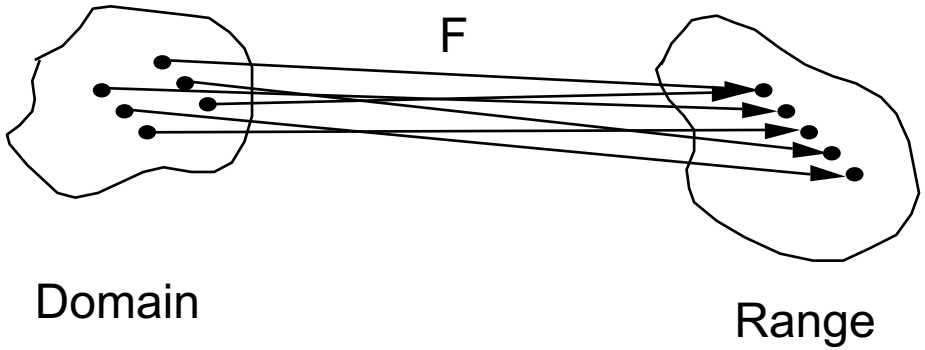
Chapter 5

Boundary Value Testing

Functional Testing

The rationale for referring to specification-based testing as “functional testing” is likely due to the abstraction that any program can be viewed as a mapping from its Input Domain to its Output Range:

Output = F (Input)



Functional testing uses information about functional mappings to identify test cases. We illustrate much of this with a function F of two variables, x_1 and x_2 . Extension to more variables is straightforward.

Mainline Spec-based Testing Techniques

- Boundary Value Testing (4 flavors)
- Equivalence Partitions (Chapter 6)
- Special Value Testing
- Output Domain (Range) Checking
- Decision Table Based Testing, aka Cause and Effect Graphs (Chapter 7)

Boundary Value Testing

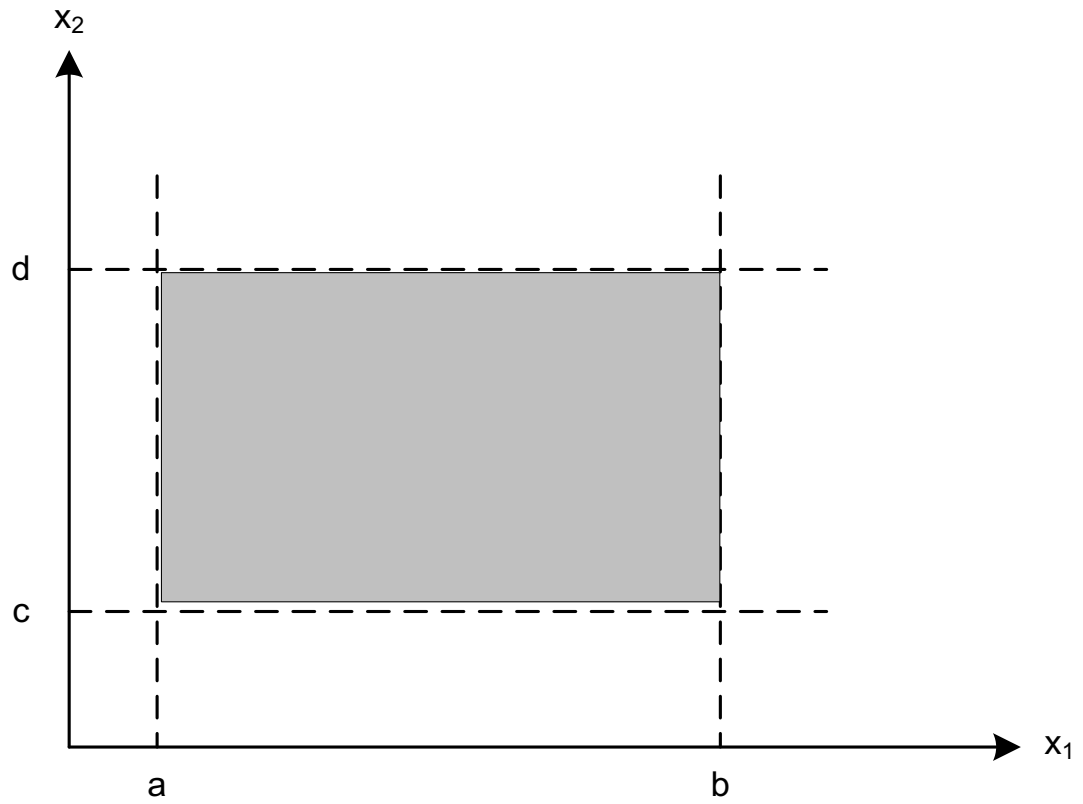
- Two considerations apply to boundary value testing
  - are invalid values an issue?
  - can we make the “single fault assumption” of reliability theory?
- Consequences...
  - invalid values require the robust choice
  - multiplicity of faults requires worst case testing
- Taken together, these yield four variations
  - Normal boundary value testing
  - Robust boundary value testing
  - Worst case boundary value testing
  - Robust worst case boundary value testing

