Black Box Software Testing

Spring 2005

REQUIREMENTS ANALYSIS FOR TEST DOCUMENTATION

Cem Kaner, J.D., Ph.D.
Professor of Software Engineering
Florida Institute of Technology
and
James Bach
Principal, Satisfice Inc.

These notes are partially based on research that was supported by NSF Grant EIA-0113539 ITR/SY+PE: "Improving the Education of Software Testers." Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Acknowledgement

- Some of the material in the sections on test documentation is from the Third Los Altos Workshop on Software Testing (LAWST), February 7 and 8, 1998. Kaner founded LAWST and was co-organizer of LAWST 3.
- At LAWST, we discussed test documentation (test planning strategies and materials). The agenda item was:
  - How do we know what test cases we have? How do we know which areas of the program are well covered and which are not?
  - How do we develop this documentation EFFICIENTLY? As many of you know, I despise thick test plans and I begrudge every millisecond that I spend on test case documentation. Unfortunately, some work is necessary. My question is, how little can we get away with, while still preserving the value of our asset?
- The following people attended LAWST 3: Chris Agruss, James Bach, Karla Fisher, David Gelperin, Kenneth Groder, Elisabeth Hendrickson, Doug Hoffman, III, Bob Johnson, Cem Kaner, Brian Lawrence, Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Johanna Rothman, Jane Stepak, Melora Svoboda, Jeremy White, and Rodney Wilson.
- Bret Pettichord contributed to this material as we reviewed / modified it for *Lessons Learned in Software Testing.*

Definitions

- The set of test planning documents might include:
  - Lists or descriptions of *test cases*
  - High-level designs of test cases and *suites* (collections of related tests)
  - Descriptions of the platforms (hardware and software environments) that you will test on, and of relevant variations among the items that make up your platform. Examples of variables are operating system type and version, browser, printer, printer driver, video card/driver, CPU, hard disk capacity, free memory, and third party utility software
  - Descriptions (such as protocol specifications) of the interactions of the software under test (SUT) with other applications that the SUT must interact with. Example: SUT includes a web-based shopping cart, which must obtain credit card authorizations from VISA
  - A *Testing Project Plan*, which identifies classes of tasks and broadly allocates people and resources to them
  - Anything else that you would put in a hard copy or virtual binder that describes the tests you will develop and run
- This set of materials is called the *test plan* or the *test documentation set.*

Basic Documentation Components

- **Lists**:
  - Such as lists of fields, error messages, DLLs
- **Outlines**: organize information into a hierarchy of lists and sublists
  - Such as a function list or an "objectives outline"
- **Tables**: organize information in two dimensions showing relationships between variables
  - Such as boundary tables, decision tables, combination test tables
- **Matrices**: special type of table used for data collection. Rows specify what to test, columns specify the test, and you fill cells in at test time to record results
  - Such as the numeric input field matrix, configuration matrices
- **Models**: visual, verbal, logical or tangible descriptions of the product that the tester might trace through for test ideas
  - Such as UML diagrams, architecture diagrams, state charts

» Refer to Testing Computer Software, pages 217-241.

Outline example: Objectives outline

- Test objectives:
  - Inputs
    - Field-level
      - (list each variable)
    - Group-level
      - (list each interesting combination of variables)
  - Outputs
    - Field-level
      - (list each variable)
    - Group-level
      - (list each interesting combination of variables)

» Based on materials in his SQE's *Systematic Software Testing* course, with thanks to David Gelperin

Objectives outline

- Requirements-based objectives
  - Capability-based (from functional design)
    - Functions or methods including major calculations (and their trigger conditions)
    - Constraints or limits (non-functional requirements)
    - Interfaces to other products
    - Input (validation) and Output conditions at up to 4 levels of aggregation, such as
      - field / icon / action / response message
      - record / message / row / window / print line
      - file / table / screen / report
      - Database
    - Product states and transition paths
    - Behavior rules
      - truth value combinations

Objectives outline

- Design-based objectives
  (resulting from architectural design)
  - Processor and invocation paths
  - Messages and communication paths
  - Internal data conditions
  - Design states
  - Limits and exceptions
- Code-based objectives
  - Control-based
    - Branch-free blocks (i.e. statements)
    - (Top) branches
    - Loop bodies: 0,1, and even
    - Single conditions: LT, EQ, and GT
  - Data-based
    - Set-use pairs
    - Revealing values for calculations

