

Black Box Software Testing

2004 Academic Edition

PART 10 -- SPECIFICATION-BASED TESTING

by

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.

Kaner & Bach grant permission to make digital or hard copies of this work for personal or classroom use, including use in commercial courses, provided that (a) Copies are not made or distributed outside of classroom use for profit or commercial advantage, (b) Copies bear this notice and full citation on the front page, and if you distribute the work in portions, the notice and citation must appear on the first page of each portion. Abstracting with credit is permitted. The proper citation for this work is "Black Box Software Testing (Course Notes, Academic Version, 2004) www.testingeducation.org", (c) Each page that you use from this work must bear the notice "Copyright (c) Cem Kaner and James Bach, kaner@kaner.com", or if you modify the page, "Modified slide, originally from Cem Kaner and James Bach", and (d) If a substantial portion of a course that you teach is derived from these notes, advertisements of that course should include the statement, "Partially based on materials provided by Cem Kaner and James Bach." To copy otherwise, to republish or post on servers, or to distribute to lists requires prior specific permission and a fee. Request permission to republish from Cem Kaner, kaner@kaner.com.

Specification-Based Testing: Readings

- Gause, D.C. & Weinberg, G.M. (1989) Exploring Requirements: Quality Before Design, Dorset House.
- Kaner, C. (2003) Liability for defective documentation, Conference of the ACM Special Interest Group for the Design of Communications, http://www.testingeducation.org/articles/liability_sigdoc.pdf
- Kaner, C. (1998) Liability for Product Incompatibility. *Software QA*, Vol. 5, #4 (August/September), p. 33. <http://www.kaner.com/pdfs/liability.pdf>
- Spector, Saying One Thing, Meaning Another
- Wilson Software RX: Secrets of Engineering Quality Software

Specification-based testing

- Tag line:
 - “Verify every claim.”
- Fundamental question or goal
 - Check the product’s conformance with every statement in every spec, requirements document, etc.
- Paradigmatic case(s)
 - Testing against contractual specifications
 - Testing dominated by traceability to written specifications
 - User documentation testing

Specifications: Common tasks

- Identify specifications (implicit or explicit)
- Review specifications for
 - Identifiable claims about the product
 - Adequacy / Completeness (it covers the issues)
 - Correctness (it describes the program)
 - Ambiguity
 - Content (not a source of design errors)
 - Testability support
- Create traceability matrices
- Document management (spec versions, file comparison utilities for comparing two spec versions, etc.)
- Participate in review meetings
- Test each claim against the product, report errors

Implicit specifications

- Whatever specs exist
- Software change memos that come with each new internal version of the program
- User manual draft (and previous version's manual)
- Product literature
- Published style guide and UI standards
- Published standards (such as C-language)
- 3rd party product compatibility test suites
- Published regulations
- Internal memos (e.g. project mgr. to engineers, describing the feature definitions)
- Marketing presentations, selling the concept of the product to management
- Bug reports (responses to them)
- Reverse engineer the program.
- Interview people, such as
 - development lead
 - tech writer
 - customer service
 - subject matter experts
 - project manager
- Look at header files, source code, database table definitions
- Specs and bug lists for all 3rd party tools that you use
- Prototypes, and lab notes on the prototypes

