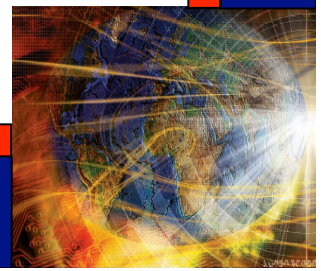
Mainline Functional Testing Techniques

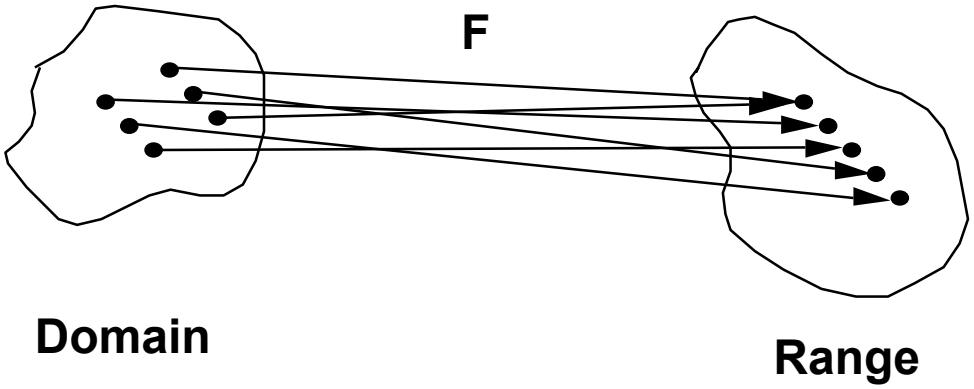
- Boundary Value Testing (4 flavors)
- Equivalence Partitions (another 4 flavors)
- Special Value Testing
- Output Domain (Range) Checking
- Decision Table Based Testing (aka Cause and Effect Graphs



Functional Testing

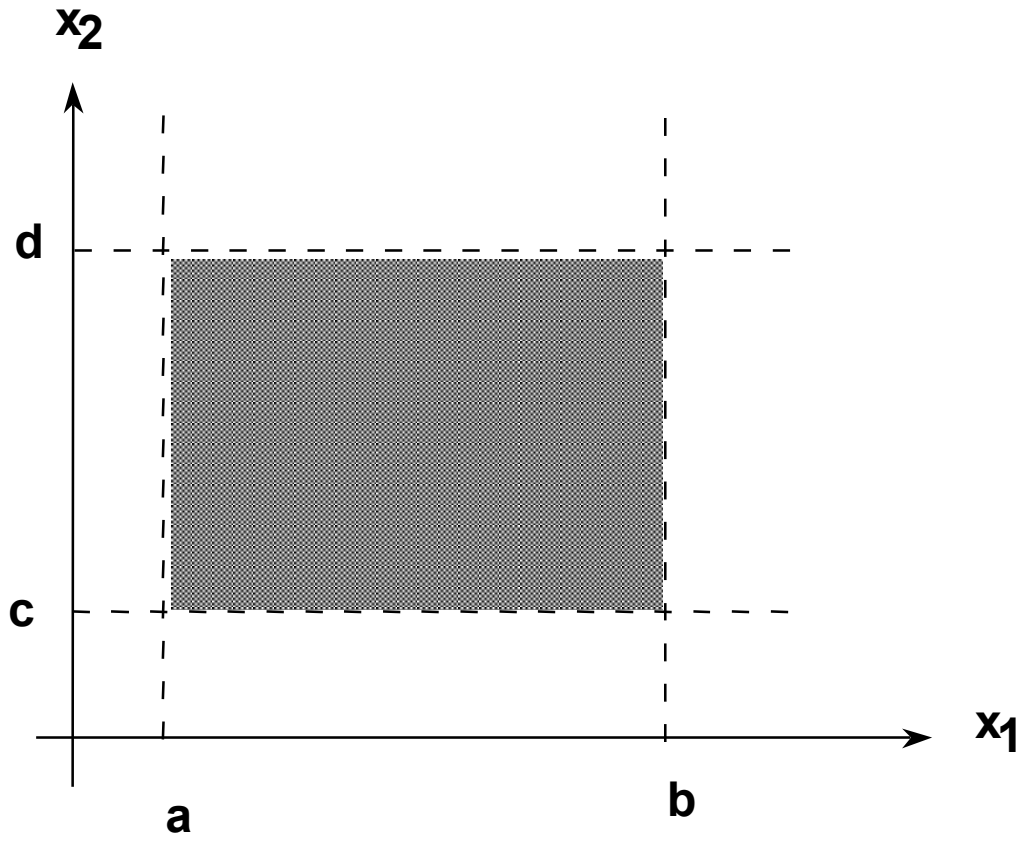
Ultimately, any program can be viewed as a mapping from its Input Domain to its Output Range:

Output = F (Input)

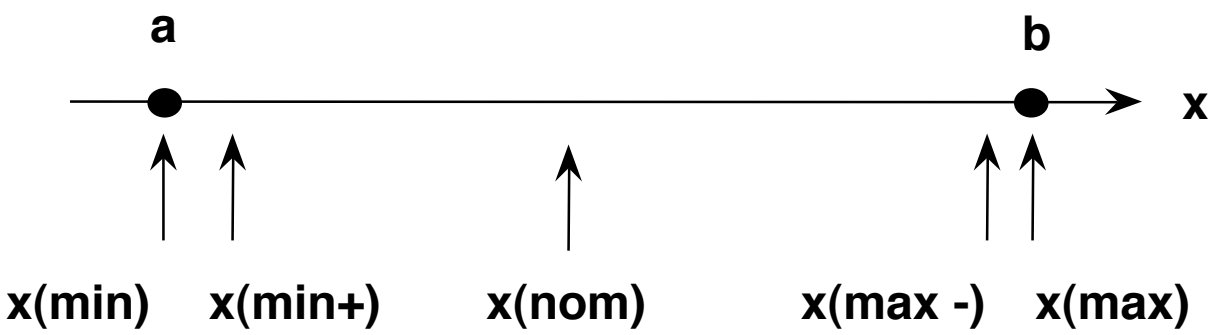


Functional testing uses information about functional mappings to identify test cases.

Input Domain of $F(x_1, x_2)$



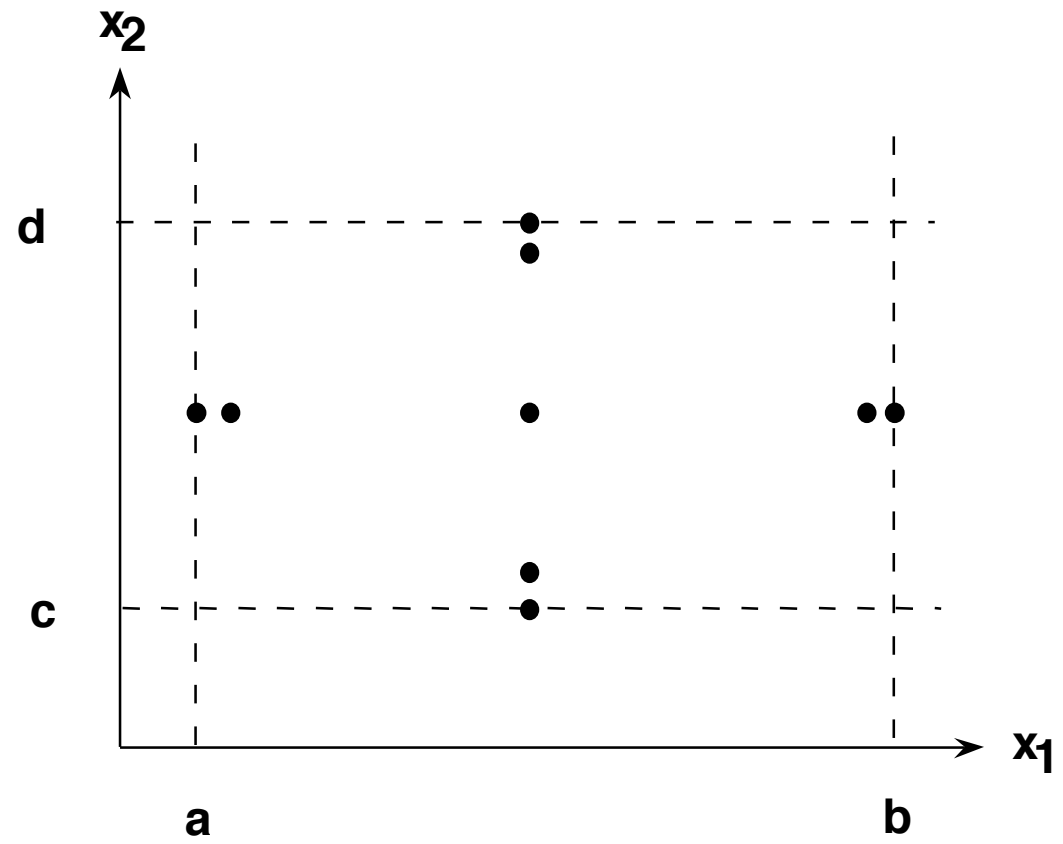
Input Boundary Value Testing



test cases for a variable x, where a <= x <= b

Experience shows that errors occur more frequently for extreme values of a variable.

