Functional Testing
Boundary Value Analysis

Lecture 5a

Agenda

- Verification and Validation
- Functional Testing
- Boundary Value Analysis
  - Normal boundary value testing
  - Robust boundary value testing
  - Worst-case boundary value testing
  - Robust worst-case boundary value testing
  - Special Value Testing
  - Random Testing
- Limitations of Boundary Value Analysis

Verification and Validation

Verification and Validation

- Testing is part of a broader process of software verification and validation (V & V).
  - Verification and validation are not the same thing, although they are often confused.
- Software **verification** is the **process** of checking that the software **meets** its stated functional and non-functional **requirements**.
- The aim of software **validation** is to **ensure** that the software meets the **customer's expectations**. It goes beyond checking conformance with the specification to demonstrating that the software does what the customer expects it to do.

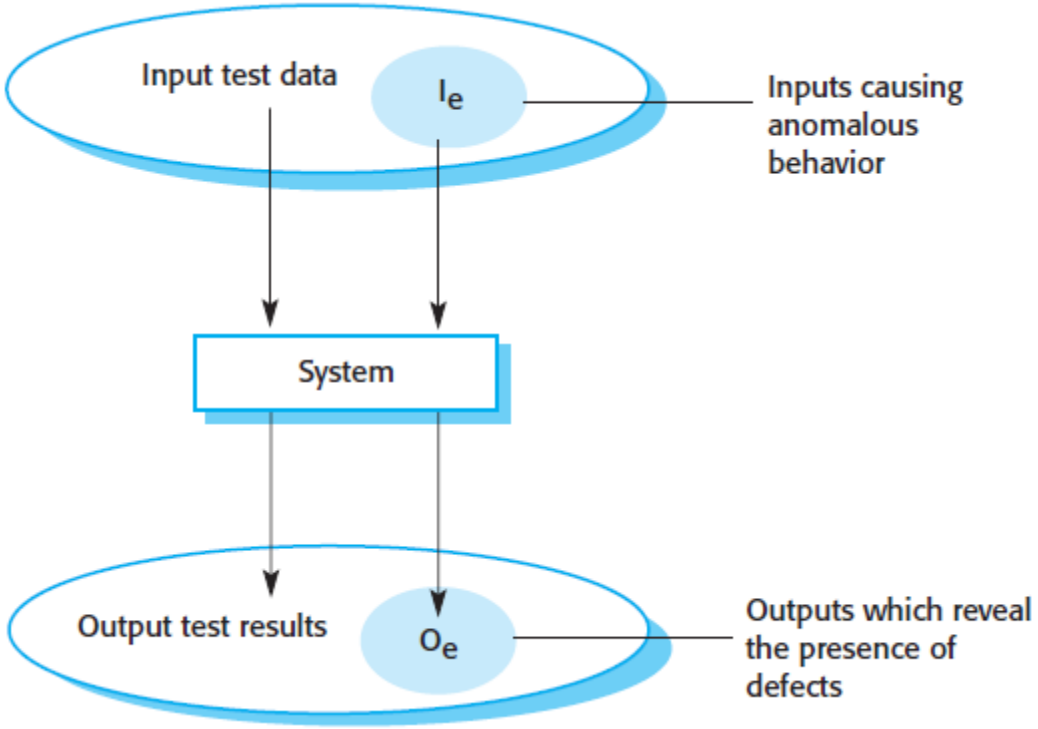
Verification and Validation

- Verification: “Are we building the product right?”
- Validation: “Are we building the right product?”
- Although testing plays an extremely important role in V&V, many other activities are also necessary, such as:
  - Software inspections and reviews
  - Quality and configuration audits
  - Performance monitoring
  - Simulation
  - Algorithm analysis
  - Documentation review.

Functional Testing

Functional Testing

- An input-output model of software testing:

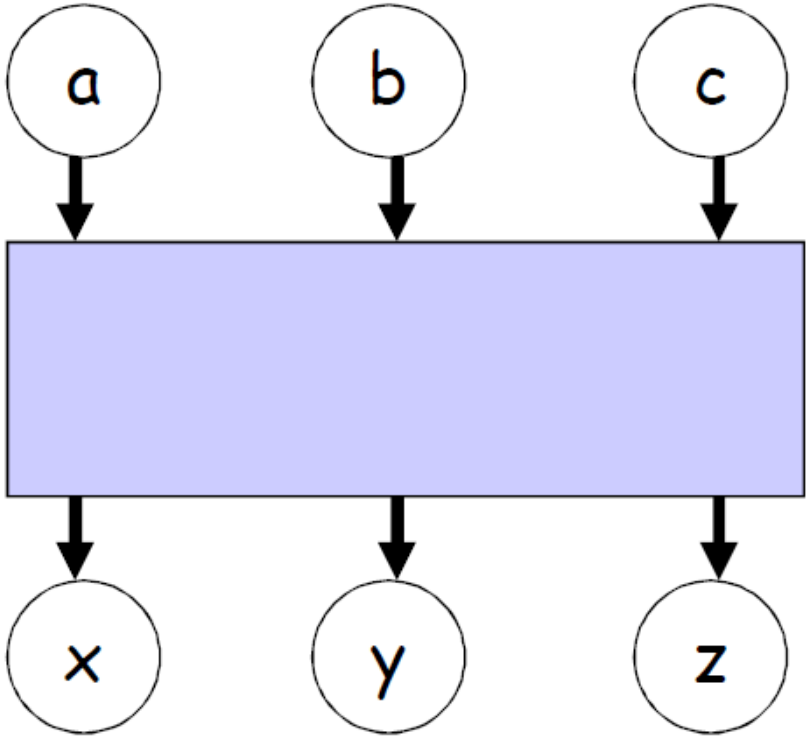


The Fundamental Problem of Testing Software

- We cannot test for everything
- No system can be completely tested
- The need to have a clever testing methodology
- **Tests must be carefully designed**

Inputs and Outputs

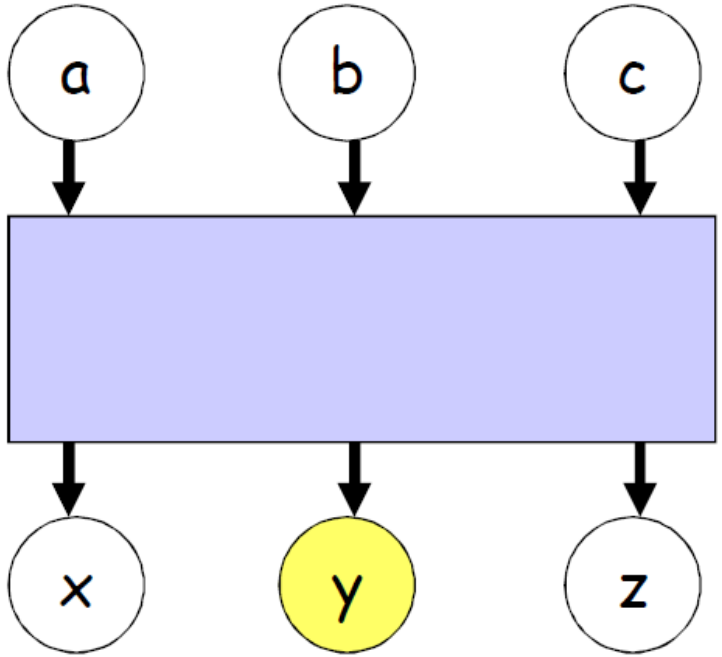
- Do you base your tests on inputs or outputs?



- Start from critical **OUTPUTS** (cf. risk analysis)

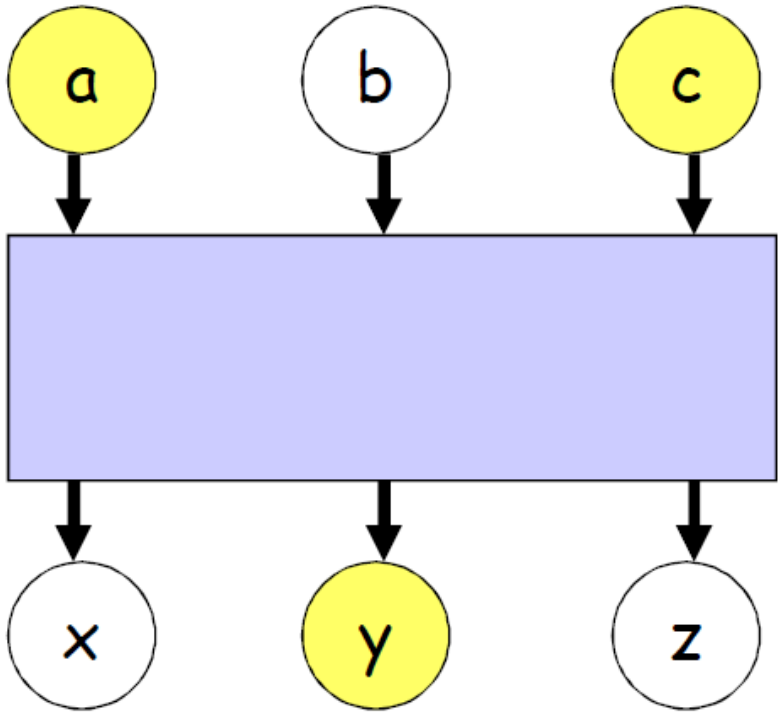
Inputs and Outputs

- Do you test all the inputs for a given output?



