fluid, the more likely we will hit upon the right tests at the right time [Bach].

By writing down test scripts in advance and following them tends to disrupt the intellectual processes and make testers unable to find important problems quickly [Bach].



# Exploratory test execution - Taking testing notes

When performing exploratory testing it is **very important to take notes** of what you are doing, seeing or suspect, which could be an anomaly or a fault. Make your own conclusions regarding the quality of the function based on the results from your explorations.

| Category | Notes |
|---|---|
| **Test coverage** | <What input data have I studied and given feedback on to improve its quality? Which reviews have I participated in?<br>How much have I covered in my test execution and how much remain> |
| **Evaluation notes** | <What have I learned so far by executing the test ideas/attempts for these specific functions? Does it seem stable? Have I found any faults?<br>What test attempts am I going to do next based on my idea list and current observations> |
| **Risk/Strategy list** | <Which risks have been covered and which ones remain among the most important ones? Which part of the initial test strategy should I follow and does it need any changes?> |
| **Test execution log** | <Any specific notes I would like to highlight in addition to the recordings made during the test execution> |
| **Issues, Questions and Anomalies** | <Any strange behavior I don't understand. Is there anything wrong with my test idea/test attempt, test environment, test data, misunderstanding of the function, test script or the system under test?> |





# Useful tools for Exploratory testing

- **Glocksoft Resource Viewer** is intended for viewing of resources of executable files (with the extension .exe and .dll). It displays information about program resources including dialogs, icons, strings (such as error messages the developer has put into the code but "forgot" to document for the user) and more, it allows you to open any executable windows or resource file and gain access to the Windows resources that it contains. The cost of the tool is $75 per license for one PC.
  http://www.glocksoft.com/resource_viewer.htm

- **Spector Pro** is used for monitoring and recording every detail of all PC and Internet activities (logs everything you do, with time stamps). All details of what you do on the computer – chats, instant messages, emails, the web sites visited, what searches are made, posted pictures and which you have looked at, all keystrokes, all programs that have been run on the computer and much more. And because of its advanced surveillance screen snapshot features, you get to see not only WHAT is happening on the screen, but the EXACT order in which it is done, step by step. The cost of the tool is $99,95 per license for one PC.
  http://www.spectorsoft.com/products/SpectorPro_Windows/

## Testing combinatorial explosions with a Two-wise parameter selection

- One of the most difficult testing problems is the demand for flexibility in many systems where the behavior of the available Features should be possible to change and adapt according to customers' needs.
- There could be thousands of parameters that should be possible to change. Features could be optional and/or adapted in many ways. Also interactions with other Features may be possible to tailor for a specific customer and application.
- When the flexibility increase the number of possible tests grow exponentially which makes testing much more difficult (and often impossible to cover all combinations) if using traditional methods. Fortunately there exist techniques to handle such situations. The test technique is called n-wise combinatorial testing.
- All two-wise parameter combinations are included in the example to the right covering each value at least once. The application under test are then added for each configuration.

**Two-wise configuration combinations**

| Combination | OS | Browser | Resolution |
|---|---|---|---|
| 1 | Win XP | Netscape | 1024 x 768 |
| 2 | Win 98 | IE | 640 x 480 |
| 3 | Win XP | IE | 800 x 600 |
| 4 | Win ME | Netscape | 640 x 480 |
| 5 | Win 98 | Netscape | 800 x 600 |
| 6 | Win ME | IE | 1024 x 768 |
| 7 | Win XP | IE | 640 x 480 |
| 8 | Win ME | IE | 800 x 600 |
| 9 | Win 98 | Netscape | 1024 x 768 |





## Why Two-wise testing finds many logical and data dependent faults

- Most faults are of the category of:
  **One Factor faults**
  A one factor fault is a consistent problem triggered by any of the possible input parameters equivalence classes (either only one, or up to all of the equivalence classes from one to all parameters) used by the function. The function under test does not work, and any test using that input value range or a specific value in the range of that function would find the fault.
- **Two Factor faults**
  If there exists a consistent problem when specific equivalence classes with two parameter values occur together, it is called a two factor fault. Indeed, a two factor fault is an indication of a two-wise incompatibility or incorrect interaction between two parameters and their selected values. It is the pairing of this functions specific parameter value (or range of values) with another functions specific parameter value (or range) that fails even though all other pairings work properly.

# Why Two-Wise testing find many logical and data dependent faults

A two-wise coverage approach would find most of the faults (70 - 98%) when testing possible parameter values used by a large numbers of functions and their parameter values combined. Tests can be reduced by 99,99999…% or more compared to exhaustive testing, which is impossible in most cases.

