Black Box Software Testing

Spring 2005

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.

Spec-based testing?

- We've seen at least three different meanings:
  - **A style of testing (collection of test-related activities and techniques) focused on discovering what claims are being made in the specifications and on testing them against the product.**

    This is what we mean by spec-based testing.

  - A style of testing focused on proving that the statements in a specification (and the code that matches the statements) are logically correct.
  - A set of test techniques focused on logical relationships among variables that are often detailed in specifications.

Context factors

- Is this *intended* as an authoritative document? Who is its champion?
- Who cares if it's kept up to date and correct? Who doesn't?
- Who is accountable for its accuracy and maintenance?
- What are the corporate consequences if it is inaccurate?

Why did they write the specification?

- Enforceable contract for custom software?
- Facilitate and record agreement among stakeholders? About specific issues or about the whole thing?
- Vision document?
- Support material for testing / tech support / technical writers?
- Marketing support?
- Sales or marketing communication?
- Regulatory compliance?

Context factors

- To what extent is a test against the spec necessary, sufficient, or useful?
- To what extent can you change the product or process via spec review / critique?
- Will people invest in your developing an ability to understand the spec?

Why are you reviewing the spec or testing the product against the specification?

- Contract-related risk management?
- Regulatory-related risk management?
- Development group wants to use the spec as an internal authoritative standard?
- Learn about the product?
- Prevent problems before they are coded in?
- Identify testing issues before you get code?
- Help company assess product drift?
- It's a source of information—test tool to help you find potential bugs? (in product or spec?)

Spec testing issues

- What **is** the specification?
- What does the specification say?
- Critiquing the specification (what it says):
  - How it says what it says
  - What it says about the product
  - What it says about the testing of the product
- Critiquing the specification (doing the critique)
- Driving tests from the specification
- Legal issues

What is *the specification?*

- What is **a** specification?
  - For our purposes, we include any document that describes the product and drives development, sale, support, or the purchasing of the product.
- What is the scope of **this** specification?
  - Some specs cover the entire product, others describe only part of it (such as error handling).
  - Some specs address the product from multiple points of view, others only from one point of view.
- Do we have the **right** specification?
  - Right version?
  - Source control?
  - Do we verify version?
    - File compares?

What is *the specification?*

- Is this a stable specification?
  - Is it under change control?
    - Should it be?
- Supplementary information assumed by the specification writer
  - Some aspects of the product are unspecified because they are defined among the staff, perhaps in some other (uncirculated?) document
- Implicit specifications
  - Some aspects of the product are unspecified because there are controlling cultural or technical norms.
  - These are particularly important
    - Rather than making an unsupported statement that "It's bad" (e.g. "users won't like it"), you can justify your assertions

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
- 3rd party tech support databases, magazines and web sites with reports of bugs in your product, common bugs in your niche or on your platform and for discussions of how some features are supposed (by some) to work.
- Localization guide (probably 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 listservs, websites, etc.
- Exact comparisons with products you emulate
- Content reference materials (e.g. an atlas to check your on-line geography program)

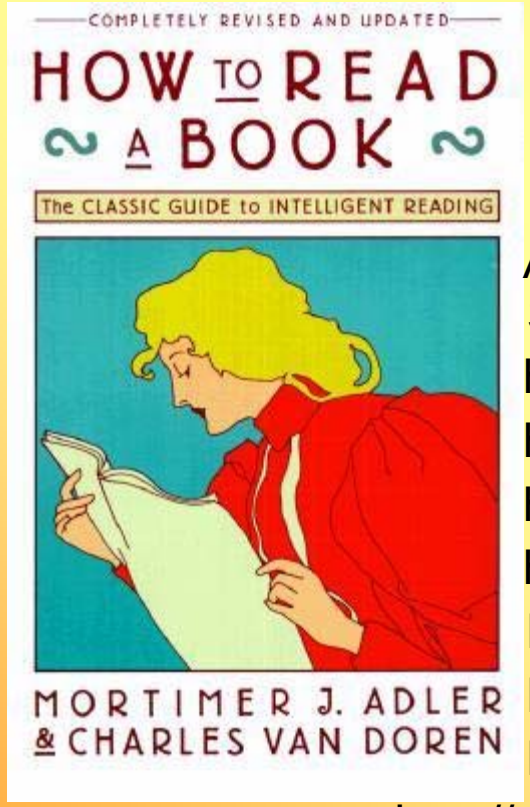
Spec testing issues

- What is the specification?
- What does the specification say?
- Critiquing the specification (what it says):
  - How it says what it says
  - What it says about the product
  - What it says about the testing of the product
- Critiquing the specification (doing the critique)
- Driving tests from the specification
- Legal issues

What does the spec say?

