Test Documentation

Thoughts From The

3rd LOS ALTOS WORKSHOP
ON SOFTWARE TESTING

(Los Altos, CA, February 1998)

Cem Kaner

J.D., Ph.D.

Test Documentation: Preliminary Note

On February 7 and 8, the Third Los Altos Workshop on Software Testing 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, KarlaX Fisher, David Gelperin, Kenneth Groder, Elisabeth Hendrickson, Doug Hoffman, III, Bob Johnson, Cem Kaner, Brian Lawrence, Thomas Lindemuth, Brian Marick, Thanga Meenakshi, Noel Nyman, Jeffery E. Payne, Bret Pettichord, Johanna Rothman, Jane Stepak, Jeremy White, and Rodney Wilson.

(Well, I should say that I think those people attended. I'm embarrassed to say that I got on the plane and realized that I didn't have a final attendance list with me. I'm pretty sure that this was the full crowd.)

We came up with a lot of ideas. The material listed here is only a subset. I'm still thinking about the range and implications of our discussion. This is my first public talk on LAWST since the meeting, and my first real attempt to sort out some of the good ideas in a way that will be useful to people outside of the LAWST group. I hope you'll tolerate some disorganization--this is thinking in progress. And, your comments, criticisms, and *virtual* tomatoes are most welcome.

NOTICE:

This talk does not necessarily reflect the views of each of the LAWST 3 attendees. Nor is it a comprehensive layout of the material we discussed at LAWST 3.

Test Documentation: Fine Print

NOTICE

This talk does not necessarily reflect the views of each of the LAWST 3 attendees. Nor is it a comprehensive layout of the material we discussed at LAWST 3.

This communication should not be interpreted as legal advice or a legal opinion. The transmission of this communication does not create an attorney-client relationship between me and you. Do not act or rely upon law-related information in this communication without seeking the advice of an attorney. Finally, nothing in this message should be interpreted as a "digital signature" or "electronic signature" that can create binding commercial transactions.

Any advice given in this communication should be taken with a grain of salt. Don't believe everything that you read. Your mileage may vary. Each dealer negotiates its own prices. Please keep your hands in the car at all times. Do not tap on glass. Do not eat anything that has been on the floor for more than 10 seconds. Power tools are not an effective cure for headaches. If this were an actual emergency, this broadcast would be followed by official information and instructions. Keep your hands to yourself. Do not point. Please do not feed the animals. High Voltage, keep out! Contents under pressure, may explode. Not to be taken internally. Wait at least 1/2 hour after eating before using this material.

This material does not reflect the thoughts or opinions of either myself, my company, my friends, or my cat. Don't quote me on that; don't quote me on anything. All rights reserved. Copyright © 1998--Permanent or transitory reproductions of this material are not allowed. If you even THINK about this page without my written permission, you owe me $1 for unauthorized copying.

By reading this page, you agree that this material is provided to you "as is" without warranty of any kind, express or implied. All responsibility for its use rests with you. In addition, you agree that your remedy for defects in this material is limited to the price you paid for it, less $25 per page for handling expense. You expressly agree that this limitation of liability is applicable even if I have been notified of the existence of defects in this material. In no event will I be liable to you even for a known defect that causes you predictable harm.

These pages are subject to change without notice. Bugs may be slightly enlarged to show detail. Any resemblance to actual persons, living or dead, is unintentional and purely coincidental. Hand wash only, tumble dry on low heat.

Do not bend, fold, mutilate, or spindle. No substitutions allowed. For a limited time only. This material is void where prohibited, taxed, or otherwise restricted. Equal opportunity employer. No shoes, no shirt, no bugs, no paycheque. Quantities are limited while supplies last; or until the ship date (which will come first). If defects are discovered, do not attempt to fix them yourself. Return to an authorized service cente where they will be suitably deferred. Caveat emptor, caveat testor, caveat litigator. Read at your own risk. Parental advisory - explicit lyrics; text may contain material some readers may find objectionable, parental guidance is advised. Keep away from sunlight, pets, and small children. Limit one-per-family please.