Rationale for Boundary Value Testing

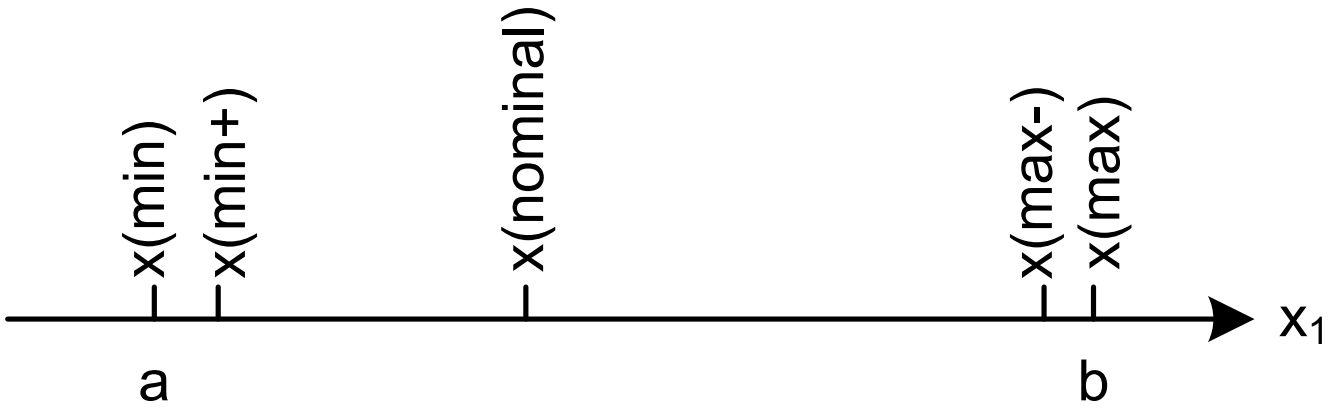
- Industrial, commercial, and defense software all note that faults seem to be more prevalent when variables have values at or near their extreme boundaries.
- In the early 1990s, there was a commercial test tool, named simply “T”, that implemented boundary value testing.
- T was originally requested by the U.S. Army
- The product has been merged into
  - Teamwork from Cadre Systems, and
  - Software through Pictures from Aonix