Table Example: Configuration Planning

	Var 1	Var 2	Var 3	Var 4	Var 5
Config 1	V1-1	V2-1	V3-1	V4-1	V5-1
Config 2	V1-2	V2-2	V3-2	V4-2	V5-2
Config 3	V1-3	V2-3	V3-3	V4-3	V5-3
Config 4	V1-4	V2-4	V3-4	V4-4	V5-4
Config 5	V1-5	V2-5	V3-5	V4-5	V5-5
Config 6	V1-6	V2-6	V3-6	V4-6	V5-6

This table defines 6 standard configurations for testing. In later tests, the lab will set up a Config-1 system, a Config-2 system, etc., and will do its compatibility testing on these systems. The variables might be software or hardware choices. For example, Var 1 could be how much RAM on the computer under test (V1-1 is 512 meg, V1-2 is 128 meg., etc.). Var 2 could be the operating system and version, etc.

Matrix Example: Configuration Matrix

	Config 1	Config 2	Config 3	Config 4	Config 5	Config 6
Test 1	Pass	Pass	Pass	Pass	Pass	
Test 2		Fail	Pass	Pass	Pass	
Test 3	Pass	Pass	Pass	Pass	Pass	
Test 4	Pass	Fail	Fail		Pass	
Test 5	Fail	Pass	Fail	Pass	Pass	

This matrix records the results of a series of tests against the 6 standard configurations that we defined in the Configuration Planning Table.

In this table, Config 1 has passed 3 tests, failed 1, and hasn't yet been tested with Test 2. Config 6 is still untested.

IEEE Standard 829 for software test documentation

- Test plan
- Test-design specification
- ***Test-case specification***
  - *Test-case specification identifier*
  - *Test items*
  - *Input specifications*
  - *Output specifications*
  - *Environmental needs*
  - *Special procedural requirements*
  - *Intercase dependencies*
- Test-procedure specification
- Test-item transmittal report
- Test-log

We often see one or more pages per test case.

For a balanced discussion of pro's and cons of IEEE 829, with much more detail than I can provide in this lecture, see Lessons Learned in Software Testing.

Costs

How long does it take to document one test case?

– Productivity of one hour to one day per page of test documentation?
– If a thoroughly documented test case requires one page (or several), how many tester days (tester years) would it take to document 5000 tests?

Test documentation is an expensive product.

Costs of heavyweight approaches

- **High Initial Cost:** What is your documentation cost per test case?
- **High Maintenance Cost:** How many test cases will you have to modify in how many places when a change occurs?
- **Project Inertia**: The more that software design changes cause documentation maintenance costs, the more we'll resist change.
- **Drives Test Design:** Detailed, static documentation favors invariant regression testing. It's challenging to cost effectively introduce variation in test parameters to improve coverage, or to randomly combine features of old tests to improve efficiency of regression tests
- **Discourages Exploration**: How do we document exploratory testing? If we have to document all tests, we strongly discourage exploration.
- **Discourages High Volume Automation:** How many tests can you afford to document?
- **Failing Track Record:** Many teams start to follow 829 but drop it mid-project. By this point, they've filled out all the boilerplate but left the key testing details as TBD (To Be Determined). How does this affect net quality of test documentation and test planning? To legal exposure in the event of a quality-related lawsuit?

Requirements engineering

- Anything that drives or constrains design
- A generally accepted assertion in software engineering is that
  - ***If you don't know what you're trying to build***
    - ***Whatever you build***
      - ***Probably won't be what your customer wanted***
- Prescriptive standards and templates that encourage people to do the same thing in all contexts (rather than carefully tailoring the work to the context) will yield inappropriate products (waste time and money creating the wrong types of documentation).
- One of the challenges in talking about 829 is that on the surface it ***appears*** ***descriptive***, not ***prescriptive***:
  - But it groups concepts together in mandatory ways (such as, "A test plan *shall have* the following structure" with a long list of stuff), and
  - Groups that try to follow it often (usually?) try to include all the categories of information it calls for.

Requirements for test docs?

Before adopting a solution

- *Some specification that says what we will write*

Maybe we should ask:

- Who wants this?
- What will they use it for?
- What do they need?
- How much will it cost?
- Are they (we) willing to spend that much?
- Will the benefits outweigh the costs?
- What other effects will creating this material have on us?

And base our decision about what to write on the requirements we actually determine that we have.

IMPORTANT: You don't have to do all this up front. You can gain information over time and build docs gradually. The goal is responsiveness to stakeholder needs—many will appear or become clear only as the project evolves.

Discovering requirements?