No money down; no purchase necessary; you need not be present to win. Some assembly required. Batteries are not included. Action figures sold separately. No preservatives added; safety goggles may be required during use. Sealed for your protection, do not use if the safety seal is broken. Call before you dig.

For external use only. If a rash, redness, irritation, or swelling develops, quit reading the code. Use only with proper ventilation. Avoid extreme temperatures and store in a cool dry place. Keep away from open flames and avoid inhaling fumes. Avoid contact with mucous membranes. Do not puncture, incinerate, or store above 120 degrees Fahrenheit. Do not place near flammable or magnetic source.

Smoking these pages may be hazardous to your health and may be a felony under your state's laws. The best safeguard, second only to abstinence, is the use of a good laugh. Text used in these materials is made from 100% recycled electrons and magnetic particles; no animals were used to test these materials. No salt, MSG, artificial color or flavor added. If ingested, do not induce vomiting. If symptoms persist, consult your test manager.

Slippery when wet. Must be 18 to enter. Possible penalties for early withdrawal. Offer valid only at participating E-mail sites, slightly higher west of the Rockies. Allow four to six weeks for delivery. Disclaimer does not cover hurricane, lightning, tornado, tsunami, volcanic eruption, earthquake, flood, and other Acts of God, misuse, neglect, unauthorized repair, damage from improper installation, broken antenna or marred cabinet, incorrect line voltage, software errors, missing or altered serial numbers, sonic boom vibrations, electromagnetic radiation from nuclear blasts, customer adjustments that are not covered in the joke list, and incidents owing to airplane crash, ship sinking, motor vehicle accidents, leaky roof, broken glass, falling rocks, mud slides, forest fire, unwanted children, flying projectiles, or dropping the item. Other restrictions may apply.

Test Documentation: Overview

1. **Terminology**
2. **Some Common Mistakes**
3. **Requirements for Test Documentation**
4. **A Few Good Techniques**
5. **Notes on Development of a Documentation Strategy**
6. **Notes on Group Processes for Developing and Reviewing Test Documentation**

Test Documentation: Terminology

David Gelperin urged us to rethink our use of the word "test plan" because it is overused, and too often used to mean different things. Here are the words I'll probably use going forward:

- » **Test Case**
- » **Test Suite**
- » **Test Objective**
- » **Test Strategy**
- » **Test Design**
- » **Testing Project Plan (aka test plan)**
- » **Test Documentation (aka test plan)**

Test Documentation: Some Common Mistakes

Let's not spend much time on these. I'd rather focus on what works. But here are blunders that you've probably encountered:

» **Death by Detail**

- ***Myth of perfectly reproducible test cases***
- ***Brainless paperwork: reduced productivity and creativity***
- ***Overemphasis on simple tests rather than harsh ones***

» **No Detail**

- ***How do you tell that the program has failed a test case?***
- ***Comfort without justification***

» **Ancestor Worship**

- ***There is value in inspecting ancestral test cases, in measuring their code coverage, and in using them as a mine for insights.***
- ***Beware of false positives and false negatives.***
- ***Beware of undocumented test cases.***
- ***Reverse engineering, rewriting, and the 10% rule.***

» **No Source Control**

» **Mis-set Management Expectations**

- ***Managers may think that the documented tests are 100% of the tests.***
- ***Managers may think that all test cases will/should be documented in detail.***
- ***Managers may think that every test case that appears on a test document should be run.***

Test Plan Requirements: Contrasting Objectives

- ***Is the test documentation set a <u>product</u> or a <u>tool</u>?***

- ***Is it a <u>process model</u>, a <u>product model</u>, or a <u>defect finder</u>?***

- ***Is your software quality driven by <u>legal considerations</u> or by <u>market forces</u>?***

- ***How much <u>traceability</u> do you need? What docs are you tracing back to and who controls them?***

Test Plan Requirements: Contrasting Objectives -2

- ***Is your testing approach primarily oriented toward <u>conformance to specs or other written criteria</u> or toward proving <u>nonconformance with customer expectations</u>?***