Input Domain of $F(x_1, x_2)$

where $a \leq x_1 \leq b$ and $c \leq x_2 \leq d$

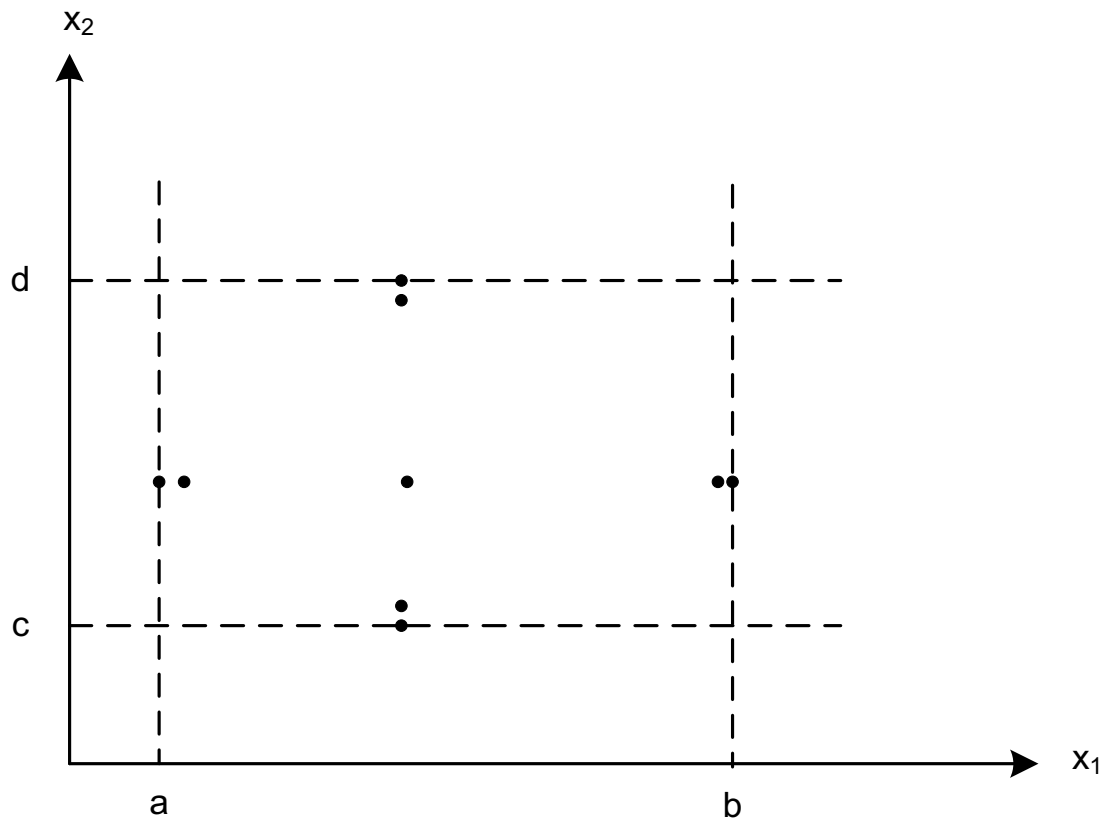


Input Boundary Value Testing



- Test values for variable x, where
  - $a \leq x_1 \leq b$, and
  - x(min), x(min+), ... x(max) are the names from the T tool.

Normal Boundary Value Test Cases



As in reliability theory, two variables rarely both assume their extreme values.