**Smarter Testing finds more faults in less time.**





# A useful tool for Two-wise/Pair-wise testing

- **PICT** The Pair-wise Independent Combinatorial Testing tool (PICT) can help you efficiently design test cases and test configurations for software systems. With PICT, you can generate tests that are more effective than manually generated tests and create them in a fraction of the time required by hands-on test case design. PICT generates a compact set of parameter value choices that represent the test cases you should use to get comprehensive combinatorial coverage of your parameters.

  http://download.microsoft.com/download/f/5/5/f55484df-8494-48fa-8dbd-8c6f76cc014b/pict33.msi

# Test reporting

| Test area | Initial Risk Level | Needed test effort | Current risk level | Current test effort | W 1 | W 2 | W 3 | W 4 | W 5 | W 6 | Q ass. | Comments |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Area 1** | Low | Low, QL1 | **Low** | **None** | | | | QL0 | | QL1 | ☹ | Feature(s) not yet delivered from design and integration. RFT not fulfilled. No testing possible. |
| **Area 2** | Medium | Medium, QL2 | **Low** | High | **QL0** | | QL1 | **QL1** | **QL2** | | ☺ | On track, no faults. |
| **Area 3** | High | High, QL4 | **High** | **Blocked** | | | **QL0** | | QL1 | | ☹ | Crashes, IR12345 |
| **Area 4** | High | High, QL3 | **Medium** | **Pause** | **QL0** | | QL1 | **QL1** | QL2 | | 😐 | IR1212 under investigation. |
| **Area 5** | Medium | Medium, QL2 | **Medium** | High | **QL0** | | QL1 | | QL2 | | 😐 | Configuration problems |

**The Current Risk level is based on the results from all performed preventive actions and/or all performed tests at the current week when the progress report was written. The Quality Level includes required test coverage, the level of positive/negative tests, test effort and achieved stability under various usage and load conditions to be reached for a "sufficient" system quality.**

**= Delayed test item according to plan. Not Ready For Test from design and system integration.**

**QL1 = Planned ready date to achieve a specific Quality Level.**

**QL2 = QL-level achieved on time according to the test plan.**

**QL1 = QL-level achieved later than planned.**





# When to stop testing

## Coverage

All planned/required test ideas and test charters should have been executed and passed, according to the current risk areas/levels, where faults are found (by exploratory testing) and the coverage objectives stated in the test goals (e.g. system requirements, Use Cases, etc.).