- ***Does your preferred testing style rely on <u>already-defined tests</u> (regression) or <u>exploration</u>?***

- ***Should test docs focus on <u>what</u> to test (objectives) or on <u>how</u> to test for it (procedures)?***

Test Plan Requirements: Contrasting Objectives -3

- ***Should detailed control of the project by the test plan should come <u>early</u>, <u>late</u>, or <u>never</u>?***
- ***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?***
- ***Who are the <u>primary readers</u> of these test documents and how important are they?***

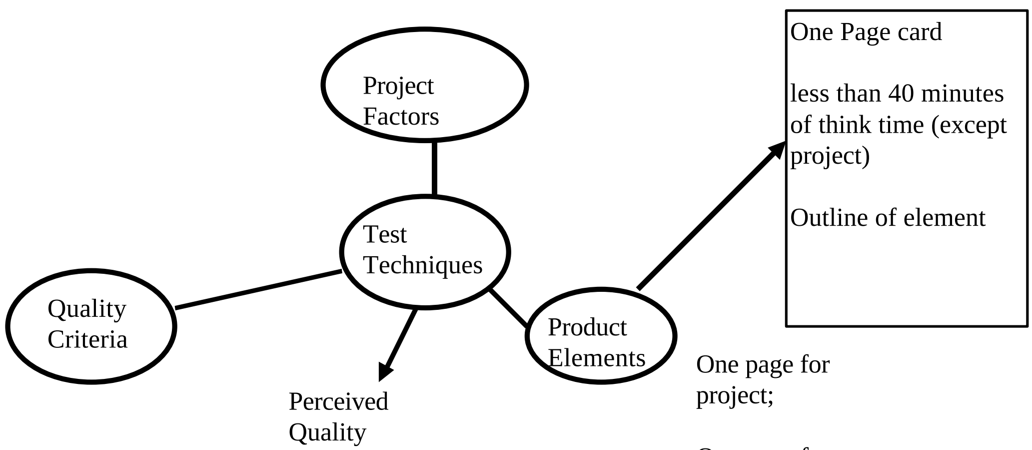
Test Plan Requirements: Contrasting Objectives -4

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

Test Documentation: A Few Good Techniques

- **Tripos-based description of test objectives (Bach)**
- **Boundary and equivalence analysis (Myers)**
- **Reusable test matrix (Nguyen, Kaner)**
- **Automated reusable test matrix (Hendrickson)**
- **Objectives list (Gelperin)**
- **Multi-variable test combination chart (Gelperin)**
- **Data relationship chart (Kaner)**

Test Documentation: Tripos



One page for project;

One page for element;

One page per test

(Content reviewed; elements not linked; need to write down and ask questions; one page insisted upon unless compelling argument for violation)

TRIPOS Model [Bach, STLabs]

Test Docs: Boundary & Equivalence Analysis

Two tests belong to the same *equivalence class* if you expect the same result (pass / fail) of each. Testing multiple members of the same equivalence class is, by definition, redundant testing.

Boundaries **mark the point or zone of transition from one equivalence class to another. The program is more likely to fail at a boundary, so these are the best members of (simple, numeric) equivalence classes to use.**

Note how the boundary case has two ways to fail. It can fail because the program's treatment of the equivalence class is broken OR because the programmer's treatment of inequalities is broken.

More generally, you look to subdivide a space of possible tests into relatively few classes and to run a few cases of each. You'd like to pick the most powerful tests from each class.

Boundary Analysis Table

Variable	Equivalence Classes	Boundaries and Special Cases	Notes
First number	-99 to 99 > 99 < -99 non-number expressions	99, 100 -99, -100 / ; 0 null entry	max bounds min bounds ASCII bounds, next section 0 always interesting
Second number	same as first	same	same
Sum	-198 to 198		Are there other sources of data for this variable? Ways to feed it bad data?

The simplest way to build a boundary analysis over time is to put the information that you gather into a table.

The table should eventually contain all variables. This means, all input variables, all output variables, and any intermediate variables that you can somehow observe.