Method

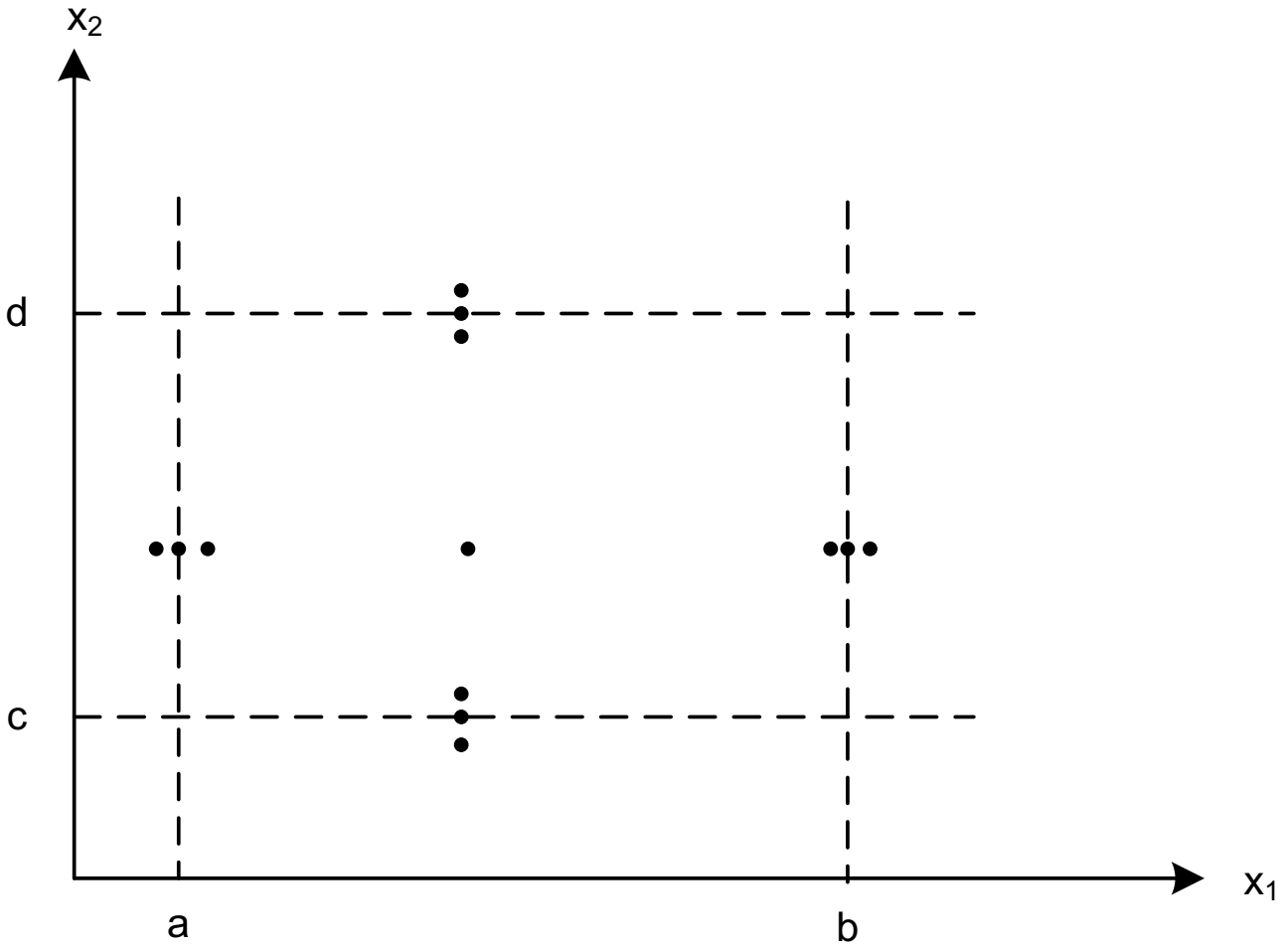
- Hold all variables at their nominal values.
- Let one variable assume its boundary values.
- Repeat this for each variable.
- This will (hopefully) reveal all faults that can be attributed to a single variable.

Exercise: why might this not work?