- Much of what is written about specification analysis has to do with the specification-in-the-small—interpreting the fine details in one or two pages of text
  - These are useful skills
  - But specifications are often one or two *thousand* pages (or more)
    - spread across multiple documents
    - which incorporate several other documents by reference
    - using undefined, inconsistently defined or idiosynchratically defined vocabulary
- Specification readers often suffer severe information overload.
- Active reading skills and strategies are essential for effective specification analysis

Basics of active reading



Adler, M.J. and van Doren, C. (1972) How to Read a Book. Simon and Schuster, New York, NY.

http://radicalacademy.com/adlermarkabook.htm

http://www.justreadnow.com/strategies/active.htm

http://www.somers.k12.ny.us/intranet/reading/PLAN.html

http://istudy.psu.edu/FirstYearModules/Reading/Materials.html

http://www.mindtools.com/pages/article/newISS_04.htm

http://www.clt.astate.edu/bdoyle/TextbookRdng.ppt

http://titan.iwu.edu/~writcent/Active_Reading.htm

http://www.itrc.ucf.edu/forpd/strategies/stratCubing.html

http://www.ncrel.org/sdrs/areas/issues/students/learning/lr2befor.htm

There's an excellent table of *Skills Good Readers Use* at http://www.forpd.ucf.edu/content/lesson8/lesson8topic6.htm

Active reading

- **Prioritize what you read, by**
  - **Surveying (read table of contents, headings, abstracts)**
  - **Skimming (read quickly, for overall sense of the material)**
  - **Scanning (seek specific words or phrases)**
- **Search for information in the material you read, by**
  - **Asking information-gathering questions and search for their answers**
  - **Creating categories for information and read to fill in the categories**
  - **Questioning / challenging / probing what you're reading**
- **Organize it**
  - **Read with a pen in your hand**
  - **If you underline or highlight, don't do so until AFTER you've read the section**
  - **Make notes as you go**
    - Key points, Action items, Questions, Themes, Organizing principles
  - **Use concise codes in your notes (especially on the book or article). Make up 4 or 5 of your own codes. These 2 are common, general-purpose:**
    - ? means I have a question about this
    - ! means new or interesting idea
  - **Spot patterns and make connections**
    - Create information maps
  - **Relate new knowledge to old knowledge**
- **Plan for your retention of the material**
  - **SQ3R (survey / question / read / recite / review)**
  - **Archival notes**

Active Reading: Cubing

Cubing involves attacking a problem from 6 perspectives. Originally developed as a writing strategy, it's often suggested for active reading as well.

For the feature or concept that you are trying to understand:

1. **Describe it:** describe its physical attributes (size, shape, etc.) and its functional attributes;
2. **Compare it:** What's it similar to? Why do you think so?
3. **Associate it:** What other ideas, products, etc. does it bring to mind?
4. **Analyze it:** Break it down into its components. How are they related? How do they work together?
5. **Apply it:** What can you (or the user) do with it?
6. **Evaluate it:** Take a stand. List reasons that it is good (good feature, good implementation, good design, good idea, etc.) or bad. If you want to be neutral, make two lists—one of all the ways that it's good, the other of all the ways that it's bad.