In constructing this table, you might well just LIST all (or many) of the variables first, filling in information about them as you obtain it.

Boundary Table as a Test Plan Component

- Makes the reasoning obvious.
- Makes the relationships between test cases fairly obvious.
- Expected results are pretty obvious.
- Several tests on one page.
- Can delegate it and have tester check off what was done. Provides some limited opportunity for tracking.
- Not much room for status.

Question, now that we have the table, do we have to do all the tests? What about doing them all each time (each cycle of testing)?

Partitioning

In theory, the key to partitioning is dividing the space into mutually exclusive subsets. Each subset is an equivalence class. This is very nice in theory, but let's look at printers.

LaserJet II compatible printers

- Big class
- HP II original was weak in graphic-complexity related error handling but strong in paper handling. Depending on the kind of risk we're testing against, we might or might not choose this printer as the exemplar of the class.

Partitioning

Device compatibility testing illustrates a multidimensional space with imperfect divisions between classes and with several different failure risks. The key to success is to remember that partitioning is merely a sampling strategy. The goal is to work from a rational basis in order to select a few valuable representatives from a much larger population of potential tests.

If you can think of different ways that the program can fail in its interaction with a device (such as a printer), then FOR EACH TYPE OF ERROR, you look for the specific device (model, version of printer) that is most likely to confound the program.

From an equivalence class of “LaserJet II compatibles” you get several different, uniquely powerful, class representatives.

A strong sampling strategy rests on our knowledge of the world, not just of the spec.

Examples from a Class Exercise: Equivalence Class and Boundary Brainstorm

There are many types of variables, including input variables, output variables, internal variables, hardware and system software configurations, and equipment states. Any of these can be subject to equivalence class analysis. Here are some common results from the class brainstorms:

- ranges of numbers
- character codes
- how many times something is done
  - » (e.g. shareware limits on the number of uses of the software)
  - » (e.g. how many times you can do it before you run out of memory)
- how many records in a database, how many names in a mailing list, how many variables in a spreadsheet, how many bookmarks, how many abbreviations
- size of the sum of variables, or the size of some other computed value (think binary and think digits)
- size of a number that you enter (number of digits) or size of a character string
- size of a concatenated string
- size of a path specification
- size of a file name
- size (in characters) of a document
- size of a file (note special values such as exactly 64K, exactly 512 bytes, etc.)
- size of a document on a page, in terms of the memory requirements for the page. This might just be in terms of resolution x page size, but it may be more complex if we have compression algorithms
- size of the document on the page (compared to page margins) (across different page margins, page sizes)
- equivalent output events (such as printing documents)
- amount of available memory (> 128 meg, > 640K, etc.)
- visual resolution, size of screen, number of colors
- operating system version
- variations within a group of “compatible” printers, sound cards, modems, etc.
- equivalent event times (when something happens)
- timing: how long between event A and event B (and in which order--races)
- length of time after a timeout (from JUST before to way after) -- what events are important?
- speed of data entry (time between keystrokes, menus, etc.)
- speed of input -- handling of concurrent events
- number of devices connected / active
- system resources consumed / available (also, handles, stack space, etc.)
- date (year 2000-related boundaries) and time (23:59; end of week, end of month)
- transitions between algorithms (optimizations) (different ways to compute a function)
- most recent event, first event
- input or output intensity (voltage)
- speed / extent of voltage transition (e.g. from very soft to very loud sound)

Refer to Testing Computer Software, pages 7-11, 126-133, 399-401

Using Test Matrices to Simplify Partitioning

After testing a simple numeric input field a few times, you've learned the drill. The boundary chart is reasonably easy to fill out for this, but it wastes your time.

Use a test matrix to show/track a series of test cases that are essentially the same.

- For example, for most input fields, you'll do a series of the same tests, checking how the field handles boundaries, unexpected characters, function keys, etc.
- As another example, for most files, you'll run essentially the same tests on file handling.

The matrix is a concise way of showing the repeating tests.

- Put the objects that you're testing on the rows.
- Show the tests on the columns.
- Check off the tests that you actually completed in the cells.