Input Boundary Value Test Cases (2 Variables)



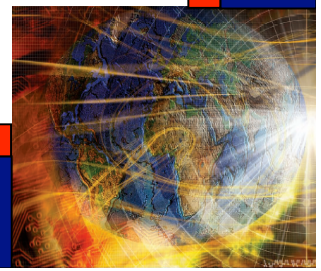
test cases for a variables x_1 and x_2 , where $a <= x_1 <= b$ and $c <= x_2 <= d$

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

Method

- Hold all variables at their nominal value.
- 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.

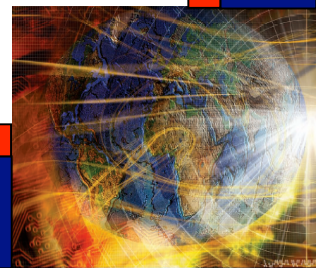
Exercise: why might this not work?



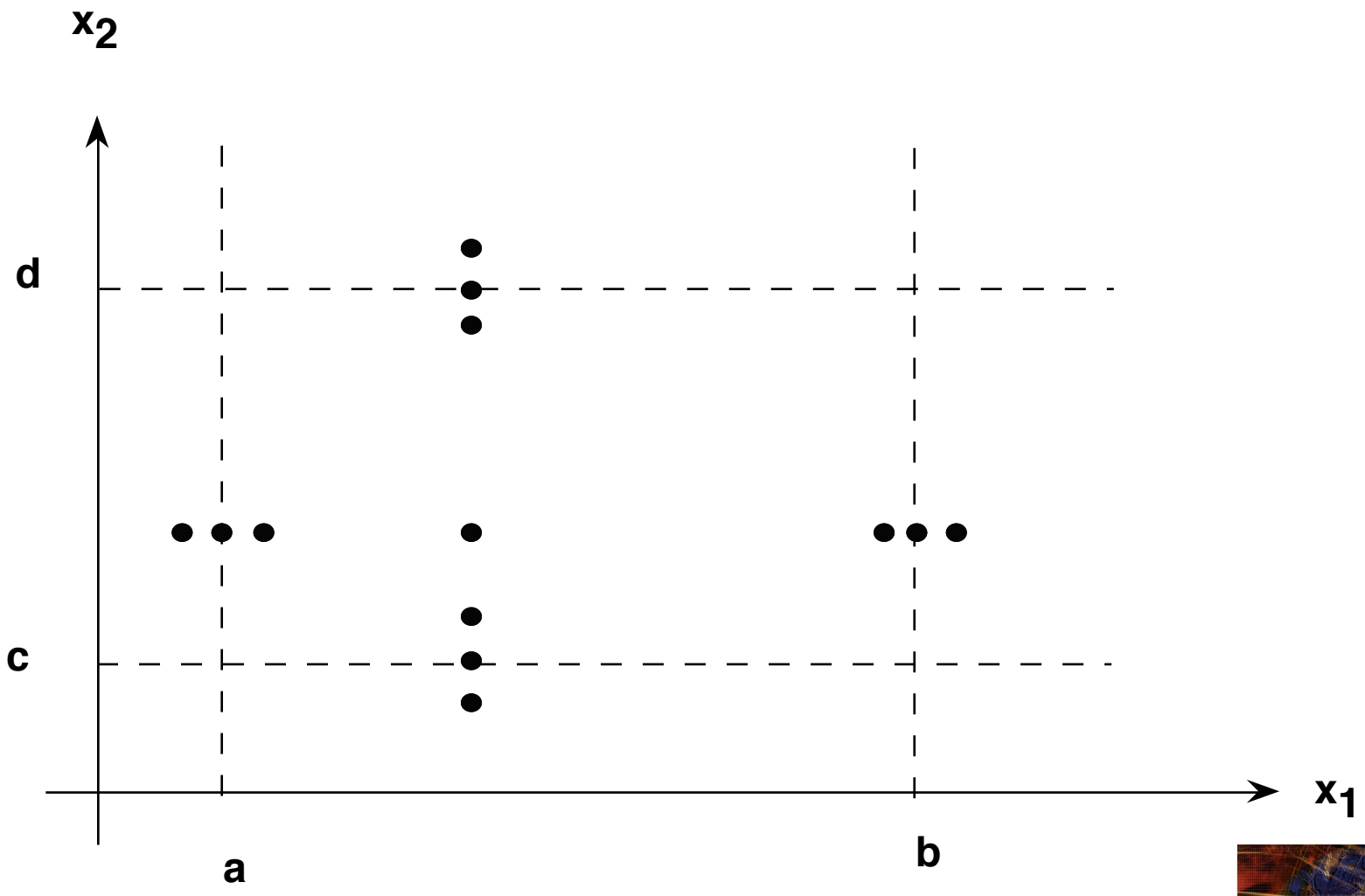
Robustness Testing

test cases for a variable x, where a <= x <= b

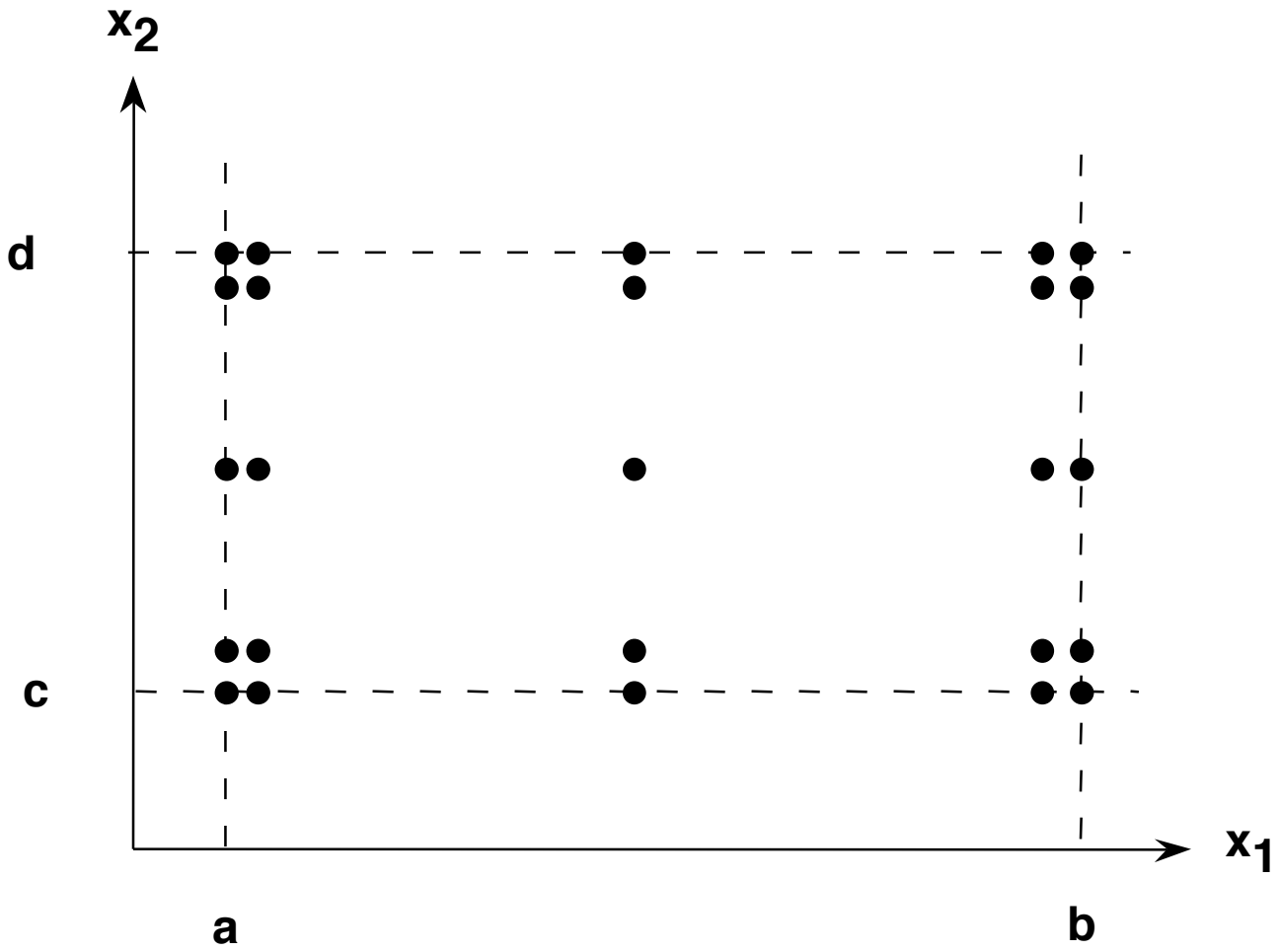
1. stress input boundaries
2. acceptable response for invalid inputs?
3. leads to exploratory testing (test hackers)
4. can discover hidden functionality



