Testing in Darkness

How to discover test requirements and stay out of the critical path in a crisis.

by Hans Schaefer
hans.schaefer@ieee.org

Contents

- The situation
- Collecting information
- Getting concrete requirements
- Prioritizing the test
- Staying outside the critical path

The situation

”Test this product” !

- No or incomplete
  - requirements
  - user documentation
  - design documentation
- You come in late for test planning
- Not enough time to test everything

Which test we talk about?

System or Acceptance Test

Test the product against user or customer or market requirements

Functions

- **pr. user / customer class**

Properties

- **generally**
- **pr. user**
- **pr. function**

User / customer is anybody who has an interest in the product.

Example what to test

Functions

- **Driver**
  - **Drive**
  - **Brake**
  - **Horn …**
- **Fireman**
  - **Firedoor**
  - **Waterpump(s)**
- **Operation office**
  - **Pulls trains of a certain weight at a certain speed**

Properties

- Generally
  Safety
- pr. User (driver)
  Comfort in cab
- pr. Function
  Speed of fire door opening

And how did I know all this?

Step 1: Collect information

What are the requirements?

Who knows about them?

Where and how much is the risk?

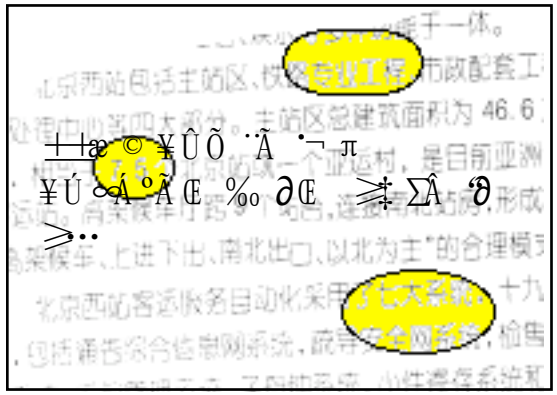
Good with some application domain knowledge!

Where to look?

- Documents
- People
- The product itself
- Prototypes
- Whatever information there may be

Documents which may give you a clue

- Requirements specification (?)
- Change requests
- User manual(s)
- Online help
- Project definition
- Contract
- Data dictionary
- Advertising
- The code (!)
- Standards the product must follow
- The previous product and its documentation
- Competitors' products and their documentation



People to ask questions

Project manager

- Existing documents / baseline?
- Project history?
- A list of project participants?
- Risk with the use of the product?

The chief system designers

- What are the design constraints?

Developers

- What does the product do? Existing test material?
- What objects / entities are being manipulated?
- Design decisions, restrictions?

Users, people who know the intended / previous use

- Users: What do they do with the product? What are the objects used? Existing production data?
- Users' managers: The business and its priorities? Risk with the use?

The product: Explorative testing

Run the product and collect information

- Windows
- Objects
- Menus
- Dialogs
- Decisions /choices
- The way things are implemented
- Functions implying other functions
- Properties implying other properties
- Defects implying other defects

Heuristics (instead of specifications)

Touring Heuristic
Submitted by Mike Kelly on Tue, 20/09/2005 - http://www.testingreflections.com/node/view/2823
I seem to have a winner with my test reporting heuristic. I've used it several times now. It helps me envision and explain my test reporting. I think I will need something similar for application touring.

Here is my attempt:

FCC CUTS VIDS

The mnemonic stands for the following:

Feature tour
Complexity tour
Claims tour

Configuration tour
User tour
Testability tour
Scenario tour

Variability tour
Interoperability tour
Data tour
Structure tour

*** Feature tour: Move through the application and get familiar with all the controls and features you come across.**
*** Complexity tour: Find the five most complex things about the application.**
*** Claims tour: Find all the information in the product that tells you what the product does.**
*** Configuration tour: Attempt to find all the ways you can change settings in the product in a way that the application retains those settings.**
*** User tour: Imagine five users for the product and the information they would want from the product or the major features they would be interested in.**
*** Testability tour: Find all the features you can use as testability features and/or identify tools you have available that you can use to help in your testing.**
*** Scenario tour: Imagine five realistic scenarios for how the users identified in the user tour would use this product.**
*** Variability tour: Look for things you can change in the application - and then you try to change them.**
*** Interoperability tour: What does this application interact with?**
*** Data tour: Identify the major data elements of the application.**
*** Structure tour: Find everything you can about what comprises the physical product (code, interfaces, hardware, files, etc…).**

Example: Result of exploration
A general test requirement.

In a screen form you find that the system gives an error message for an input error and gives a new chance for input.

Include a test requirement for this in every screen form.

The result

- Lists of
  - Documents
  - People / stakeholders
  - Functions
  - Properties
  - Objects
  - existing test data
- Idea of
  - Risk, importance
- Decision
  - What to research in which depth

Step 2: Getting concrete requirements

Translate what you have found to testable requirements.

From unclear to precise.

Many questions to the involved people.

Example

”The product will be in use every working day”

This may mean:

- MTBF > 8 hours (or 12, or 16, or 24 ???)
- Service may be done during nights or weekends (or only very seldom???)
- Maybe some functions don’t need to be up all day?

If you can express a requirement with numbers, or make a series of tests to demonstrate its presence, then it is testable.