Reusable Test Matrices

Test Matrix for a Numeric Input Field

Additional Instructions:

		Nothing	Valid value	At LB of value	At UB of value	At LB of value - 1	At UB of value + 1	Outside of LB of value	Outside of UB of value	0	Negative	At LB number of digits or chars	At UB number of digits or chars	Empty field (clear the default value)	Outside of UB number of digits or chars	Non-digits	Wrong data type (e.g. decimal into integer)	Expressions	Space	Non-printing char (e.g., Ctrl+char)	DOS filename reserved chars (e.g., "\ * . :")	Upper ASCII (128-254)	Upper case chars	Lower case chars	Modifiers (e.g., Ctrl, Alt, Shift-Ctrl, etc.)	Function key (F2, F3, F4, etc.)

Matrices

- You can often re-use a matrix like this across products and projects.
- You can create matrices like this for a wide range of problems. Whenever you can specify multiple tests to be done on one class of object, and you expect to test several such objects, you can put the multiple tests on the matrix.
- Mark a cell blue if you ran the test and the program passed it. Mark the cell read if the program failed.
- Write the bug number of the bug report for this bug.
- Write (in the cell) the automation number or identifier if the test case has been automated.

Matrices

Problems?

- **What if your thinking gets out of date? (What if this program poses new issues, not covered by the standard tests?)**
- **Do you need to execute every test every time? (or ever?)**
- **What if the automation ID number changes? -- We still have a maintenance problem but it is not as obscure.**
- **This still supports exploration.**

Automated Reusable Test Matrices

Walk back through the Numeric Input matrix, but from the point of view of automating it. Depending on your automation tool, the following script should be reasonably easy:

- Identify the variable
- Input the range
- Input the main out-of-range error message
- Have the script walk the program against all of the tests for this variable, checking for valid results or out-of-range errors.

In theory, you should be able to set this up as a real-time input tool. (Look at a variable, specify it, test it.)

Objectives list

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 examples in Gelperin's Systematic Software Testing course.)*

Objectives list

Requirements-based Objectives

- Capability-based (resulting 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
    - *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*

- *(Based on examples in Gelperin's Systematic Software Testing course.)*

Objectives list

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
    - *(Based on examples in Gelperin's Systematic Software Testing course.)*

Combination Chart

Cases	*Variable 1*	*Variable 2*	*Variable 3*	*Testing Issue 1*	*Testing Issue 2*	*Testing Issue 3*