- Ideally no, find **DEPENDENCIES** (i.e. inputs on which outputs depend)

Dependencies



- What do we need to know about them?
- How do we find them?
- How do we know we have got them right?

Dependencies: What Do We Need to Know About Them?

- Which inputs count for each output
  - So that we can vary just the inputs that count
  - We can do it in a systematic way
  - We don't miss any test cases
  - We avoid redundant tests
- Do we need to understand HOW an output depends on its inputs?
  - No, only WHAT the inputs are
  - Complete dependency testing is non-discriminatory and therefore less informative than selective testing

Dependencies: How Do We Find Them?

- Who knows the dependencies?
  - The USERS
- Problems with users?
  - They often know TOO MUCH – you must be selective
  - They would like to tell you HOW outputs depend on the inputs – you don't want to know
- Other sources of information
  - Specifications
- Software code itself?
  - Should be avoided if at all possible: it is the CODE that we are TESTING

Dependencies: How do we know we have got them right?

- Carry out a preliminary VERIFICATION
  1. For each output, select all the inputs that Should Not Matter (SNMs)
  2. Set each of the other inputs to the same (random) value within the valid range
  3. Run the code once and save the answer
  4. Run a testing harness which randomly varies the SNMs within the valid ranges
  5. Compare the outputs. If the values of SNMs really do not matter, you should always get the same output
  6. Log all the variables that do change the outputs and verify whether they should or should not matter

Dependencies and Inputs

```
CASE 1
    Total = amount * tax_rate
```

```
CASE 2
    if (amount <= 1800)
          Total = amount
    else if (amount > 1800 .AND. amount < 15000)
          Total = amount * tax_rate[1]
    else
          Total = amount * tax_rate[2]
```

- Is dependency of the output ‘Total’ on the input ‘amount’ the same in both cases?
- What’s the key difference?

Dependencies and Inputs

- Three types of input
  - ❖Should Not Matter Variables (SNMs)
  - ❖Results-Only Variables (ROVs)
  - ❖Gate Variables (GVs)
- Testing
  - ❖SNMs – No testing (i.e. verification only)
  - ❖ROVs – Single value + Boundary + Out-of-range tests
  - ❖GVs – Test should cover all the paths (Path Testing)

Dependencies and Inputs: QUIZ 1

```
CASE 2
    if (amount <= 1800)
            Total = amount
    else if (amount > 1800 .AND. amount < 15000)
            Total = amount * tax_rate[1]
    else
            Total = amount * tax_rate[2]
```

- We are testing an output 'Total'
- What kind of input (SNM, ROV or GV) is
  - Amount?
  - Tax_rate?
- Why?

Dependencies and Inputs: QUIZ 2

CASE 3

Dialled_area_code = <user-entered area code>

Phone_number = <user-entered destination phone number>

User_area_code = <obtained automatically>

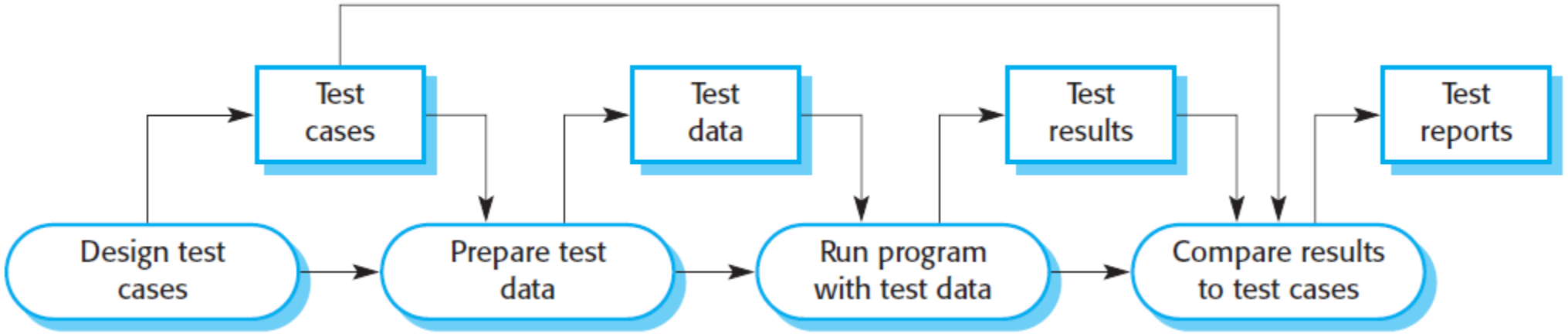
Full_destination = concatenate(Dialled_area_code, Phone_number)

Call_cost_rate = ComputeRate(User_area_code, Dialled_area_code)

- What kind of input (SNM, ROV or GV) is
  - Dialled_area_code?
- Why?