Implicit specifications

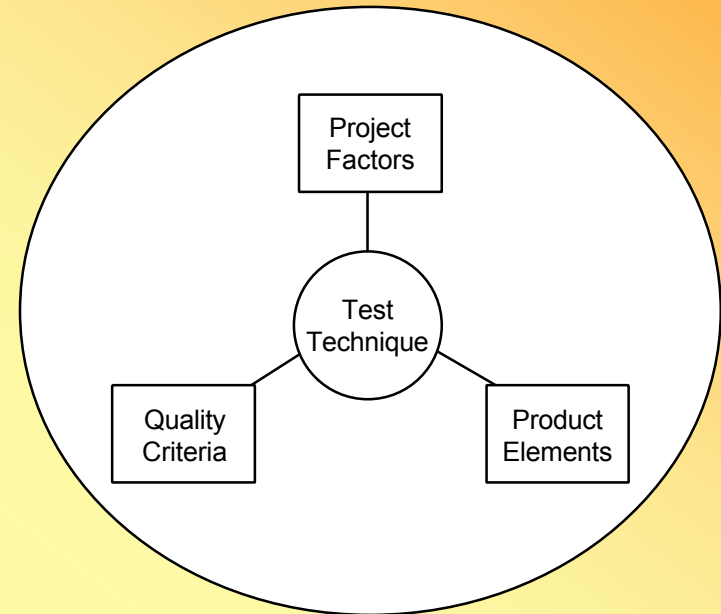
- Interview development staff from the last version.
- Look at customer call records from the previous version. What bugs were found in the field?
- Usability test results
- Beta test results
- Ziff-Davis SOS CD and other tech support CD's, for bugs in your product and common bugs in your niche or on your platform
- BugNet magazine / web site for common bugs
- News Groups, CompuServe Fora, etc., looking for reports of bugs in your product and other products, and for discussions of how some features are supposed (by some) to work.
- Localization guide (probably one that is published, for localizing products on your platform.)
- Get lists of compatible equipment and environments from Marketing (in theory, at least.)
- Look at compatible products, to find their failures (then look for these in your product), how they designed features that you don't understand, and how they explain their design. See listserv's, NEWS, BugNet, etc.
- Exact comparisons with products you emulate
- Content reference materials (e.g. an atlas to check your on-line geography program)

Reviewing a specification

- We can approach a spec from many different angles
 - *What is it saying about the product?*
 - What claims does it make?
 - Do you have enough information to understand the claims?
 - Is some information missing?
 - *Are the claims accurate?*
 - Test the claims against the product.
 - *Would all stakeholders understand the claims in the same way?*
 - Test for ambiguity
 - *Should the specified product be changed?*
 - Review for design errors
 - *Will the specified product be testable?*

Reviewing a specification: Surfacing the claims

- To find and organize the claims, I use an active reading approach based on the Heuristic Test Strategy Model (described in our course notes on Risk Based Testing):
- As you read the spec,
 - Start from the assumption that every sentence in the spec is meant to convey information.
 - Take three writing pads, mark them *Project*, *Product*, and *Quality*.
 - On the appropriate pad, note briefly what the spec tells you about:
 - the project and how it is structured, funded or timed, or
 - the product (what it is and how it works) or
 - the quality criteria that you should evaluate the product against.



Reviewing a specification: Surfacing the claims

- As you note what you *have* discovered, make some additional notes in a different pen color, such as:
 - Items that haven't yet been specified, that you think are relevant.
 - References to later parts of the specification or to other documents that you'll need to understand the spec.
 - Questions that come to mind about how the product works, how the project will be run or what quality criteria are in play.
 - Your disagreements or concerns with the product / project as specified.
- Beware of getting too detailed in this. If the spec provides a piece of information, you don't need to rewrite it. Just write down a pointer (and a spec page number). Your list is a quick summary that you build as you read, to help you read, not a rewriting of the document.
- As you read further, some of your earlier questions will be answered. Others won't. Ask the programmers or spec writers about them.

Exercise

- Pretend that you are a member of the test team for Star Office. Here are some (fictitious) details:
 - The code is developed remotely (much of it is developed by programmers in another country).
 - You have access to the source code and many bug reports from users because the program is open source.
 - This is a significant update to the program.
 - Lots of bugs will be fixed.
 - The program will read/write Office 2002 files

Exercise (Alternate version)

- Pretend that you are a member of the test team for Star Office. Here are some (fictitious) details:
 - The code is developed remotely (much of it is developed by programmers in another country).
 - You have access to the source code and many bug reports from users because the program is open source.
 - This is a significant update to the program.
 - Lots of bugs will be fixed.
 - The word processor has been substantially revised so that it interprets and formats bullets, numbered lists, and table layouts more compatibly with MS Word. This is in response to customer complaints that the formatting is often incorrect or inconsistent when one creates a Word file or a StarOffice document with these features, and then use StarOffice to read a Word file or Word to read an exported StarOffice file.