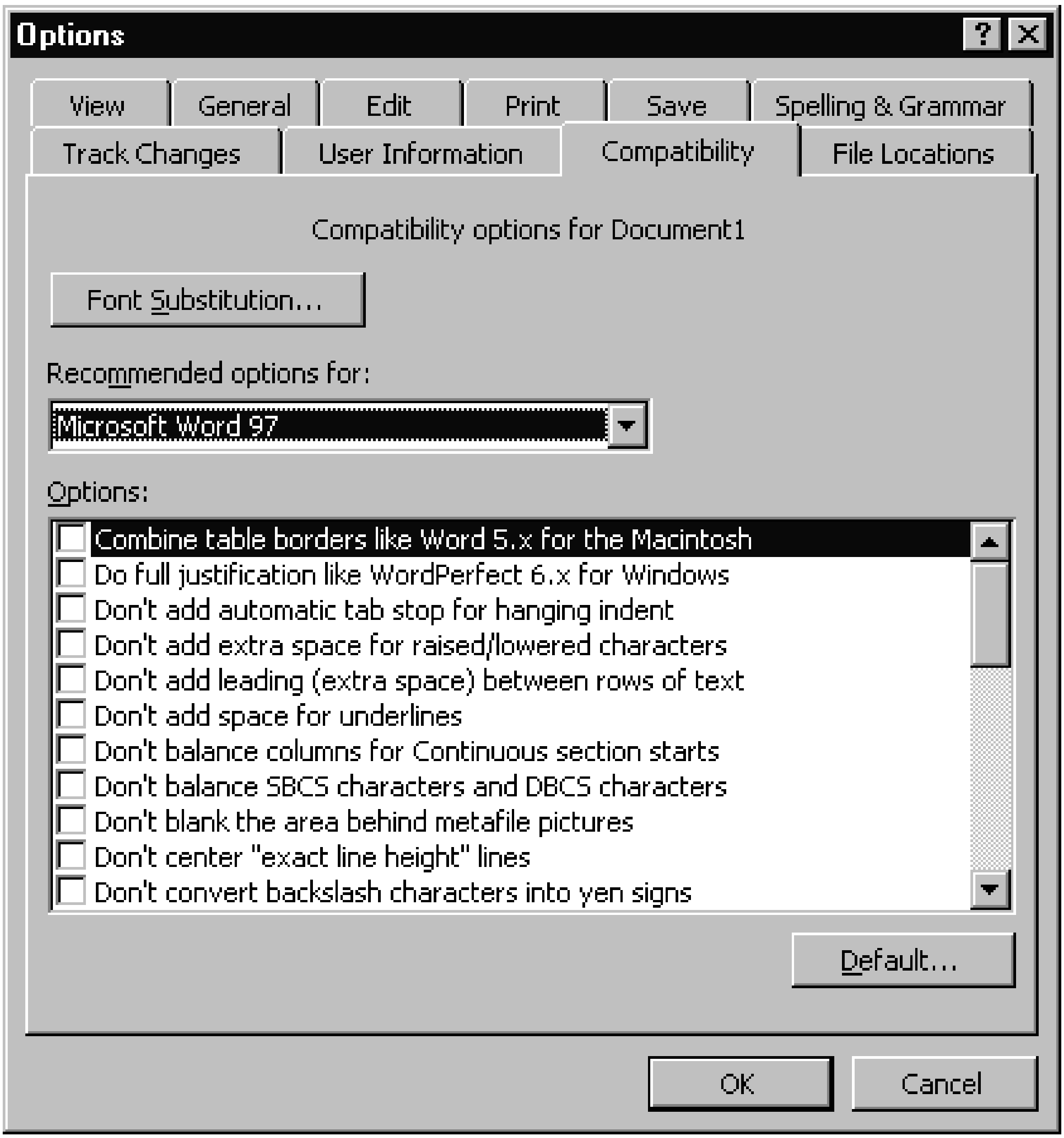
The idea here is that there are a few variables that you will test together, in order to look at a joint effect. “Testing Issues” might include some underlying variable that you want to test, or some output that you want to manipulate, or some other event that is determined by the combination of variables, not by any one of them alone.

Each row is a test case.

The variables’ entries are typically actual values.

The “testing issues” entries are the values or events you are trying to manipulate or observe.

Complex Data Relationships



A Tabular Format for Data Relationships

Field	*Entry Source*	*Display*	*Print*	*Related Variable*	*Relationship*
Start Date				End Date	Constraint to a range
End Date				Start Date	Constraint to a range

Once you identify two variables that are related, test them together using boundary values of each or pairs of values that will trigger some other boundary.

This is not the most powerful process for looking at relationships. An approach like Cause-Effect Graphing is more powerful, if you have a perfect specification.

I started using this chart as an ***exploratory*** tool for simplifying my look at relationships in overwhelmingly complex programs. (There doesn't have to be a lot of complexity to be "overwhelming.")

A Tabular Format for Data Relationships

THE TABLE'S FIELDS

Field: ***Create a row for each field (`Consultant`, `End Date`, and `Start Date` are examples of fields.)***

Entry Source: ***What dialog boxes can you use to enter data into this field? Can you import data into this field? Can data be calculated into this field? List every way to fill the field -- every screen, etc.***

Display: ***List every dialog box, error message window, etc., that can display the value of this field. When you re-enter a value into this field, will the new entry show up in each screen that displays the field? (Not always -- sometimes the program makes local copies of variables and fails to update them.)***

Print: ***List all the reports that print the value of this field (and any other functions that print the value).***

Related to:: ***List every variable that is related to this variable. (What if you enter a legal value into this variable, then change the value of a constraining variable to something that is incompatible with this variable's value?)***