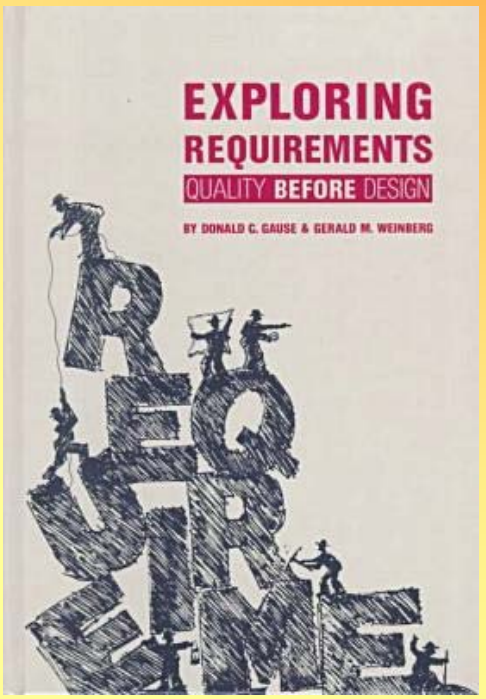
As you develop your cube, work through the specification (and any other documents you have) to collect the information you need to do these tasks.

http://www.itrc.ucf.edu/forpd/strategies/stratCubing.html
http://www.uhv.edu/ac/research/prewrite/generateideas.pdf
http://www.humboldt.edu/~tdd2/Cubing.htm

Asking questions

Here are some key contrasts:

- **Hypothetical** *(what would happen if …)* **vs. behavioral** *(what have you done / what has happened in the past in response to …)*
- **Factual** *(factual answers can be proved true or false)* **vs. opinion** *(what is the author's—or your–interpretation of these facts.)*
- **Historical** *(what happened already)* **vs. predictive** *(what the author—or you—expect to happen in the future under these conditions)*
- **Open** *(calls for an explanatory or descriptive answer; doesn't reveal the answer in the question)* **vs. closed** *(calls for a specific true answer, often answerable yes or no)*
- **Context-dependent** *(the question is based on the specific details of the current situation)* **vs. context-free** *(the question is usable in a wide range of situations—it asks about the situation but was written independently of it).*



Gause / Weinberg is a superb source for context-free questions

More questions

- **Causal** *(Why did this happen? Why is the author saying that?)*
- **Ask for evidence** *(What proof is provided? Why should you believe this?)*
- **Evidentiary sufficiency** *(Is this conclusion adequately justified by these data?)*
- **Trustworthiness of the data** *(Were the data collection and analysis methods valid and reliable?)*
- **Critical awareness** *(What are the author's assumptions? What are your assumptions in interpreting this?)*
- **Clarification** *(What does this mean? Is it restated elsewhere in a clearer way?)*
- **Comparison** *(How is this similar to that?) and* **Contrast** *(How is this different from that?)*
- **Implications** *(If X is true, does that mean that Y must also be true?)*
- **Affective** *(How does the author (or you) feel about that?)*
- **Relational** *(How does this concept, theme or idea relate to that one?)*
- **Problem-solving** *(How does this solve that problem, or help you solve it?*

More questions

- **Relevance** *(Why is this here? What place does it have in the message or package of information the author is trying to convey? If it is not obviously relevant, is it a distractor?)*
- **Author's comprehension** *(Does the author understand this? Is the author writing in a way that suggests s/he is inventing a concept without having researched it?)*
- **Author credibility** *(What basis do you have for believing the author knows what s/he is talking about?)*
- **Author perspective / bias** *(What point of view is the author writing from? What benefit could the author gain from persuading you that X is true or desirable (or false, etc.)?)*

- *The Michigan Educational Assessment Association has some useful material at* http://www.meap.org/html/TT_QuestionTypes.htm

More questions

- **Application** *(How can you apply what the author is saying? How does the author apply it?)*
- **Analysis** *(Can you (does the author) break down an argument or concept into smaller pieces?)*
- **Synthesis** *(Does the author (or can you) bring together several facts, ideas, concepts into a coherent larger concept or a pattern?)*

(More along these lines come from Bloom's taxonomy…)

The classic context-free questions

Traditional news reporters' questions:

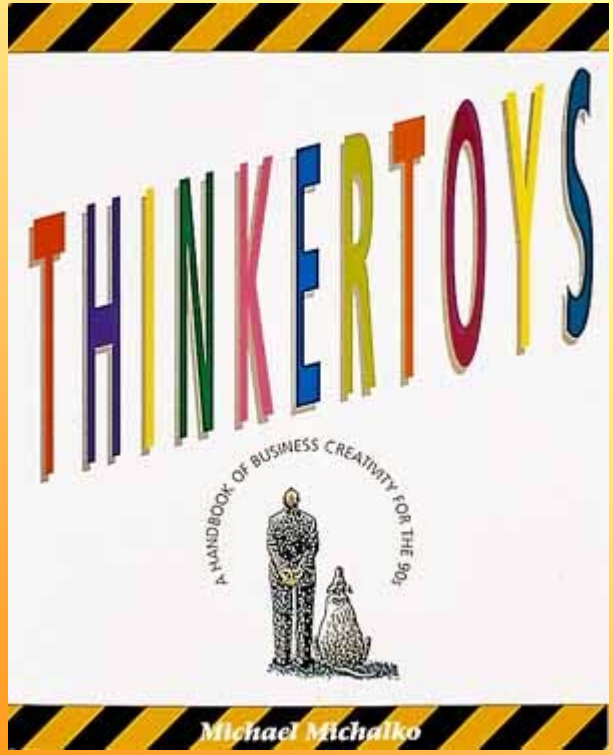
- **Who?**
- **What?**
- **When?**
- **Where?**
- **How?**
- **Why?**

For example, *Who will use this feature? What does this user want to do with it? Who else will use it? Why? Who will choose not to use it? What do they lose? What else does this user want to do in conjunction with this feature? Who is not allowed to use this product or feature, why, and what security is in place to prevent them?*

We use these in conjunction with questions that come out of the testing model (see below). The model gives us a starting place. We expand it by asking each of these questions as a follow-up to the initial question.

Using context-free questions to define a problem

- Why is it necessary to solve the problem?
- What benefits will you receive by solving the problem?
- What is the unknown?
- What is it that you don't yet understand?
- What is the information that you have?



- What is the source of this problem? (Specs? Field experience? An individual stakeholder's preference?)
- Who are the stakeholders?
- How does it relate to which stakeholders?
- What isn't the problem?
- Is the information sufficient? Or is it insufficient? Or redundant? Or contradictory?
- Should you draw a diagram of the problem? A figure?

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

Using context-free questions to define a problem

- Where are the boundaries of the problem?
- What product elements does it apply to?
- How does this problem relate to the quality criteria?
- Can you separate the various parts of the problem? Can you write them down? What are the relationships of the parts of the problem?
- What are the constants (things that can't be changed) of the problem?
- What are your critical assumptions about this problem?
- Have you seen this problem before?
- Have you seen this problem in a slightly different form?
- Do you know a related problem?
- Think of a familiar problem having the same or a similar unknown.
- Suppose you find a problem related to yours that has already been solved. Can you use it? Can you use its method?
- Can you restate your problem? How many different ways can you restate it? More general? More specific? Can the rules be changed?
- What are the best, worst, and most probable cases you can imagine?

Using context-free questions to evaluate a plan

- Will this solve the whole problem? Part of the problem?
- What would you like the resolution to be? Can you picture it?
- How much of the unknown can you determine?
- What reference data are you using (if any)?
- What product output will you evaluate?
- How will you do the evaluation?
- Can you derive something useful from the information you have?
- Have you used all the information?
- Have you taken into account all essential notions in the problem?
- Can you separate the steps in the problem-solving process? Can you determine the correctness of each step?
- What creative thinking techniques can you use to generate ideas? How many different techniques?
- Can you see the result? How many different kinds of results can you see?
- How many different ways have you tried to solve the problem?

Based on: *The CIA's Phoenix Checklists (Thinkertoys, p. 140) and Bach's Evaluation Strategies (Rapid Testing Course notes)*

Using context-free questions to evaluate a plan

- What have others done?
- Can you intuit the solution? Can you check the results?
- What should be done?
- How should it be done?
- Where should it be done?
- When should it be done?
- Who should do it?
- What do you need to do at this time?
- Who will be responsible for what?
- Can you use this problem to solve some other problem?
- What is the unique set of qualities that makes this problem what it is and none other?
- What milestones can best mark your progress?
- How will you know when you are successful?
- How conclusive and specific is your answer?