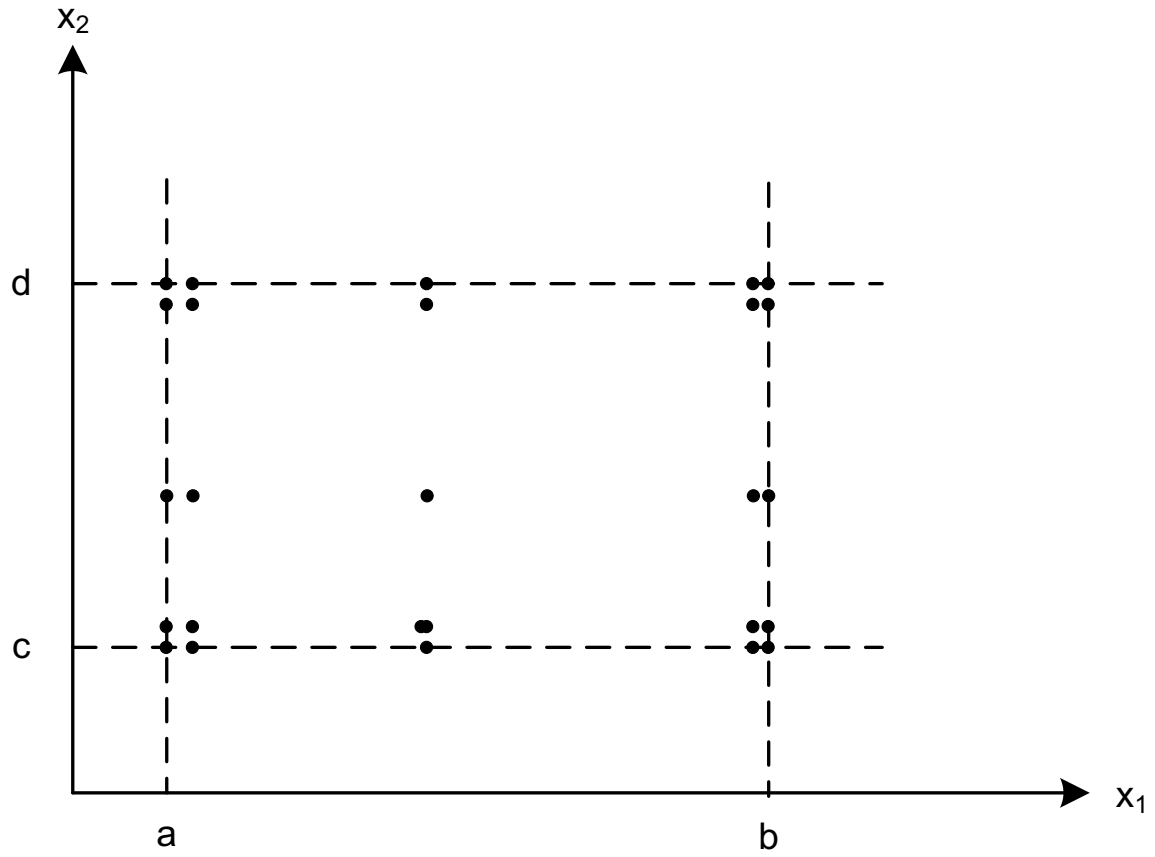
Robustness Testing

- Stress boundaries
- Possible advantages
  - find hidden functionality
  - leads to exploratory testing
- But...
  - what are the expected outputs?
  - what if the programming language is strongly typed? (*e.g.,* Ada)