Robustness Testing (2 Variables)



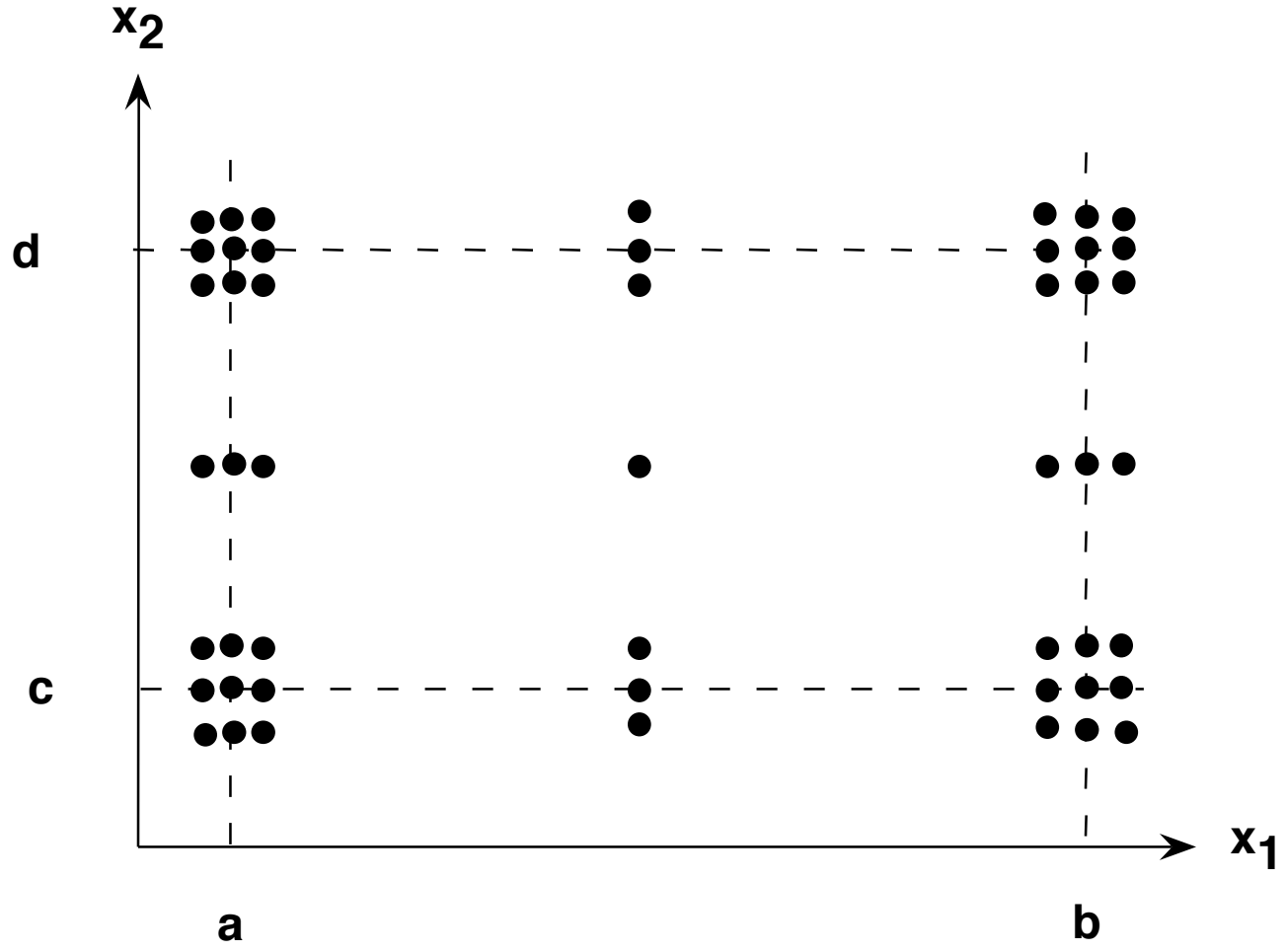
Worst Case Testing (2 Variables)



Eliminate the "single fault" assumption.