Relationship: ***Identify the relationship to the related variable.***

A Tabular Format for Data Relationships

Many relationships among data:

- **Independence**
  - » Varying one has no effect on the value or permissible values of the other.
- **Causal determination**
  - » By changing the value of one, we determine the value of the other.
  - » For example, in MS Word, the extent of shading of an area depends on the object selected. The shading differs depending on Table vs. Paragraph.
- **Constrained to a range**
  - » For example, the width of a line has to be less than the width of the page.
  - » In a date field, the permissible dates are determined by the month (and the year, if February).
- **Selection of rules**
  - » For example, hyphenation rules depend on the language you choose.

A Tabular Format for Data Relationships

- **Logical selection from a list**
  - » **processes the value you entered and then figures out what value to use for the next variable. Example: timeouts in phone dialing:**
    - ***0 on complete call 555-1212 but 95551212?***
    - ***10 on ambiguous completion, 955-5121***
    - ***30 seconds incomplete   555-121***
- **Logical selection *of* a list:**
  - » **For example, in printer setup, choose:**
    - ***OfficeJet, get Graphics Quality, Paper Type, and Color Options***
    - ***LaserJet 4, get Economode, Resolution, and Half-toning.***
- **Look at Marick for discussion of catalogs of tests for data relationships.**

Data Relationship Table

Looking at the Word options, you see the real value of the data relationships table. Many of these options have a lot of repercussions.

You might analyze all of the details of all of the relationships later, but for now, it is challenging just to find out what all the relationships ARE.

The table guides exploration and will surface a lot of bugs.

PROBLEM

Works great for this release. Next release, what is your support for more exploration?

About Cem Kaner

I focus on the satisfaction and safety of customers and workers. This cuts across several academic and technical disciplines. To develop competence in the field, I've worked in several related areas:

Law (J.D., 1993). Currently in a small solo practice that provides direct legal and retained expert services. Public service includes prosecution (3 months full-time volunteer, Santa Clara County, Deputy DA); grievance officer and contract advisor for the National Writers Union; consumer complaint investigator / mediator (Santa Clara County Dept. of Consumer Affairs); and Board of Directors, Northern California Hemophilia Foundation. I am deeply involved in the drafting of Article 2B of the Uniform Commercial Code (a new law that will govern all contracts for software) and laws governing digital signatures.

Quality (Certified by ASQC in Quality Engineering, 1992). I served as an Examiner for the California Quality Awards in 1994 and 1995.

Experimental Psychology (Ph.D, 1984: perceptual measurement, cognition, physiological foundations). Continuing education in human factors / ergonomics.

Mathematics & Philosophy (B.A., Arts & Sciences, 1974).

Technical Communication (courses at UC Santa Cruz Extension). I published *Testing Computer Software* in 1988 and the 2nd edition (with Hung Nguyen and Jack Falk) in 1993. It received the *Award of Excellence* in the Society for Technical Communication's *Northern California Technical Publications Competition*. I've managed three tech pubs groups, and my staff have won several STC awards. I've published about 50 papers and am currently working on three new books: *Testing Computer Software* (3rd Edition); *Bad Software: Consumer Protection for Computer Software*; and *Good Enough Testing for Good Enough Software.*

Organization Development (courses from Community at Work, plus experience as an Associate, then Senior Associate at Psylomar -- Organization Development)

Computing. I first studied FORTRAN in 1967 (many other languages later). In 1970-73, I learned valuable lessons the hard way about human factors, reliability, and real world requirements via a failing service-bureau-based computerization of my family's retail businesses. I began doing my own work with computers in 1976, while a Psychology graduate student. We used them as real time lab control systems, simulators, and data analyzers. Interested in the human side of the machines, my colleagues and I explored ways to improve software usability and overall system reliability. In 1983, I moved to Silicon Valley. Since then I've worked in the Valley as a human factors analyst (user interface designer), programmer, test manager, technical publications manager, software development manager, middle manager (director), and (my current role) independent consultant.