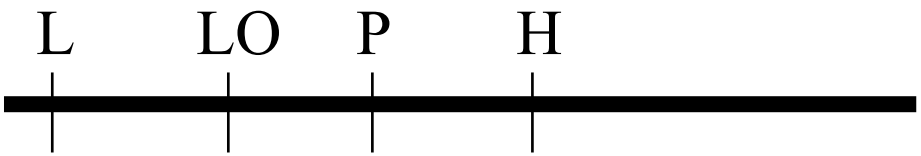
Examples of preliminary (test) requirements and definitions

Users	Administrator	The one who is responsible for the system in use
	"User"	Using the system in his/her daily job
	Hacker	Should not have access
Functions	Reservation handling	Reservations can be registered, changed and followed up
	Booking control	System knows the availability of seats. Overbooking functionality implied.
Objects	Reservation	A series of seats assigned to a passenger
	Seat	A place in a flight on a certain date
	…..	…..
Properties	Reliability	MTBF > 8 hours
	Installability	Can be done by anybody who knows Windows

How to document property requirements

Use Tom Gilbs method:

Four levels:

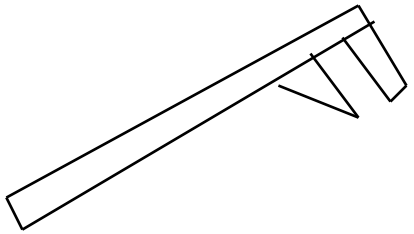


Planned

Lowest acceptable

Highest known (state of the art)

Level in old system



And: How do you intend to measure

If nobody wants to give you numbers?

Give them numbers yourself, far outside acceptable ranges. Discuss.

Example: MTBF = 1 minute?
10 minutes?
1 hour ?
etc.

Still in trouble ?

If nobody can/wants to give you CONCRETE answers, then

either

the requirement is not important,

Don't test it

or

it is badly understood,

Test the area well!

Maybe you have to stop the release.

Ordering this information

For every user group and every object in the system, set up a list of functions and properties.

Add a list of possible trouble / exceptions.

Concrete enough to make test cases.

Reference the source.

Methods to structure test conditions / requirements

- Find conditions in specifications etc.
- Cause effect graphs
- Decision tables
- Graphs
- Data flows
- Business flows
- State transition diagrams
- Entity life history charts
- Use cases

Choose what works for you! Any graphs are better than text!

Example: Decision tables

- Find all conditions and resulting actions for a function
- Describe combinations of conditions
- Discard non-interesting combinations
- Or just test all pairs of inputs
- Manual or tool aided method (CASEMAKER, SQSTest, imbus Testbench, allpairs, Microsoft PICY tool, Classificaiton Tree Editor)

Example: Part of a decision table

Reservation found	N	Y	Y	Y
Flight will go	-	Y	Y	Y
Customer type = gold	-	Y	N	N
Customer type = silver	-	N	Y	N
Customer type = normal	-	N	N	Y
Overbooked	-	-	-	Y
Error msg "Res. not found"	Y	N	N	N
Error msg ”no seat available”		N	N	Y
Make ticket with seat		Y	Y	N
Make ticket standby		N	N	Y
Empty window for next dialog	Y	Y	Y	Y

From decision table to test

- Number of possibilities 2^n (n = # of conditions)
- Safest to test all combinations
- Too expensive
- Minimization:
  - For AND test all YES, and one and one NO
  - For OR test all NO and one and one YES
  - Or use a tool (CASEMAKER, AETG,…)
- Check completeness
  - Are common exceptions and trouble included in the test?

Step 3: Prioritize the test

With the given resources,

- **what to test first**
- **what later**
- **what to cut out**
- **what to test to which depth?**

Which requirements are how important?

- Depends on project objectives
  - Short time to release?
  - Rich functionality / advanced system
  - Special properties (available nowhere else)
  - Few defects
- Importance of single features
  - Risk of problems
    - How often used
    - How defect prone
  - Impact if it doesn't work

Priority
- RISK BASED TESTING

We test what is most important and worst

Important things - see project objectives

Worst things - where defects cost much, where there are lots of defects

- lots of changes
- conflicts
- made under pressure
- made by less qualified personnel
- complex
- many defects in earlier test / review
- etc.

Release criteria

For every test requirement, answer this question:

- **Must it be fulfilled?**
- **Can we live without it? With part of it?**
- **What is the effect if it is not there on release date?**
- **Can we test it within the constraints of this project?**

What is ”good enough”?

The result of the whole

A test plan, with priorities, as a tree structure

- Global test requirements
- User / Customer / stakeholder

Either

  - Functions
  - possible trouble and exceptions
  - Properties

- Object

  - events

Or

  - exceptions, trouble
  - alternatives

- Properties to test globally

Staying out of the critical path

Probably you are involved late. The system is (nearly) ready for testing.

Minimize the time to make test material!

Try to find existing test material!

- ask the developers
- ask for production data from existing systems
- ask potential customers
- standard test suites?

Cut down complicated tests!

Actually you should not be in this situation

The project will probably fail anyway.

Your test may leave large holes, and it is risky.

But sometimes you ARE in this situation and have to survive.

Somehow you ALWAYS need to do this!

Inform management about the RISK!

Literature

(1) Testing in the Dark, by Johanna Rothman and Brian Lawrence, STQE Magazine Vol 1 (1999), issue 2. www.stqemagazine.com

(2) Ståle Amland, Risk based testing, Paper at EuroSTAR 1999. Stale@amland.no

(3) Hans Schaefer, Surviving under time and budget pressure, Proceedings of STAR West 1998. hans.schaefer@ieee.org

(4) "Test Process Improvement", by Martin Pol & Tim Koomen, Addison Wesley, 1999 www.iquip.nl/tpi

(5) James Bach, "Good Enough Quality: Beyond the Buzzword", IEEE Computer, Aug. 1997, pp. 96-98

(6) Tom Gilb, Principles of Software Engineering Management, Addison Wesley, 1988

(7) Ross Collard, Test Design: Developing Test Cases from Use Cases, STQE Magazine Vol 1 (1999), Issue 4. www.stqemagazine.com

(8) Len DiMaggio, Looking under to hood, STQE Magazine 1/2000