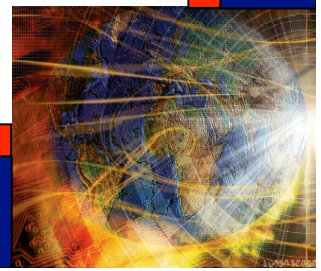
Murphy's Law?

Robust Worst Case Testing (2 Variables)



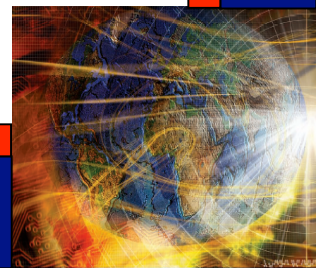
Special Value Testing

1. complex mathematical (or algorithmic) calculations
2. worst case situations (similar to robustness)
3. problematic situations from past experience
4. "second guess" likely implementations
5. experience helps
6. frequently done by customer/user
7. defies measurement
8. highly intuitive
9. seldom repeatable
10. (and is often most effective)

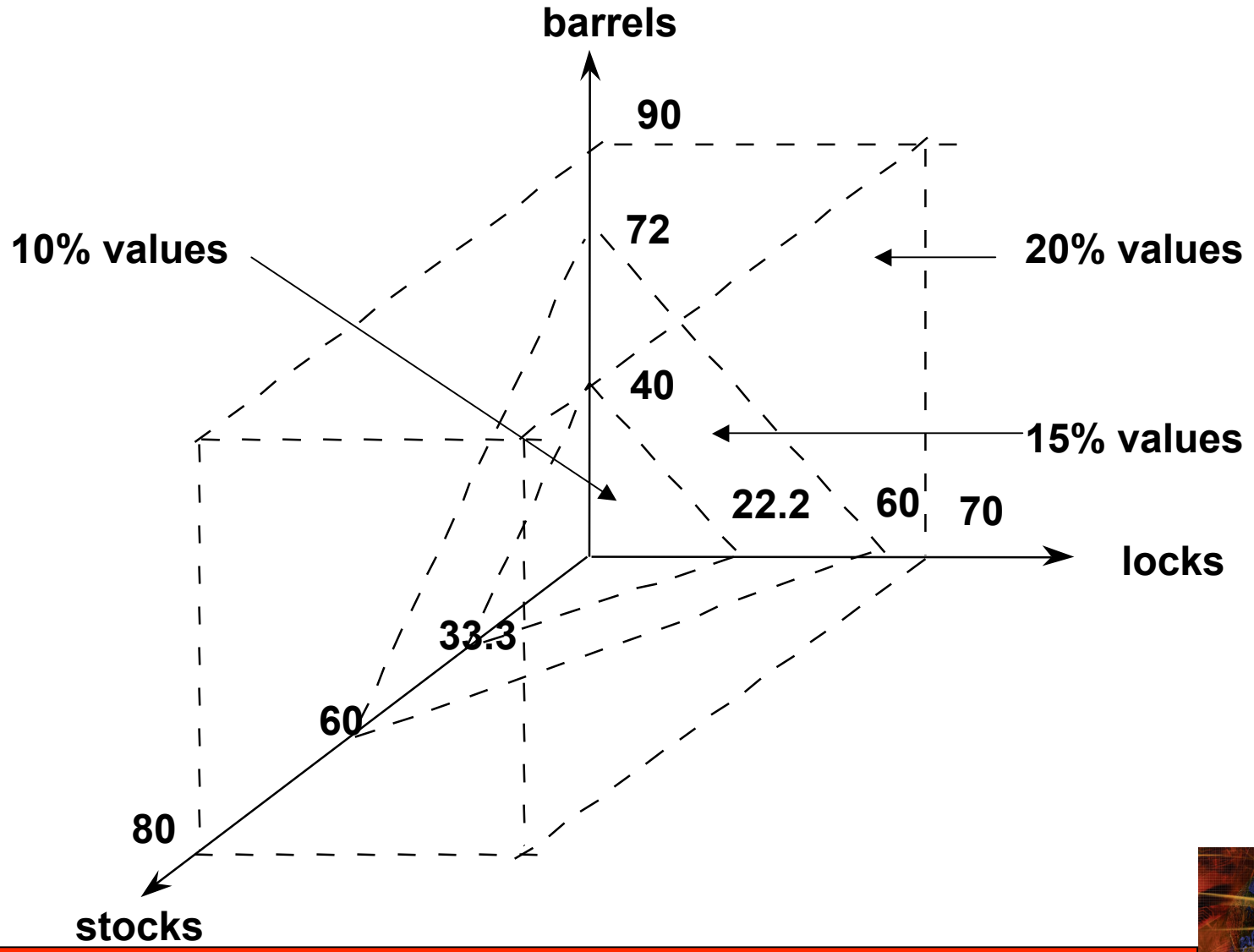


Output Range Coverage

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 Lock, Stock, and Barrel



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.