- Ask questions about stakeholders
  - [Stakeholders]
    - Who are the stakeholders?
    - Who would use or be affected by the test documentation?
  - [Interests]
    - What are their interests?
    - (Not what they find interesting--how does this serve their broad objectives?)
  - [Actions]
    - What will they do with the documentation?
  - [Objects]
    - What things do they want? What types of document are of high or low value?
- Ask additional questions:
  - Context-free questions
  - Context-free questions specific to test planning

Stakeholders

- Anyone who would benefit from your system.
- Favored:
  - You want to meet their requirements.
  - Such as your customers or your tech support staff.
- Disfavored
  - You design the product to make it harder for them to meet their requirements.
  - Such as embezzlers (they want security holes in your system and would appreciate getting test documentation that thoroughly describes your system's security weaknesses). Disfavored stakeholders would often succeed through your failure or weakness. Competitors might be disfavored or neutral stakeholders
- Neutral
  - You don't consider their interests in planning or implementing your system.
  - They have no influence on your system.
- Vested or investment-backed
  - They invested in your system and probably have some legal rights in it.
- With influence
  - Your development organization will listen carefully to their opinions

Stakeholder interests

- In the last slide, we asked who we were paying attention to. In this slide, we recognize that even people who are attempting to serve "the best interests of the project" will perceive those differently depending on their situation, including their other interests.
- People and companies have their own objectives, such as getting a raise, a promotion, a new job, saving their marriage, paying debts, etc.
- Your project or product might serve or disserve their interests.
  - For example, a marketing executive might push to broaden the scope of your product because he believes that a product that is perceived as more important or more ambitious will look better on his resume (even if a more objective analysis would predict lower profitability as a consequence).

    Your product might have to adjust to support some people's interests, even if the adjustments would be otherwise seen as boneheaded.

Actions

How will people use the documentation?

- *Actions* might support or interfere with an individual's (or company's) interests.
  - A tech support manager improves speed and quality of support for calls about compatibility with peripherals by using test data from the bug tracking system. She asks for more extensive compatibility testing and more thorough writeups of results.
  - A tech writer asks that function lists be kept up to date because these are useful outlines for the reference manual.
  - A project manager doesn't want detailed project task breakdowns and time estimates in the test plan because these will show the project is far behind schedule before the manager has time to fix the problem or shift the blame.
- *There will be more actions, for more types of documentation, than you can afford to support*

Objects

- What artifacts will you create, and how will these enable the actions that your stakeholders will wish to take?
  - There are many different possible test-related documents.
  - The challenge is to pick the set that
    - You have time to create (and keep as up to date as necessary)
    - Someone will use
    - Serve the overall mission of the testing effort

Test Docs Requirements Questions

- Is test documentation a <u>product</u> or <u>tool</u>?
- Is software quality driven by <u>legal issues</u> or by <u>market forces</u>?
- How quickly is the design changing?
- How quickly does the specification change to reflect design change?
- Is testing approach oriented toward proving <u>conformance</u> to specs or <u>nonconformance</u> with customer expectations?
- Does your testing style rely more on <u>already-defined tests</u> or on <u>exploration</u>?
- Should test docs focus on <u>what</u> to test (<u>objectives</u>) or on <u>how</u> to test for it (<u>procedures</u>)?
- Should the docs ever control the testing project?

Test Docs Requirements Questions

- If the docs control parts of the testing project, should that control come early or late in the project?
- Who are the <u>primary readers</u> of these test documents and how important are they?
- How much <u>traceability</u> do you need? What docs are you tracing back to and who controls them?
- To what extent should test docs support <u>tracking</u> and reporting of <u>project status</u> and <u>testing progress</u>?
- How well should docs support <u>delegation</u> of work to new testers?
- What are your assumptions about the <u>skills and knowledge</u> of new testers?
- Is test doc set a <u>process model</u>, a <u>product model</u>, or a <u>defect finder</u>?

Test Docs Requirements Questions

- A test suite should provide prevention, detection, and prediction. Which is the most important for this project?
- How maintainable are the test docs (and their test cases)? And, how well do they ensure that test changes will follow code changes?
- Will the test docs help us identify (and revise/restructure in face of) a permanent shift in the risk profile of the program?
- Are (should) docs (be) automatically created as a byproduct of the test automation code?

Ultimately, write a mission statement

- Try to describe your core documentation requirements in one sentence that doesn't have more than three components.

The test documentation set will primarily support our efforts to find bugs in this version, to delegate work, and to track status.

Two contrasting missions

The test documentation set will support ongoing product and test maintenance over at least 10 years, will provide training material for new group members, and will create archives suitable for regulatory or litigation use.