

Black Box Software Testing

2004 Academic Edition

PART 4 -- DOMAIN TESTING 2

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.

Exercise

For each of the following,

- List the variable(s) of interest.
- List the valid and invalid classes.
- List the boundary value test cases.
- Lay out the results in a boundary table.

1. FoodVan delivers groceries to customers who order food over the Net. To decide whether to buy more vans, FV tracks the number of customers who call for a van. A clerk enters the number of calls into a database each day. Based on previous experience, the database is set to challenge (ask, “Are you sure?”) any number greater than 400 calls.

2. FoodVan schedules drivers one day in advance. To be eligible for an assignment, a driver must have special permission or she must have driven within 30 days of the shift she will be assigned to.

Introduction

- As you just saw in the last example, one of the underlying risks addressed by domain testing is ambiguity. Interpretation of the specification is often most difficult for the boundary cases. This is one of the key reasons that we test equivalence classes at their boundaries rather than at random “equivalent” points inside the set. (Read Hamlet & Taylor, 1988; Ostrand & Balcer, 1988.)
- There are four fundamentally different ways to think about domain testing. (Read Kaner, 2004)
- Here, we will develop the risk-management foundation for domain analysis. (Read Weyuker & Jeng, 1991 for an example of this analysis.)
- One of the reasons that the oversimplified, mechanical views of domain testing have lasted so long is that courses often consider the simple cases and stop, moving on to something else.
- We’ll introduce the multidimensional complexities here, studying them in more detail in the next Section.
- In this Section, we start by attacking the notion that a domain must be linear (Clarke et al, 1982), considering domains that can’t be ordered from small to large. (Kaner et. al, 1993 on printer testing, Kaner, 1988). We then look at a few examples of multivariable testing with related variables. You’ll run into lots of examples like this on the job.

Understanding domain testing

- People were treating values as equivalent long before anyone proposed a theoretical description of domain testing.
- The most important idea in domain testing is that it provides a sensible basis for sampling from a domain.
- Definition: Domain
 - In mathematics,
 - The domain of a function is the set of all input values over which the function is defined.
 - The range (or output domain) of the function is the set of all values that the function can produce.
 - Early descriptions of domain testing focused on inputs, but we routinely applied the analysis to outputs (Testing Computer Software, 1st edition, 1988, reflected that practice.)

Understanding domain testing

*In domain testing, we
partition a domain
into sub-domains (equivalence classes)
and then test
using values from each sub-domain.*

Understanding domain testing

1. What is equivalence?

4 views of what makes values equivalent. Each has practical implications

- ***Intuitive Similarity***: two test values are equivalent if they are so similar to each other that it seems pointless to test both.
 - This is the earliest view and the easiest to teach
 - Little guidance for subtle cases or multiple variables
- ***Specified As Equivalent***: two test values are equivalent if the specification says that the program handles them in the same way.
 - Testers complain about missing specifications may spend enormous time writing specifications
 - Focus is on things that were specified, but there might be more bugs in the features that were un(der)specified

Understanding domain testing

1. What is equivalence?

- ***Equivalent Paths***: two test values are equivalent if they would drive the program down the same path (e.g. execute the same branch of an IF)
 - Tester should be a programmer
 - Tester should design tests from the code
 - Some authors claim that a complete domain test will yield a complete branch coverage.
 - No basis for picking one member of the class over another.
 - Two values might take program down same path but have very different subsequent effects (e.g. timeout or not timeout a subsequent program; or e.g. word processor's interpretation and output may be the same but may yield different interpretations / results from different printers.)
- ***Risk-Based***: two test values are equivalent if, given your theory of possible error, you expect the same result from each.
 - Subjective analysis, differs from person to person. It depends on what you expect (and thus, what you can anticipate).
 - Two values may be equivalent relative to one potential error but non-equivalent relative to another.

Understanding domain testing

2. Test which values from the equivalence class?

Most discussions of domain testing start from several assumptions:

- (a) The domain is continuous [*This is easily relaxed -- CK*]
- (b) The domain is linearizable (members of the domain can be mapped to the number line) or, at least, the domain is an ordered set (given two elements, one is larger than the other or they are equal)
- (c) The comparisons that cause the program to branch are simple, linear inequalities

“It is possible to move away from these assumptions, but the cost can be high.” --- Clarke, Hassell, & Richardson, p. 388

- If we think in terms of paths, can we use any value that drives the program down the correct path? This approach is common in coverage-focused testing. Unfortunately, it doesn't yield many failures. See Hamlet & Taylor
- If you can map the input space to a number line, then boundaries mark the point or zone of transition from one equivalence class to another. These are said to be good members of equivalence classes to use ***because the program is more likely to fail at a boundary.***

Understanding domain testing

2. Test which values from the equivalence class?

- The emphasis on boundaries is inherently risk-based
- But the explicitly risk-based approach goes further
 - Consider many different risks
 - Partitioning driven by risk
 - Selection of values driven by risk:
 - A member of an equivalence class is a **best representative** (relative to a potential error) if no other member of the class is more likely to expose that error than the best representative.
 - » Boundary values are often best representatives
 - » We can have best representatives that are not boundary values
 - » We can have best representatives in non-ordered domains

Risk-based equivalence

- Consider these cases. Are these paired tests equivalent?

– If you tested

Would you test

51+52

52+53

53+54

54+55

55+56

56+57

57+58

58+59

59+60

60+61

61+62

62+63

63+64

64+65

65+66

66+67

67+68

68+69

Risk-based equivalence

- Given the following potential error:

These cases would not trigger the error, even if it was there.	These cases would trigger the error.