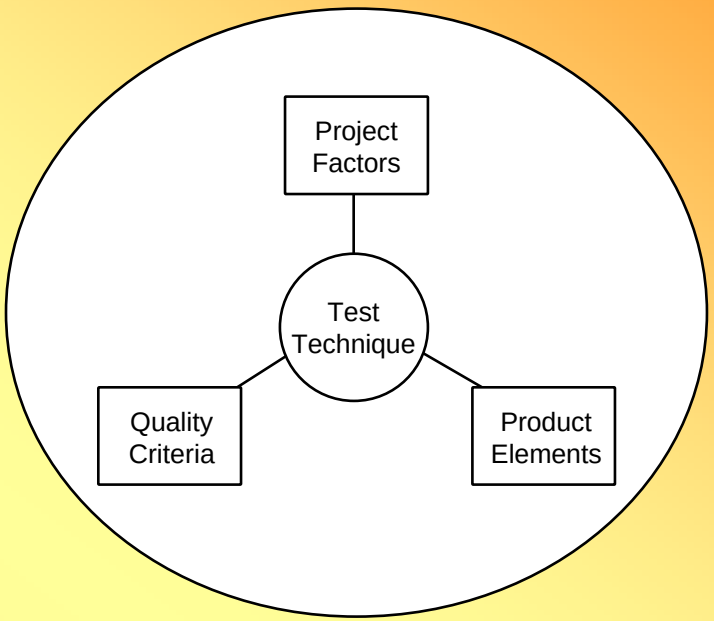
Context-Free Questions

- **Context-free process questions**
  - Who is the client?
  - What is a successful solution worth to this client?
  - What is the real (underlying) reason for wanting to solve this problem?
  - Who can help solve the problem?
  - How much time is available to solve the problem?
- **Context-free product questions**
  - What problems could this product create?
  - What kind of precision is required / desired for this product?
- **Metaquestions (when interviewing someone for info)**
  - Am I asking too many questions?
  - Do my questions seem relevant?
  - Are you the right person to answer these questions?
  - Is there anyone else who can provide additional information?
  - Is there anything else I should be asking?
  - Is there anything you want to ask me?
  - May I return to you with more questions later?

***A sample of additional questions based on Gause & Weinberg's* Exploring Requirements p. 59-64**

An active reading example

- To find and organize the claims, I use an active reading approach based on the Heuristic Test Strategy Model
- As you read the spec,
  - Start from the assumption that every sentence in the spec is meant to convey information.
  - Take four writing pads, mark them *Project*, *Product*, *Quality* and *To-Do*.



  - 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 you should evaluate the product against or
    - things you need to do, that you learned from the spec.

An active reading example

- As you note what you *have* discovered, make 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.

Spec testing issues

- What is the specification?
- What does the specification say?
- **Critiquing the specification (what it says):**
  - How it says what it says
  - What it says about the product
  - What it says about the testing of the product
- Critiquing the specification (doing the critique)
- Driving tests from the specification
- Legal issues

How it says what it says

- Ambiguity
  - Are multiple interpretations possible? Likely?
- Adequacy
  - Does it provide enough information for programming, documentation and testing?
- Completeness
  - To what extent does it cover the
    - Feature set
    - Use cases
    - Usage scenarios
    - Test-relevant information (such as boundaries, error handling, etc.)

Ambiguity analysis

- Many sources of ambiguity in software design & 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

  - Richard Bender teaches this well in his courses on Requirements Based Testing. His course has some great labs, and he coaches well. I recommend it. 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. She discusses and provides examples and exercises with many additional ambiguities in common English than I can cover here.

Common ambiguities in use of the language

- Undefined words
  - "The user may authenticate incoming documents by processing their security attributes."
- Incorrectly used words
  - *Typeface* refers to a set of characters having the same design, or to the design. *Font* refers to a specific size and style of a typeface. (See google: *define typeface* and *define font*.) A version of OpenOffice labeled a list of typefaces as fonts and a list of styles (italics, bold, etc.) as typefaces. How would you interpret help documentation that referred to "typefaces" ?
- Contradictorily defined words
  - Use "valid" to mean (sometimes) a value considered valid by a user and (other times) a value that meets input criteria constraints in a program.
- Vague words
  - Etc., will display a message, process, upgrade, performance, user friendly
- Commonly misunderstood words
  - *i.e.* (means *id est* = *that is* and calls for a restatement or redefinition of a previous word or statement) whereas *e.g.* means *exampli gratia* (for example)
- Ambiguous quantities
  - Within, between, up to, almost, on the order of
- Impossible promises
  - "The program will be fully tested." "Performance will be instantaneous."

