

Black Box Software Testing

(Professional Seminar)

Cem Kaner, J.D., Ph.D.

Professor of Computer Sciences
Florida Institute of Technology

Section:3

Boundary and Equivalence Analysis

Summer, 2002

Contact Information:

kaner@kaner.com

www.kaner.com (testing website)

www.badsoftware.com (legal website)

I grant permission to make digital or hard copies of this work for personal or classroom use, with or without fee, provided that (a) copies are not made or distributed for profit or commercial advantage, (b) copies bear this notice and full citation on the first page, and if you distribute the work in portions, the notice and citation must appear on the first page of each portion, (c) each page bear the notice "Copyright (c) Cem Kaner" or if you changed the page, "Adapted from Notes Provided by Cem Kaner". Abstracting with credit is permitted. The proper citation for this work is Cem Kaner, *A Course in Black Box Software Testing (Professional Version)*, Summer-2002, www.testing-education.org. 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 kaner@kaner.com.

Black Box Software Testing

Boundary and Equivalence Analysis

Supplementary Reading:

- Cem Kaner, Liability for Product Incompatibility
- Clarke, Hassell, & Richardson: A Close Look at Domain Testing, *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 4, July 1982.
- Ostrand & Balcer: The category-partition method for specifying and generating functional tests. *Communications of the ACM*, Vol. 31 (6), pages 676-686, June 1988.

Exercise

For each of the following, list

- The variable(s) of interest
- The valid and invalid classes
- The boundary value test cases.

1. FoodVan delivers groceries to customers who order food over the Net. To decide whether to buy more 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.

Expanding the Notion of Equivalence

Consider these cases. Are these paired tests equivalent?

(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)

(69+70, 70+71)

Example: The Hockey Game

Another Example: Boundaries May Not Be Obvious

	<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

Equivalence Classes and Partitions are Subjective.

My working definition of equivalence:

Two test cases are equivalent if you expect the same result from each.

This is fundamentally subjective. It depends on what you expect. And what you expect depends on what errors you can anticipate:

Two test cases can only be equivalent by reference to a specifiable risk.

Two different testers will have different theories about how programs can fail, and therefore they will come up with different classes.

Equivalence Classes and Boundaries

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 Class	Alternate Equivalence Class	Boundaries and Special Cases	Notes
First number	-99 to 99 digits	> 99 < -99 non-digits expressions	99, 100 -99, -100 /, 0, 9, : leading spaces or 0s null entry	
2nd number	same as first	same as first	same	
Sum	-198 to 198 -127 to 127	??? -198 to -128 128 to 198	??? 127, 128, - 127, -128	Are there other sources of data for this variable? Ways to feed it bad data?

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.)

Domain Testing

In classical domain testing

- Two values (single points or n-tuples) are equivalent if the program would take the same path in response to each.

The classical domain strategies all assume

- that the predicate interpretations are simple, linear inequalities.
- the input space is continuous and
- coincidental correctness is disallowed.

“It is possible to move away from these assumptions, but the cost can be high. The emphasis on paths is troublesome because of the high number of possible paths through the program.”

Clarke, Hassell, & Richardson, p. 388

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)?

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

Interaction example

K can run from MinInt (smallest integer) to MaxInt.

For any value of K, there is a set of values of I and J that will yield K

- The set is the null set for impossible values of K
- The set might include only two pairs, such as $K = \text{MaxInt}$, when MaxInt is prime (7 could be a MaxInt)
 - » $(I,J) \in \{(1, \text{MaxInt}), (\text{MaxInt}, 1)\}$
- The set might include a huge number of pairs, such as when K is 0:
 - » $(I,J) \in \{(0,0), (1, 0), (2, 0), \dots, (\text{MaxInt},0), (0,1), \dots, (0, \text{MaxInt},1)\}$
- **A set of pairs of values of I and J can be thought of as an equivalence set (they all yield the same value of K) and so we ask which values of K are interesting (partition number 1) and then which values of I and J would produce those K-values, and do a partitioning on the sets of (I,J) pairs to find best representatives of those.**
- **As practitioners, we do this type of analysis often, but many of us probably don't think very formally about it.**

Fuzzy Boundaries

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

Problem:

- There are about 2000 Windows-compatible printers, plus multiple drivers for each. We can't test them all.

But maybe we can form equivalence classes.

An example from two programs (desktop publishing and an address book) developed in 1991-92.

Fuzzy Boundaries

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?

Fuzzy Boundaries

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.

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.)

What exercises can we do to support development of this type of analysis?

Partitioning

Device compatibility testing illustrates a multidimensional space with imperfect divisions between classes and with several different failure risks. From an equivalence class of “LaserJet II compatibles” you get several different, uniquely powerful, class representatives.

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.

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

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.

We might miss testing a key device because we didn’t think through all of the failure modes we were looking for.

Partitioning

The problem (as laid out by a client):

“Equivalence class methods would be very valuable in situations where someone wants every OEM system, every sound and video card, every operating system, multiple technologies (e.g., DirectX 3 and 5), etc.

“How can the engineer be confident in developing an equivalence matrix that provides good coverage?”

The disappointing but very real answer:

“Despite good analysis and execution, we might simply miss a device or driver or a combination of devices and drivers that has a bug associated with it. If we convince a stakeholder that equivalence class analysis provides “complete” coverage, she will be justifiably upset if our ‘complete coverage’ misses a bug.”

Equivalence Classes: A Broad Concept

The notion of equivalence class is much broader than numeric ranges. Here are some examples:

- Membership in a common group
 - » such as employees vs. non-employees. (Note that not all classes have shared boundaries.)
- Equivalent hardware
 - » such as compatible modems
- Equivalent event times
 - » such as before-timeout and after
- 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

Variables Suited to Equivalence Class Analysis

There are many types of variables, such as:

- 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 examples:

Variables Well Suited to Equivalence Class Analysis

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 examples:

- **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**

Variables Well Suited to Equivalence Class Analysis

- | | |
|---|---|
| <ul style="list-style-type: none">▪ 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) | <ul style="list-style-type: none">▪ 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? |
|---|---|

Variables Well Suited to Equivalence Class Analysis

- | | |
|--|--|
| <ul style="list-style-type: none">▪ 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 | <ul style="list-style-type: none">• 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) |
|--|--|

Closing Notes on Boundary Analysis

- In the traditional approach, we analyzed the code from the point of view of the programmer and the specification.
- The specification provides us with only one of several classes of risk. We have to test against a broader set.
- Experts in different subject matters will see different opportunities for failure, and different classes of cases that might reveal these failures.
- Thinking outside of the box posed by the explicit design is at the essence of black box testing.