Exercise (continued)

- This is a group exercise. Divide the 4 categories of factors among yourselves:
 - Project factors
 - Product elements
 - Quality criteria
 - Risks (for the relevant list, see the Risk-based testing section (section 15) earlier in the course)
- If possible, work in pairs at flipcharts.
- Divide the category you are working on into its subcategories and work on one at a time (maybe allocate one flipchart page each).
 - For example, Stakeholders, Processes, and Staff are subcategories in the Project Factors list.

Exercise (Continued)

- On your flipcharts,
 - list what you know for each subcategory that you are working on.
 - In many subcategories, you won't know much. Along with writing down what you do know, write down questions that call for relevant information that you don't yet know.
 - You might learn that some tasks or information or decisions are time-critical. You need to do them or get them right away. If so, write them on a separate sheet or in a different bright color and start to deal with them before the end of today.
- After working in separated pairs for a while, come back and compare notes.
- These notes provide you with a first draft task list, and a lot of information about the project, more than you'd expect.

Example: Understanding what a specification means

- Important to trace from requirements to implications
- Outline of an exercise
 - Give a list of questions
 - Examples are
 - the test documentation requirements questions (in the test documentation section) and
 - the automation maintainability questions at (www.kaner.com/pdfs/shelfwar.pdf)
 - For each question
 - Ask the students to name at least two decisions that they would make on the basis of the answer to the question
 - Make this a small group exercise, splitting up the questions, groups fill flipcharts, then bring back to full class.

Testing claims against the product

Uniform Commercial Code Article 2 (2003 revision)

SECTION 2-313A. (2) If a seller in a record packaged with or accompanying the goods makes an affirmation of fact or promise that relates to the goods, provides a description that relates to the goods, or makes a remedial promise, and the seller reasonably expects the record to be, and the record is, furnished to the remote purchaser, the seller has an obligation to the remote purchaser that:

- (a) the goods will conform to the affirmation of fact, promise or description unless a reasonable person in the position of the remote purchaser would not believe that the affirmation of fact, promise or description created an obligation; and
- (b) the seller will perform the remedial promise.

(3) It is not necessary to the creation of an obligation under this section that the seller use formal words such as “warrant” or “guarantee” or that the seller have a specific intention to undertake an obligation, but an affirmation merely of the value of the goods or a statement purporting to be merely the seller's opinion or commendation of the goods does not create an obligation.

Testing a claim against the product

- Important to understand the level of generality called for when testing a spec item. For example, imagine a field X:
 - We could test a single use of X
 - Or we could partition possible values of X and test boundary values
 - Or we could test X in various scenarios
 - Which is the right one?

Ambiguity analysis

- Many sources of ambiguity in software design and development.
 - In wording or interpretation of specifications or standards
 - In expected response of the program to invalid or unusual input
 - In behavior of undocumented features
 - In conduct and standards of regulators / auditors
 - In customers' interpretation of their needs and the needs of the users they represent
 - In definitions of compatibility among 3rd party products
- Whenever there is ambiguity, there is a strong opportunity for a defect (at least in the eyes of anyone who understands the world differently from the implementation).
 - Richard Bender teaches this well. If you can't take his course, you can find notes based on his work in Rodney Wilson's *Software RX: Secrets of Engineering Quality Software*
 - An interesting workbook: Cecile Spector, *Saying One Thing, Meaning Another*

Ambiguity: Break statements into elements

- Make / read a statement about the program
- Work through the statement one word at a time, asking what each word means or implies.
 - *Thinkertoys* describes this as “slice and dice.”
 - Gause & Weinberg develop related approaches as
 - “Mary had a little lamb” (read the statement several times, emphasizing a different word each time and asking what the statement means, read that way)
 - “Mary conned the trader” (for each word in the statement, substitute a wide range of synonyms and review the resulting meaning of the statement.)
- These approaches can help you ferret out ambiguity in the definition of the product. By seeing how different people could interpret a key statement (e.g. spec statement that defines part of the product), you can imagine new tests to check which meaning is operative in the program.