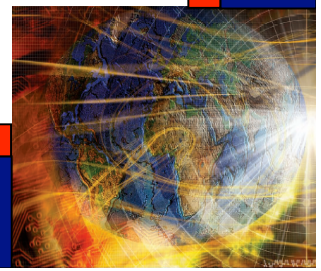
NEXTDATE(Dec, 31, 1991) returns Jan 1 1992

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.



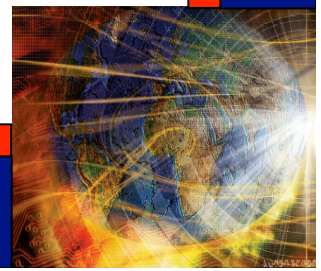
Input Domain Test Cases

Test Case	Input Values: Month	Day	Year
ID-1	Jan	15	1912
ID-2	Feb	15	1912
ID-3	Jun	15	1912
ID-4	Nov	15	1912
ID-5	Dec	15	1912
ID-6	Jun	1	1912
ID-7	Jun	2	1912
ID-3	Jun	15	1912
ID-8	Jun	30	1912
ID-9	Jun	31	1912
ID-10	Jun	15	1812
ID-11	Jun	15	1913
ID-3	Jun	15	1912
ID-12	Jun	15	2011
ID-13	Jun	15	2012

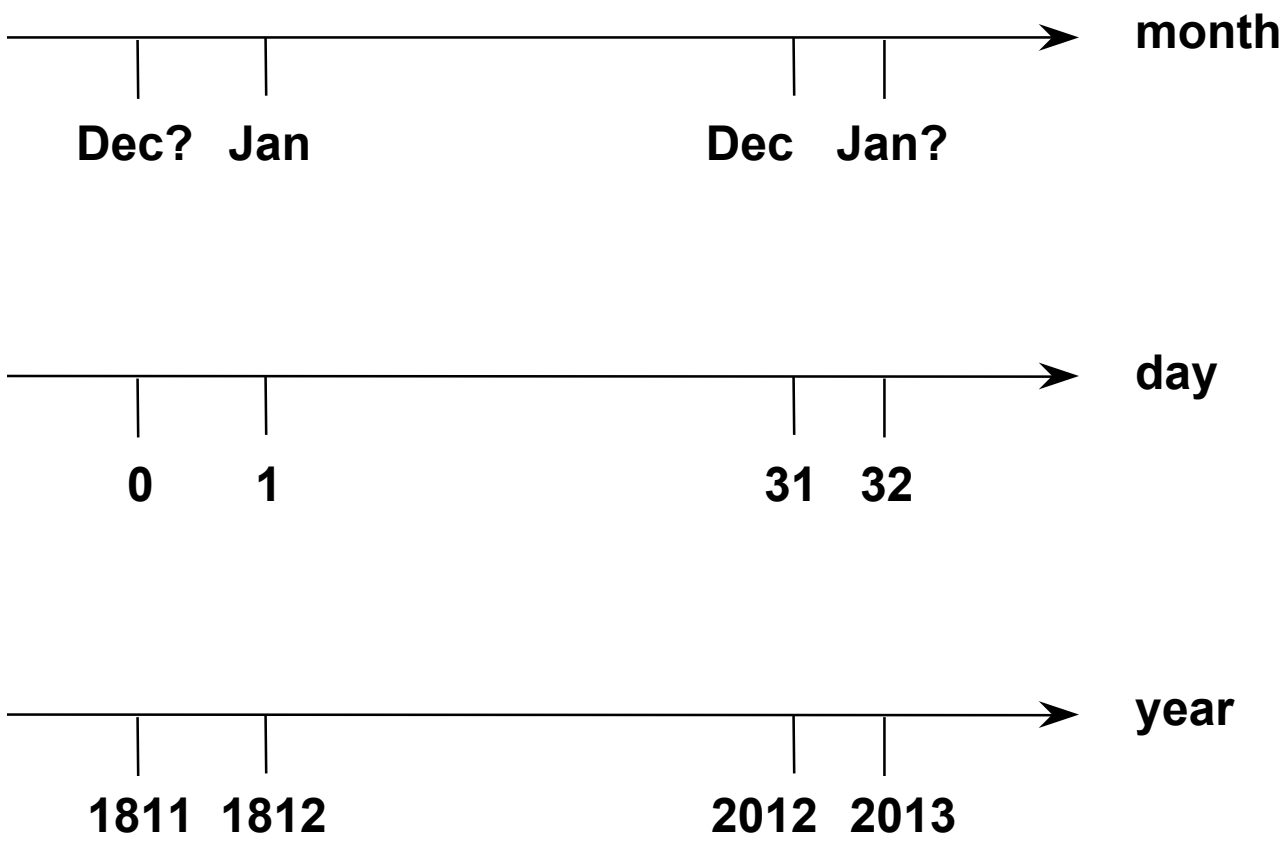
Select test cases so that one variable assumes nominal and extreme values while others are held at nominal values.