## Quality

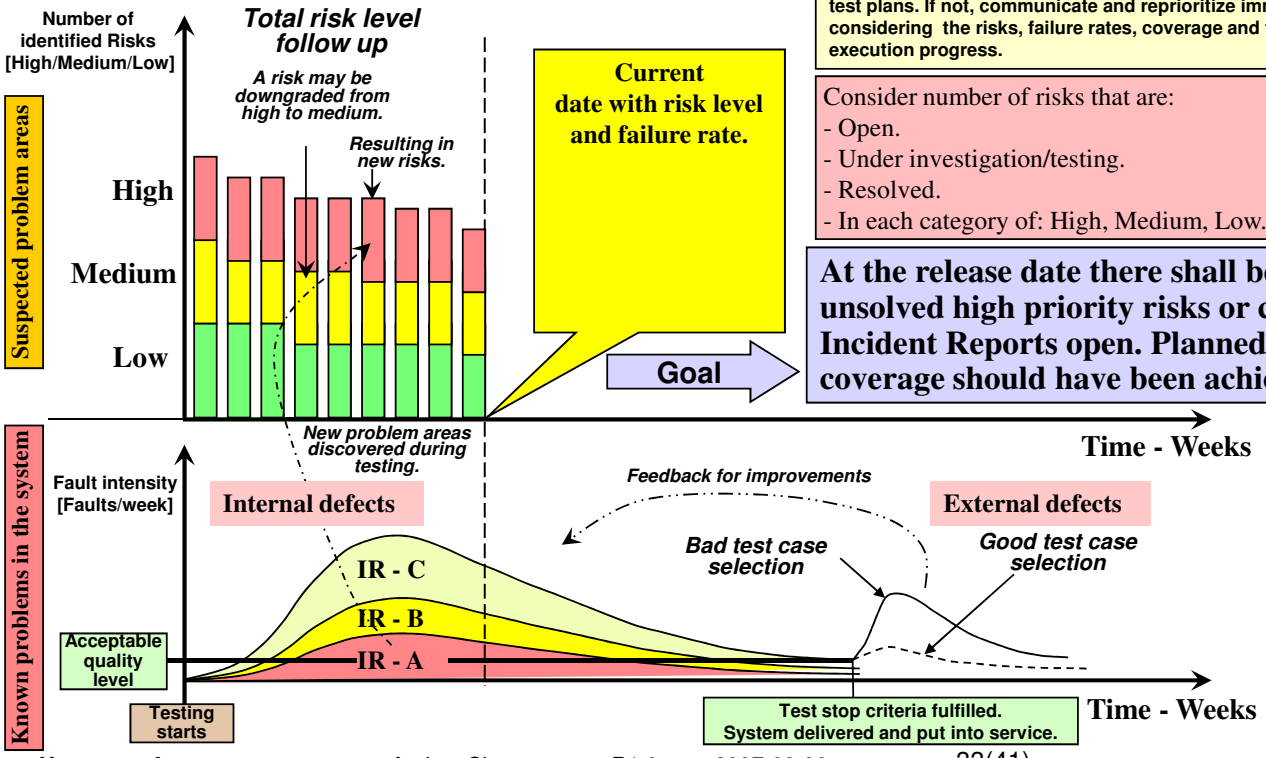
**Testing should stop when:**

- The probability of remaining faults is reduced to a level that can be accepted by the customer [B. Beizer].
- No open priority A IR's.
- The systems' **risk level** is within acceptable limits (i.e. no critical risks remain unsolved).
- The product value has been demonstrated and accepted (i.e. implicit and explicit quality attributes are satisfied).

## Time

When the agreed ship date has been reached.

# When to stop testing




When looking at the curves pay attention to the trends. Extrapolate and see if you are going to be able to meet your test plans. If not, communicate and reprioritize immediately by considering the risks, failure rates, coverage and test case execution progress.

Consider number of risks that are:
- Open.
- Under investigation/testing.
- Resolved.
- In each category of: High, Medium, Low.

**At the release date there shall be no unsolved high priority risks or critical Incident Reports open. Planned test coverage should have been achieved.**








Appendix A

# Example of forensic test methods used by the police

**The police has just arrived to a crime scene** in an office building somewhere in Stockholm, where at least 10 computers have been stolen. The main objective for the police is now to secure as much evidence they can. The personnel in the office building show them around starting with the place where the thieves entered the building. The technicians could now start their job. The first thing they do is to map how the suspects have moved around in the building. The technicians must use a lot of their imagination, creativity and reasoning based on their previous experiences and knowledge. They start by thinking – where would I put my hands and feet if I have climbed in this way? This is the way they need to think by putting themselves in the shoes of the suspect.

**On a paper lying on a desk** just beneath the smashed window they found a stain of blood. The police takes a Q-tip out of his bag and gently touch the stain to absorb some blood. This is all that is needed to get enough DNA and connect the perpetrator to the crime scene. If the DNA can be secured at the scene, the probability is quite high there is a hit in the DNA register of a suspect. It is very common that the same person has done several crimes in the past. At another crime scene a fingerprint on a milk box led to that the perpetrator could be connected to another 14 committed crimes in the past.

Appendix A

# Example of forensic test methods used by the police

Entry point to the crime scene.

The case with different test tools to be used for different test methods.

Securing evidence using the DNA-analysis test method.

Logging of test results for later analysis at the lab.

Forensic test methods are by its nature tools in the exploration of what really might have happened before the investigators arrive at a crime scene. They are very thorough in order to keep all the evidence and the pieces of the puzzle together in order to hold up in court. No clue is left unexplored, even if it doesn't make sense by itself the collection of all the evidence might give the whole picture which would not be possible to see by looking at each individual evidence. Software testers should work and think in the same way, but are rarely given the time for reflection by piecing all the individual parts together to give an accurate image of how the user may experience the product under development. Companies that value continuous improvements and see their customer as their most valuable input for feedback prosper because of their company culture (like Toyota) while others who listen more to their engineers suffer from the disconnection between what the customer wants and what the engineers like to do.

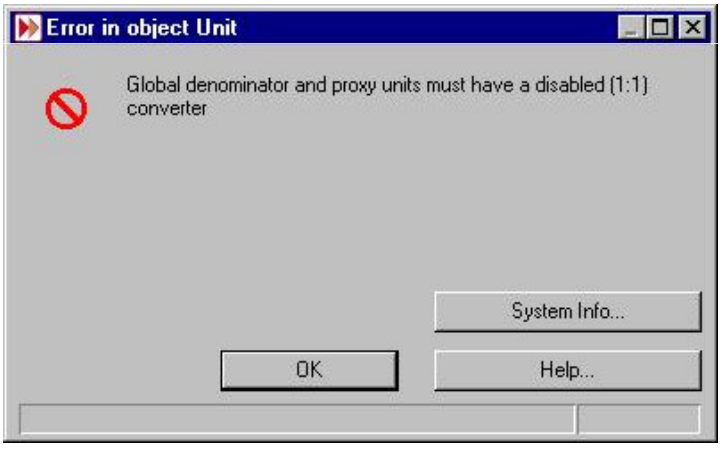




Appendix A

# Analogies with software testing

**You are testing in your lab and just found something weird** which may or may not be a fault (you don't know yet). But it intrigues you into further investigation. Your main objective as a tester is to find as many important problems in the product as possible before the customer do. You look at the evidence that caught your interest and try to catch some more information if possible. By looking into your log files and also analyze what actions you did before reaching the current state/situation will give you more clues. You can use a checklist of what to look for so you don't miss anything (in addition to your own creative thinking of what information might be useful).