Common ambiguities
Logical conditions

- Incomplete set of logical conditions
  - If A and B then C. If A and not B then D
    - What about B and not A?
- Logical operators ambiguously grouped
  - If A and B or C then D
    - Is this (A and B) or C? Is it A and (B or C)?
    - Just because precedence orders are defined by convention doesn't mean that the spec author, the spec reviewers, and the programmers know them
- Negation without explicit specification of scope
  - If not A and B then D
    - Is this (Not A) and B? Is it Not (A and B)? Is it Not-A and Not-B?
- There are plenty more of these. Look at any logic text.

Common ambiguities: Missing facts (1)

- Unspecified decision maker
  - If X is unacceptable, then
    - Unacceptable according to who?
- Assumes facts not specified
  - Spec assumes the reader is familiar with the specifics of regulations, environmental constraints, etc. These might change or differ across countries, platforms, etc.
- Ambiguity in time
  - Does X have to precede Y? In the statement, "Do A if X happens and Y happens and Z happens" does it matter if they happen in that order?
- Causes without effects
  - The case X is greater than Y will trigger special processing
- Effects without causes
  - If X occurs during processing, then …
- Effects with underspecified causes
  - General protection fault

Common ambiguities: Missing facts (2)

- Unspecified error handling
  - "The program will accept up to 3 names."
- Unspecified variables
  - The program will set a flag if this happens.
    - What flag?
- Boundaries unspecified or underspecified
  - Is 0 a positive number? If 0<x<100 is valid, how big is the maximum value that you will allow to be copied into X for evaluation?
    - (Whittaker's testing approach rests on programmers being blind to a wide range of unspecified system or program constraints)
- Unspecified quantities
  - The program will compare the value input for X to the maximum allowed
- Mentioned but undefined cases
  - "The page format dialog will display 3 column width fields at a time. The user may not specify more than 10 columns."

Ambiguity analysis: Break statements into elements

- Gause & Weinberg
  - "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 statement's resulting meaning.)
- "Slice & dice" (Thinkertoys)
  - Make / read a statement about the program. Work through the statement one word at a time, asking what each word means or implies.
- These approaches can help you ferret out ambiguity in the product definition. By seeing how different people can interpret a key statement in the spec, you can imagine new tests to check which meaning is operative in the program.

Break statements into elements:

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?*

What it says about the product

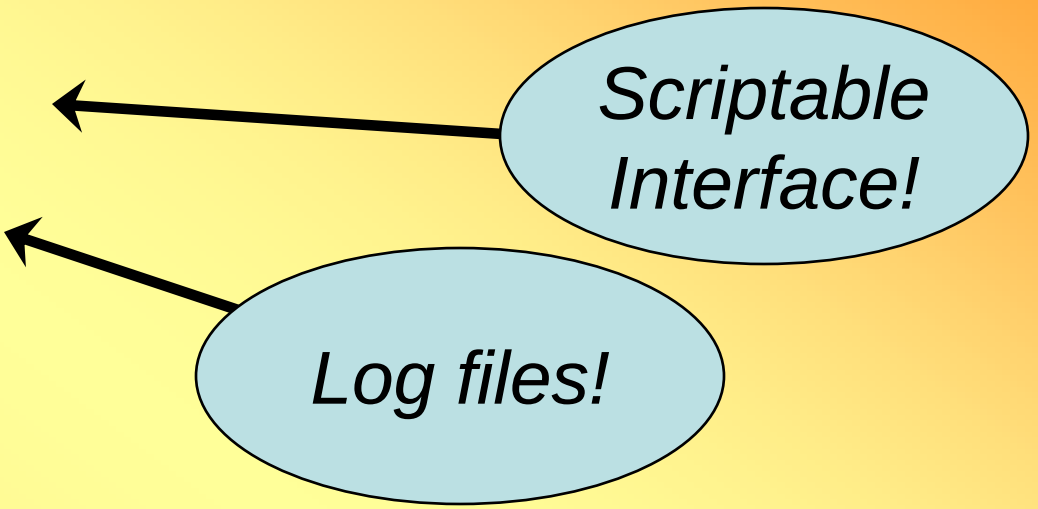
- Correctness
  - Does it accurately describe the program?
- Controversy
  - Which parts are controversial? Who are the stakeholders who disagree and why do they disagree?
- Adequacy
  - Does it provide enough information for programming, documentation and testing?
- Completeness
  - Does it cover the feature set?
- Design
  - Can you tell whether it specifies design errors?
  - Is it understandable, usable, trainable, consistent, appropriate for the market?
  - Does it set up the program / programmer for common errors?