Another example: Non-obvious boundaries

	<u>Character</u>	<u>ASCII Code</u>
	/	47
lower bound	0	48
	1	49
	2	50
	3	51
	4	52
	5	53
	6	54
	7	55
	8	56
upper bound	9	57
	:	58
	A	65
	a	97

Refer to Testing
Computer Software,
pages 9-11

Another example of non-obvious boundaries

- Still in the 99+99 program
- Enter the first value
- Wait N seconds
- Enter the second value
- Suppose our client application will time out on input delays greater than 600 seconds. Does this affect how you would test?
- Suppose our client passes data that it receives to a server, the client has no timeout, and the server times out on delays greater than 300 seconds.
 - Would you discover this timeout from a path analysis of your application?
 - What boundary values should you test? In whose domains?

In sum: equivalence classes and representative values

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.

In an ordered set, 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.

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. We call those most powerful tests the best representatives of the class.

Xref: stratified sampling: http://www.wikipedia.org/wiki/Stratified_sampling

A new boundary and equivalence table

Variable	Risk (potential failure)	Classes that should not trigger the failure	Classes that might trigger the failure	Test cases (best representatives)	Notes
First input	Fail on out-of-range values	-99 to 99	MinInt to -100 100 to MaxInt	-100, 100	
	Doesn't correctly discriminate in-range from out-of-range			-100, -99, 100, 99	
	Misclassify digits	Non-digits	0 to 9	0 (ASCII 48) 9 (ASCII 57)	
	Misclassify non-digits	Digits 0 - 9	ASCII other than 48 - 57	/ (ASCII 47) ; (ASCII 58)	

Note that we've dropped the issue of "valid" and "invalid." This lets us generalize to partitioning strategies that don't have the **concept** of "valid" -- for example, **printer** equivalence classes. (For discussion of device compatibility testing, see Kaner et al., Chapter 8.)

Examples of ordered sets

So many examples of domain analysis involve databases or simple data input fields that some testers don't generalize. Here's a sample of other variables that fit the traditional equivalence class / boundary analysis mold.

- ranges of numbers
- character codes
- how many times something is done
 - (e.g. shareware limit on number of uses of a product)
 - (e.g. how many times you can do it before you run out of memory)
- how many names in a mailing list, records in a database, variables in a spreadsheet, bookmarks, abbreviations
- size of the sum of variables, or 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

Examples of ordered sets

- size of a file (note special values such as exactly 64K, exactly 512 bytes, etc.)
- size of the document on the page (compared to page margins) (across different page margins, page sizes)
- 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.
- 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?

Examples of ordered sets

- 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 and time
- 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)

Non-ordered sets

- ***A sample problem:***
 - *There are about 2000 Windows-compatible printers, plus multiple drivers for each. We can't test them all.*
- ***They are not ordered, but maybe we can form equivalence classes and choose best representatives anyway.***
- Here are two examples from programs (desktop publishing and an address book) developed in 1991-92.

Non-ordered sets

Primary groups of printers at that time:

- HP - Original
- HP - LJ II
- PostScript Level I
- PostScript Level II
- Epson 9-pin, etc.

LaserJet II compatible printers, huge class (maybe 300 printers, depending on how we define it)

1. Should the class include LJII, LJII+, and LIIP, LJIID-compatible subclasses?
2. What is the best representative of the class?

Non-ordered sets

Example: graphic complexity error handling

- HP II original was the weak case.

Example: special forms

- HP II original was strong in paper-handling. We worked with printers that were weaker in paper-handling.

We pick different best representatives from the same equivalence class, depending on which error we are trying to detect.

Examples of additional queries for almost-equivalent printers

- Same margins, offsets on new printer as on HP II original?
- Same printable area?
- Same handling of hairlines? (Postscript printers differ.)

More examples of non-ordered sets

- Here are more examples of variables that don't fit the traditional mold for equivalence classes but which have enough values that we will have to sample from them. What are the boundary cases here?
- Membership in a common group
 - Such as employees vs. non-employees.
 - Such as workers who are full-time or part-time or contract.
- Equivalent hardware
 - such as compatible modems, video cards, routers
- Equivalent output events
 - perhaps any report will do to answer a simple the question: Will the program print reports?
- Equivalent operating environments
 - such as French & English versions of Windows 3.1

Interactions among variables

Rather than thinking about a single variable with a single range of values, a variable might have different ranges, such as the day of the month, in a date:

1-28

1-29

1-30

1-31

We analyze the range of dates by partitioning the month field for the date into different sets:

{February}

{April, June, September, November}

{Jan, March, May, July, August, October, December}

For testing, you want to pick one of each. There might or might not be a “boundary” on months. The boundaries on the days, are sometimes 1-28, sometimes 1-29, etc

- » This is nicely analyzed by Jorgensen:
- » Software Testing--A Craftsman's Approach.

Another example of interaction

- Interaction-thinking is important when we think of an output variable whose value is based on some input variables. Here's an example that gives students headaches on tests:
- I, J, and K are integers. The program calculates $K = I * J$. For this question, consider only cases in which you enter integer values into I and J. Do an equivalence class analysis from the point of view of **the effects of I and J (jointly) on the variable K**. Identify the boundary tests that you would run (the values you would enter into I and J) if
 - I, J, K are unsigned integers
 - I, J, K are signed integers

Domain Testing

- Strengths
 - Find highest probability errors with a relatively small set of tests.
 - Intuitively clear approach, easy to teach and understand
 - Extends well to multi-variable situations
- Blind spots or weaknesses
 - Errors that are not at boundaries or in obvious special cases.
 - The "competent programmer hypothesis" can be misleading.
 - Also, the actual domains are often unknowable.
 - Reliance on best representatives for regression testing leads us to overtest these cases and undertest other values that were as, or almost as, good.