Robust Boundary Value Test Cases

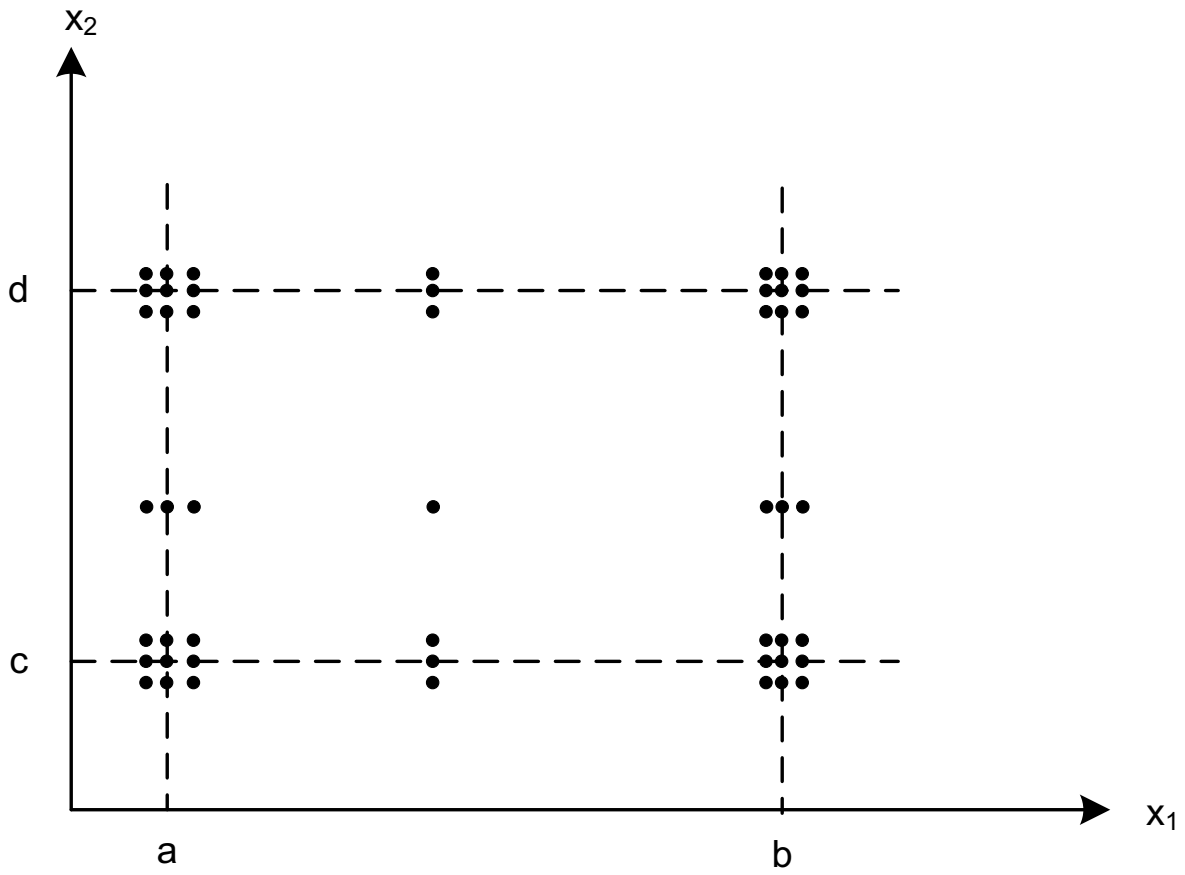


Normal Worst Case Boundary Value Test Cases



Responding to the single-fault assumption.

