Testing in the Dark: What to do with Poor or Missing Specifications

Hans Schaefer
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 software"

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.
Step 1: Collect information

What are the requirements?
Who knows about them?
Where and how much is the risk?
Where to look?

Documents
People
The product itself
Documents which may give you a clue

- Requirements specification
- Change requests
- User manual(s)
- Online help
- Project definition
- Contract
- Data dictionary
- Advertising
- 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 (Create-read-update, search-delete)
- Properties implying other properties
- Defects implying other defects
Example: Result of exploration

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.

You find that you can record an entity. Try to find other functions which search, update, use and delete this entity.
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.
- Object: Reservation
  - Series of seats assigned to a passenger
  - A place in a flight on a certain date

**Properties**
- Reliability: MTBF > 8 hours
- Installability: Can be done by anyone who knows Windows
How to document property requirements

Use Tom Gilb's 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
Manual or tool aided method (SoftTest)
Example: Part of a decision table

<table>
<thead>
<tr>
<th>Reservation found</th>
<th>Flight go</th>
<th>Customer type = gold</th>
<th>Customer type = silver</th>
<th>Customer type = normal</th>
<th>Over booked</th>
<th>Error msg</th>
<th>Make ticket with seat</th>
<th>Make ticket standby</th>
<th>Empty window for next dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>NY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>YY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>YY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>YY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>YY</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Error msg "Reservation not found" NY N N
Error msg "Flight not available" N N Y
Make ticket with seat Y Y N
Make ticket standby N N Y
Empty window for next dialog YYYY
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 (SoftTest, 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

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
  - Functions
  - possible trouble and exceptions
  - Properties

Either

- Object
  - events
  - exceptions, trouble
  - alternatives

Or

- 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.

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. sa@avenir.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


(7) Test Design, Developing test cases from use cases, by Ross Collard, STQE Magazine Vol 1 (1999), issue 4. www.stqemagazine.com
Hans Schaefer