What it says about testing

- Early in the project, you can review the spec's implications for testing, and change them or prepare for them.
  - Implications for test design
    - What test techniques will be most appropriate for this project?
    - Will you need additional training or tools for them?
    - Are there ways to simplify (or otherwise change) to product in ways that would call for simpler or cheaper or more easily structured techniques?
    - How much exploring will this project require?
      - Does your staff have the knowledge, skills and connections?
  - Test schedule and resource commitments / implications
    - When will you receive deliverables from others?
    - When are you to deliver your work?
    - What do you need to get this done?
    - Are any of your commitments unreasonable?
  - Testability support

Design reviews: Testability

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



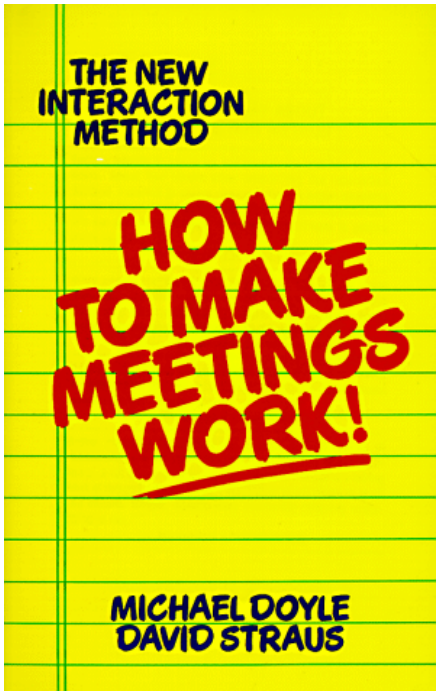
Testing is far more rapid when the product is far more testable

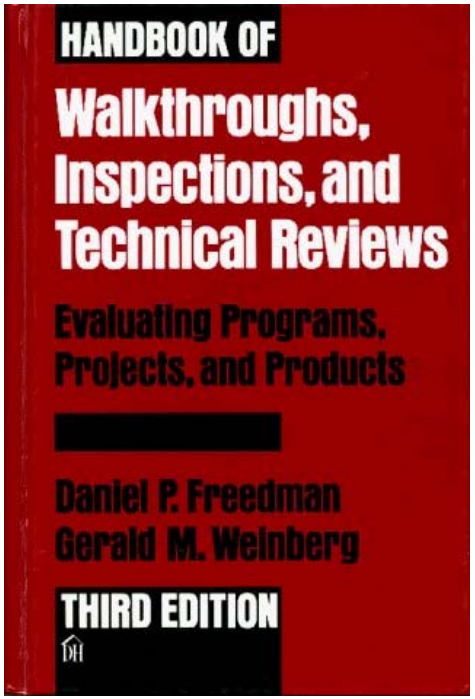
Testing the program against the spec

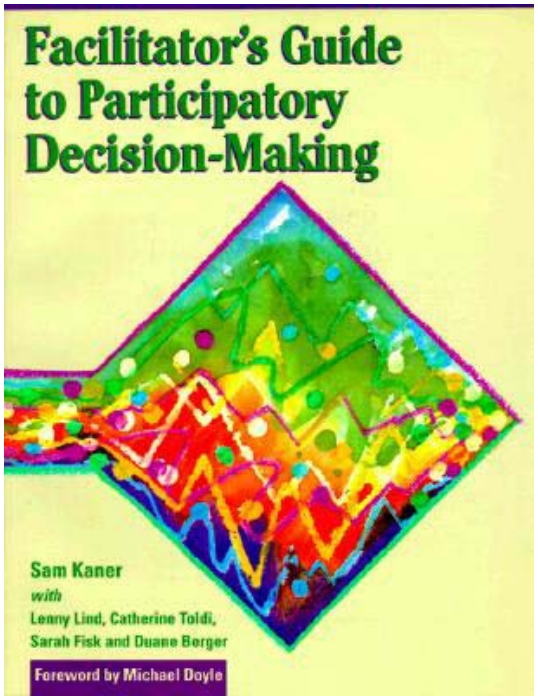
- What is the specification?
- What does the specification say?
- Critiquing the specification (what it says):
  - How it says what it says
  - What it says about the product
  - What it says about the testing of the product
- **Critiquing the specification (doing the critique)**
- Driving tests from the specification
- Legal issues