Break statements into elements: Example

Quality is value to some person

– Quality

•

•

•

– Value

•

•

•

– Some

•

•

•

– Person

• *Who is this person?*

• *How are you the agent for this person?*

• *How are you going to find out what this person wants?*

• *How will you report results back to this person?*

• *How will you take action if this person is mentally absent?*

Traceability matrix

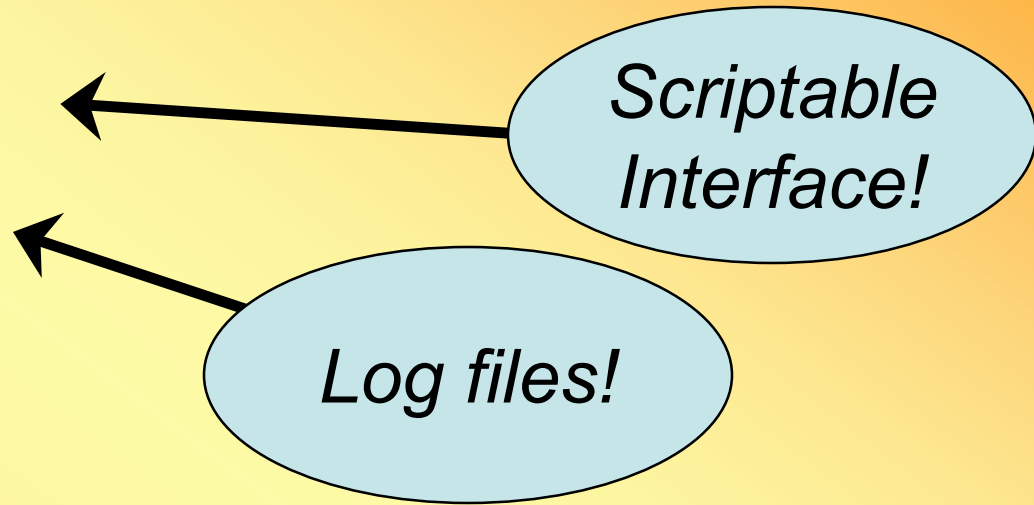
	Var 1	Var 2	Var 3	Var 4	Var 5
Test 1	X	X	X		
Test 2		X		X	
Test 3	X		X	X	
Test 4			X	X	
Test 5				X	X
Test 6	X				X

Traceability matrix

- The columns involve different test items. A test item might be a function, a variable, an assertion in a specification or requirements document, a device that must be tested, any item that must be shown to have been tested.
- The rows are test cases.
- The cells show which test case tests which items.
- If a feature changes, you can quickly see which tests must be reanalyzed, probably rewritten.
- In general, you can trace back from a given item of interest to the tests that cover it.
- This doesn't specify the tests, it merely maps their coverage.

Design reviews: Testability

- Controllability
- Observability
- Availability
- Simplicity
- Stability
- Information
- Separation of functional components
- Availability of oracles



**Testing is far more rapid
when the product is far more testable**

Sample Exam Questions

- Describe a traceability matrix.
 - How would you build a traceability matrix for Open Office's word processor?
 - What is the traceability matrix used for?
 - What are the advantages and risks associated with driving your testing using a traceability matrix?
 - Give examples of advantages and risks that you would expect to deal with if you used a traceability matrix for any two of the following features of Open Office Writer:
 - Outlines
 - Tables
 - Fonts
 - Printing

Summary: Specification-driven testing

- Strengths
 - Critical defense against warranty claims, fraud charges, loss of credibility with customers.
 - Effective for managing scope / expectations of regulatory-driven testing
 - Reduces support costs / customer complaints by ensuring that no false or misleading representations are made to customers.
- Blind spots
 - Any issues not in the specs or treated badly in the specs / documentation.