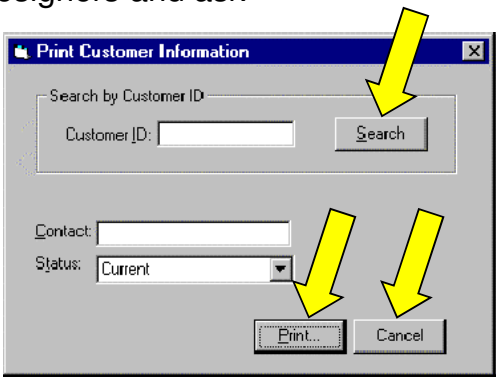


Appendix A

# Analogies with software testing

**Look at your test oracles** if you can find any help whether your test generated any undesired outcome from the users' and/or customers' point of view or not. Proceed by considering likely options of what to do next to get more information that supports your most probable hypothesis – where might the designers have misunderstood the requirements, used new untested technology, not followed the design rules, made any logical faults, used faulty assumptions, a lack of communication among designers regarding the interactions between their modules, skipped unit testing partly or completely (because of time pressure), etc. You don't know. Use available test methods, techniques and tools that best suit the situation. When you put yourself in the shoes of the designer (it is not forbidden to talk to the designers and ask questions!) you may get a better perception of their world and where they usually make mistakes. After exploring the most likely possibilities you either find the fault or dismiss it, considering your initial assumption whether there was any mistake made by the designer or not. You may also have misunderstood how to use the function you are testing (a testing mistake, but could on the other hand be a usability issue).

**Oracle example**: Command buttons should be used only for frequent or critical immediate actions








Appendix A

# Analogies with software testing

**Now put yourself in the shoes of the user**. What kind of exploration would you do in this case? Would it differ in any way compared to the designers view (even after you have looked at your test oracles including user/customer aspects)? Would the user be annoyed or irritated if your observation of the possible anomaly would go into the final version? Does the feature you are testing really solve the users' problems or abilities to reach his goals in an efficient way? Is your initial observation in any way causing the user to be mislead how to proceed, e.g. by a faulty error message or any other usability aspect?

**By using a bug taxonomy** you can create your own "DNA" record of observed problems, likely mistakes and faults made by the designers. In combination with Root Cause Analyses of the most serious faults found to date (where the faults reported by your customer should be the most important ones) we are in a far better position of making better risk assessments before testing starts, select more efficient test ideas and, when executing, quickly get the problems we see getting fixed by the developers and hopefully reduce any debates regarding incident report priorities. Things we know the user/customer regard as very serious should be paid special attention to in testing (and in the prioritization of which bugs to prioritize).

Appendix A

# Analogies with software testing

**An interesting observation** you may elaborate on is why we use the same (in most cases) test techniques as we did 30 years ago (except for pair-wise testing) when they were discovered (equivalence partitioning, boundary value analysis, etc.). There are no scientific research published to date that tells us the efficiency of our known and most used test techniques, which types of faults they find and which faults that slip through. There are no silver bullets in software testing but it would be very helpful to see in which context a certain set of techniques are efficient and why. The reason for such investigations not taking place is probably that companies don't want to expose their way of testing, which faults they find and how many that slip through. But you can at least make it in your own company.





# Experiences

- To create the test charter was made as a group activity and only took one hour to do.
- A blocking fault on the first test attempt/step halted further execution of the charter.
- All testers in the group felt it was more productive to create charters. One problem is that they have a lot of old test specifications and test cases. There is a resistance to change to something new. An idea they had was to refer to old test cases, when applicable in the list of test ideas, not to loose previous tests that had been productive.
- "System upgrade" will be the next area to create test charters for. Risk Based Testing will then be combined with Exploratory Testing.

# Conclusions

- Chartered testing focus on the purpose of testing to a higher degree than scripted testing, i.e. to find the most serious problems before the customer do.
- Chartered testing is in general 2 – 4 times more efficient in finding bugs than scripted testing.
- Scripted testing doesn't work as well as chartered testing under pressure.
  When forced to execute test cases there is no time to explore any anomalies.
- Chartered testing gives the tester more freedom to be creative to find the faults faster.
- Since ~50% of the requirements are faulty or incomplete, scripted tests usually becomes even worse.