

Black Box Software Testing

Spring 2005

DOMAIN TESTING, Part 2

by

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

Professor of Software Engineering

Florida Institute of Technology

and

James Bach

Principal, Satisfice Inc.

Copyright (c) Cem Kaner & James Bach, 2000-2004

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.

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.

Yesterday's 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.

Notes on the Exercise

- Even these simple specifications are ambiguous:
 - Does “within 30 days” mean “less than 30” or “less than or equal to 30” ?
 - When does the special permission have to have been issued?
 - If you can work tomorrow morning on the basis of permission, can you work tomorrow afternoon on the basis of experience? Is tomorrow morning within 30 days of tomorrow afternoon?
 - Do we compute 30 days in days or hours (minutes / seconds)?
 - What result if the last day you worked was 28 days ago? 29 days ago? 30 days ago?
- Even if you are clear on the answers to these, do you believe that the programmer and the specification writer will come to the same answers?

Understanding domain testing

- 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.)

A new class of example to consider:

Non-ordered sets

- Let's discard the notion that a domain must be linear (Clarke et al, 1982), and consider domains that can't be ordered from small to large. (Kaner et. al, 1993 on printer testing, Kaner, 1988).
- Boundary analysis depends on the existence of boundaries. Theorists often say that domain (boundary) analysis assumes that variables are linearizable (can be mapped to the number line). All we actually need, though is ordinality--a variable is ordinally scaled if its values can be ordered from smallest to largest.
- ***A problem:***
 - *There are about 2000 Windows-compatible printers, plus multiple drivers for each. We can't test them all.*
- These are not ordered, and so we can never do a boundary analysis of them. However, we might be able to form equivalence classes and choose best representatives.
- Here are two examples from programs (desktop publishing and an address book) developed in 1990-92.

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

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 (Kaner, 2004)

- ***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

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.)

Understanding domain testing

What is equivalence?

- ***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.
- (Read Weyuker & Jeng, 1991 for an example of domain testing that uses a risk analysis.)

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

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

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 summary

- AKA partitioning, equivalence analysis, boundary analysis
- Fundamental question or goal:
 - This confronts the problem that there are too many test cases for anyone to run. This is a stratified sampling strategy that provides a rationale for selecting a few test cases from a huge population.
- General approach:
 - Divide the set of possible values of a field into subsets, pick values to represent each subset. The goal is to find a “best representative” for each subset, and to run tests with these representatives. Best representatives of ordered fields will typically be boundary values.
 - *Multiple variables*: combine tests of several “best representatives” and find a defensible way to sample from the set of combinations.
- Paradigmatic case(s)
 - Equivalence analysis of a simple numeric field.
 - Printer compatibility testing (*multidimensional variable, doesn't map to a simple numeric field, but stratified sampling is essential.*)

Domain testing summary

- 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.
- One reason that 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.