Robust Worst Case Boundary Value Test Cases



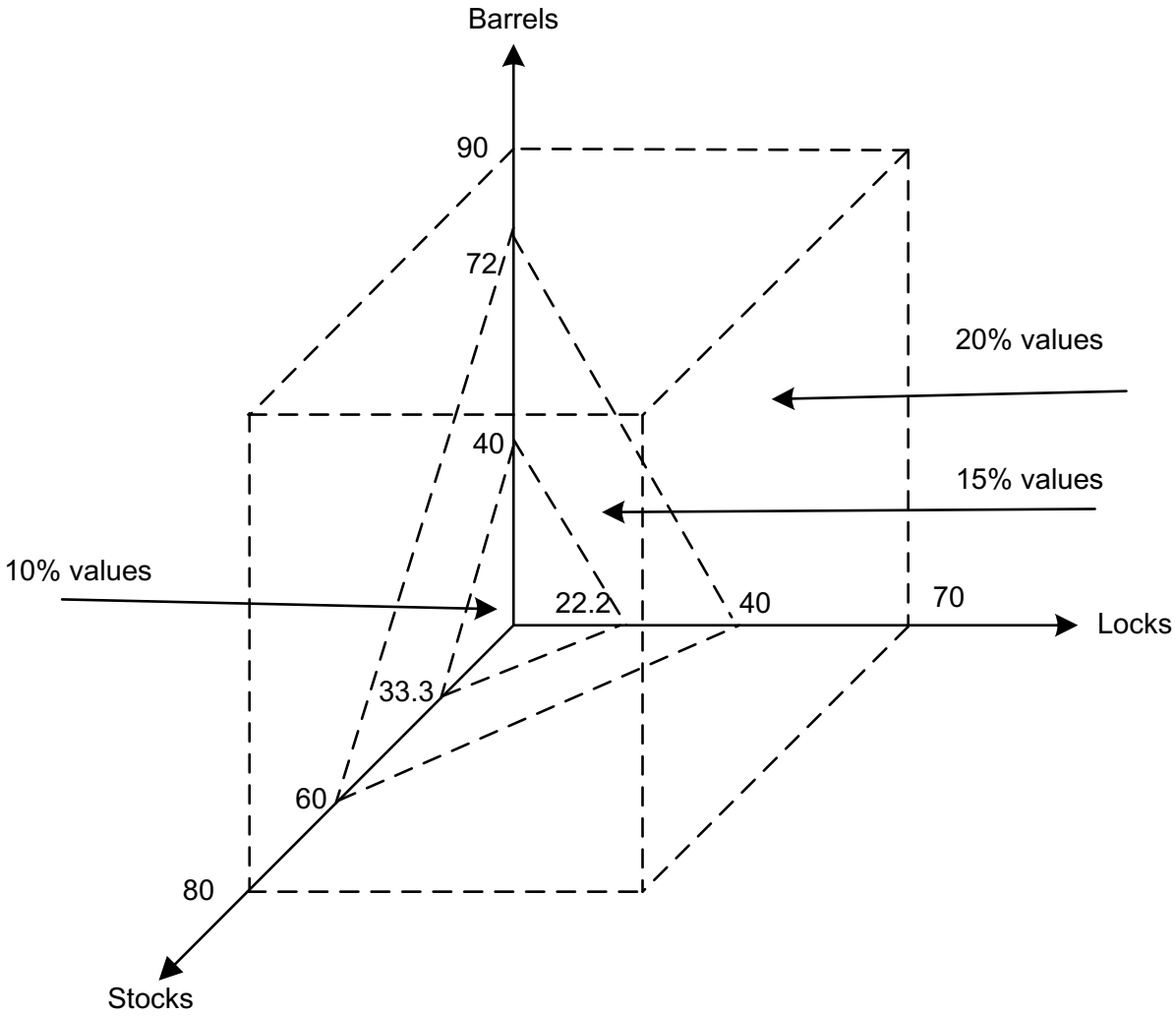
Special Value Testing

- Appropriate for
  - complex mathematical calculations
  - worst case situations (similar to robustness)
  - problematic situations from past experience
- Characterized by...
  - "second guess" likely implementations
  - experience helps
  - frequently done by customer/user
  - defies measurement
  - highly intuitive
  - seldom repeatable
  - (and is often most effective)

Output Range Coverage

1. Work "backwards" from expected outputs (assume that inputs cause outputs).
2. Mirror image of equivalence partitioning (good cross check)
3. Helps identify ambiguous causes of outputs.

Input Domain for Commission Problem



NextDate Function

NEXTDATE is a function of three variables: month, day, and year, for years from 1812 to 2012. It returns the date of the next day.

NEXTDATE(Dec, 31, 1991) returns Jan 1 1992

NEXTDATE(Feb, 21, 1991) returns Feb 22 1991

NEXTDATE(Feb, 28, 1991) returns Mar 1 1991

NEXTDATE(Feb, 28, 1992) returns Feb 29 1992

Leap Year: Years divisible by 4 except for century years not divisible by 400. Leap Years include 1992, 1996, 2000. 1900 was not be a leap year.

Boundary Value Test Cases for NextDate

- Observations
  - not much reason for robustness testing
  - good reasons for worst case testing
- Large number of test cases
  - 15 normal boundary value test cases
  - 125 worst case boundary value test cases
  - (see text for test case values)
- What problems do you see with the normal boundary value test cases on the next slide?

Boundary Value Test Cases for NextDate

Case	*Month*	*Day*	*Year*	*Expected Output*
1	1	1	1812	1, 2, 1812
2	1	1	1813	1, 2, 1813
3	1	1	1912	1, 2, 1912
4	1	1	2011	1, 2, 2011
5	1	1	2012	1, 2, 2012
6	1	2	1812	1, 3, 1812
7	1	2	1813	1, 3, 1813
8	1	2	1912	1, 3, 1912
9	1	2	2011	1, 3, 2011
10	1	2	2012	1, 3, 2012
11	1	15	1812	1, 16, 1812
12	1	15	1813	1, 16, 1813
13	1	15	1912	1, 16, 1912
14	1	15	2011	1, 16, 2011
15	1	15	2012	1, 16, 2012

Special Value Test Cases

Case	*Month*	*Day*	*Year*	*Reason*
SV-1	2	28	2012	Feb. 28 in a leap year
SV-2	2	28	2013	Feb. 28 in a common year
SV-3	2	29	2012	Leap day in a leap year
SV-4	2	29	2000	Leap day in 2000
SV-5	2	28	1900	Feb. 28 in 1900
SV-6	12	31	2011	End of year
SV-7	10	31	2012	End of 31-day month
SV-8	11	30	2012	End of 30-day month
SV-9	12	31	2012	Last day of defined interval

Output Range Test Cases

In the case of the NextDate function, the range and domain are identical except for one day . Nothing interesting will be learned from output range test cases for this example.

Part of the reason for this is that the NextDate function is a one-to-one mapping from its domain onto its range. When functions are not one-to-one, output range test cases are more useful.

Pros and Cons of Boundary Value Testing

- Advantages
  - Commercial tool support available
  - Easy to do/automate
  - Appropriate for calculation-intensive applications with variables that represent physical quantities (e.g., have units, such as meters, degrees, kilograms)
- Disadvantages
  - Inevitable potential for both gaps and redundancies
  - The gaps and redundancies can never be identified (specification-based)
  - Does not scale up well (Jorgensen's Law)
  - Tools only generate inputs, user must generate expected outputs.