Notice that Input Domain testing presumes that the variables in the input domain are independent; logical dependencies are unrecognized.

This typically results in "anomalies" like test case ID-9, which is logically impossible.



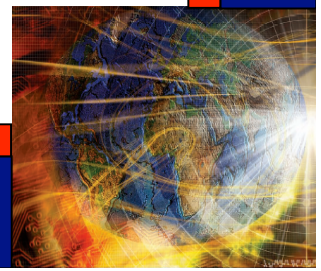
Robust Input Domain Test Cases



Robust Input Domain Test Cases

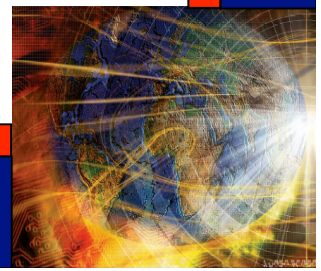
Robust Test Case	Test Case	Input Values: Month	Day	Year
RD-1	ID-5	Dec	15	1912
RD-2	ID-1	Jan	15	1912
RD-3	ID-2	Feb	15	1912
RD-4	ID-3	Jun	15	1912
RD-5	ID-4	Nov	15	1912
RD-6	ID-5	Dec	15	1912
RD-7	ID-1	Jan	15	1912
RD-8		Jun	0	1912
RD-9	ID-6	Jun	1	1912
RD-10	ID-7	Jun	2	1912
RD-11	ID-3	Jun	15	1912
RD-12	ID-8	Jun	30	1912
RD-13	ID-9	Jun	31	1912
RD-14		Jun	32	1912
RD-15		Jun	15	1811
RD-16	ID-10	Jun	15	1812
RD-17	ID-11	Jun	15	1913
RD-18	ID-3	Jun	15	1912
RD-19	ID-12	Jun	15	2011
RD-20	ID-13	Jun	15	2012
RD-21		Jun	15	2013

As with input domain testing, Robustness Testing presumes independent variables. The major difference is that robustness Testing also presumes that variables represent continuous functions. Notice that the Robustness Test Cases for Month and Day sometimes don't make sense, but those for Year do.



Special Value Test Cases

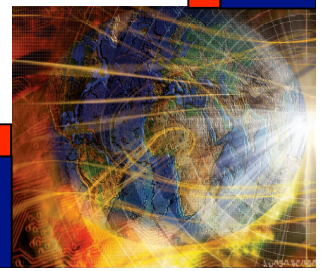
Test Case	Input Values			Reason for Test Case
	Month	Day	Year	
SV-1	Feb	28	1912	Leap day in a leap year
SV-2	Feb	28	1911	Leap day in a non-leap year
SV-3	Feb	29	1912	Leap day increment in a leap year
SV-4	Feb	28	2000	Leap day in the year 2000
SV-5	Feb	28	1900	Leap day in the year 1900
SV-6	Dec	31	1912	Change of year
SV-7	Jan	31	1912	Change of a 31 day month
SV-8	Apr	30	1912	Change of a 30 day month
SV-9	Dec	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.