Critiquing specs: Process notes

- Review meetings
  - Test groups often train to facilitate technical reviews
- Detailed comments on the specification
  - Same guidelines as for critiquing other tech pubs. See *Testing Computer Software*







Spec testing issues

- What is the specification?
- What does the specification say?
- Critiquing the specification (what it says):
  - How it says what it says
  - What it says about the product
  - What it says about the testing of the product
- Critiquing the specification (doing the critique)
- **Driving tests from the specification**
- Legal issues

Driving tests from the specification

- Who are the stakeholders?
  - There are stakeholders for all services. Who are yours?
    - Regulators? Marketing? End customer?
    - Journalists? Attorney? Court? (Expert witness?)
    - Client company (you're the outsource test lab)?
  - These stakeholders would have different test-result / test-documentation expectations from the typical project team.
- What is a good specification driven test?
  - Same as "what is a good test?"
  - But tests come from specs
  - Might be that a test that covers several spec items is preferred to a single-item test
  - Might be that tests that resolve or expose and show implications of specification ambiguities are particularly important

Driving tests from the specification

- Coverage
  - Key issue is coverage of the specification
    - Cover items (individual statements)
      - But how many tests per statement do you need?
      - Many groups require only one per spec assertion
    - Cover specified relationships
      - To test `A && B`
      - You probably want to test at least `A true and B true`
      - `A true and B false`
      - `A false and B true`

Driving tests from the spec: Coverage

- 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?
  - This partially depends on whether specification-driven testing is your exclusive style of testing
- How do we track coverage?
  - Trace tests BACK TO the specification with traceability matrices

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
Totals	2	2	3	4	1

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.
- Traceability tool risk—test case mgmt tools can drive you into wasteful over-documentation and unmaintainable repetition

Spec testing issues

- What is the specification?
- What does the specification say?
- Critiquing the specification (what it says):
  - How it says what it says
  - What it says about the product
  - What it says about the testing of the product
- Critiquing the specification (doing the critique)
- Driving tests from the specification
- **Legal issues**

Legal issues

- Warranties based on claims to the public
  - Article: *Liability for defective documentation*

    http://www.kaner.com/pdfs/liability_sigdoc.pdf
- Warranties based on claims to custom-product customer
- Claims of compatibility with other products
  - Article: *Liability for product incompatibility*

    http://www.kaner.com/pdfs/liability_sigdoc.pdf
- Errors in your product documents, that are not about your products
  - Article: *Liability for defective content*

    http://www.kaner.com/pdfs/sigdocContent.pdf

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.

Exercise (Version 1)

- 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 2003 files

Exercise (Version 2)

- Pretend that you are a member of the test team for OpenOffice. 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 an OpenOffice document with these features, and then use OpenOffice to read a Word file or Word to read an exported OpenOffice 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.

Summary

- Objectives
  - Check the product's conformance with every statement in every spec, requirements document, etc.
  - The specific objectives are context-dependent
    - Why did they write the spec?
    - Why are you testing against it?
    - Who cares about your results?
- Paradigmatic case(s)
  - Testing against contractual specifications
  - Testing dominated by traceability to written specifications
  - User documentation testing
- Strengths
  - Critical defense against warranty claims, fraud charges
  - Effective for managing scope / expectations of regulatory-driven testing
  - Reduces support costs / customer complaints by reducing risk of misrepresentations to customers.
- Risks
  - Issues not in the specs or treated badly in the specs / documentation.
  - Focus is on coverage, not risk
  - Test case management tools that lock onto traceability may harm your efficiency