Functional Testing

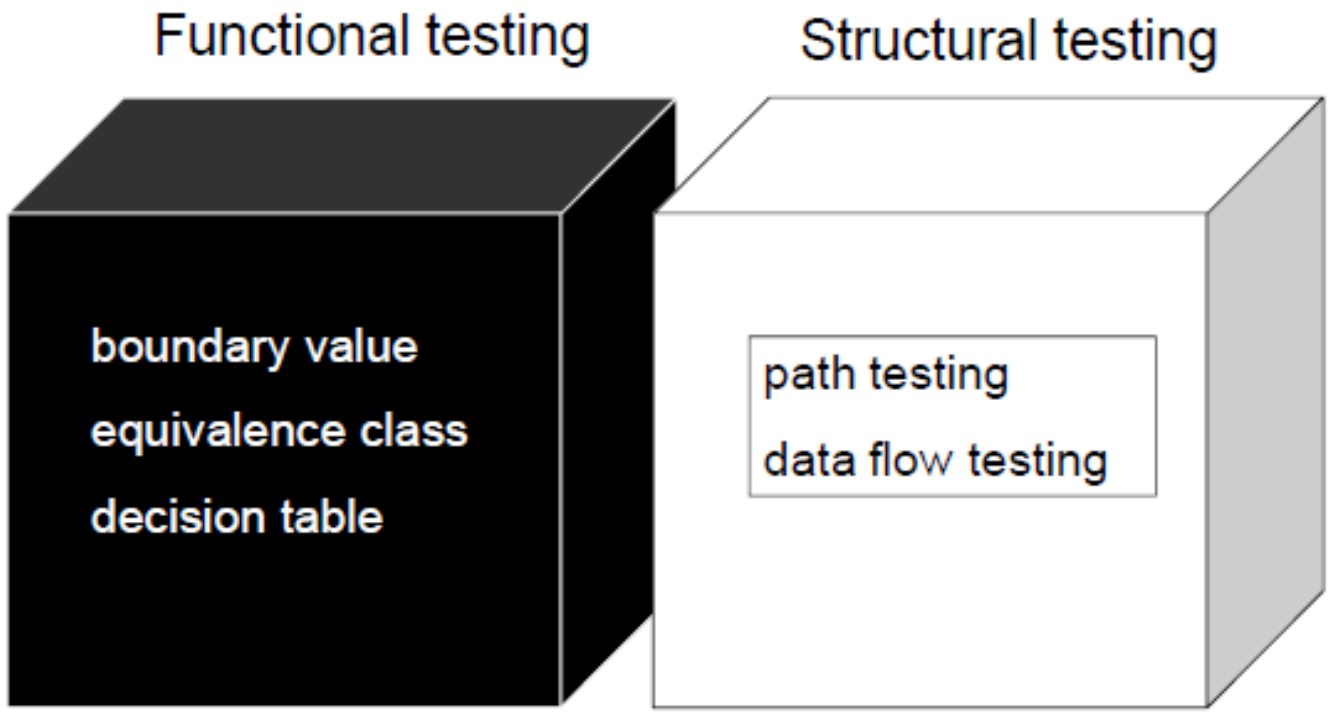
- Dijkstra (1972): “Testing can only show the presence of errors, not their absence”.
- An abstract model of the traditional testing process:



Functional Testing

- **Test cases** are specifications of the inputs to the test and the expected output from the system (the test results), plus a statement of what is being tested.
- **Test data** are the inputs that have been devised to test a system. Test data can sometimes be generated automatically, but automatic test case generation is impossible.
- People who understand what the system is supposed to do must be involved to specify the **expected test results**. However, test execution can be automated.
- The **test results** are automatically **compared** with the **predicted results**, so there is no need for a person to look for errors and anomalies in the test run.

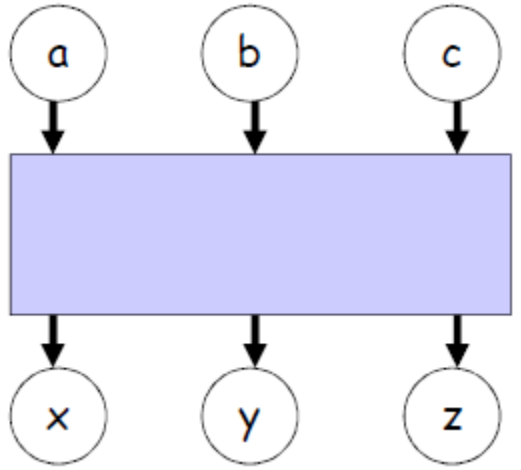
Functional Testing Techniques



Boundary Value Testing

Boundary Value Analysis

- Any software to be tested can be thought of as a function that associates its inputs with its outputs.

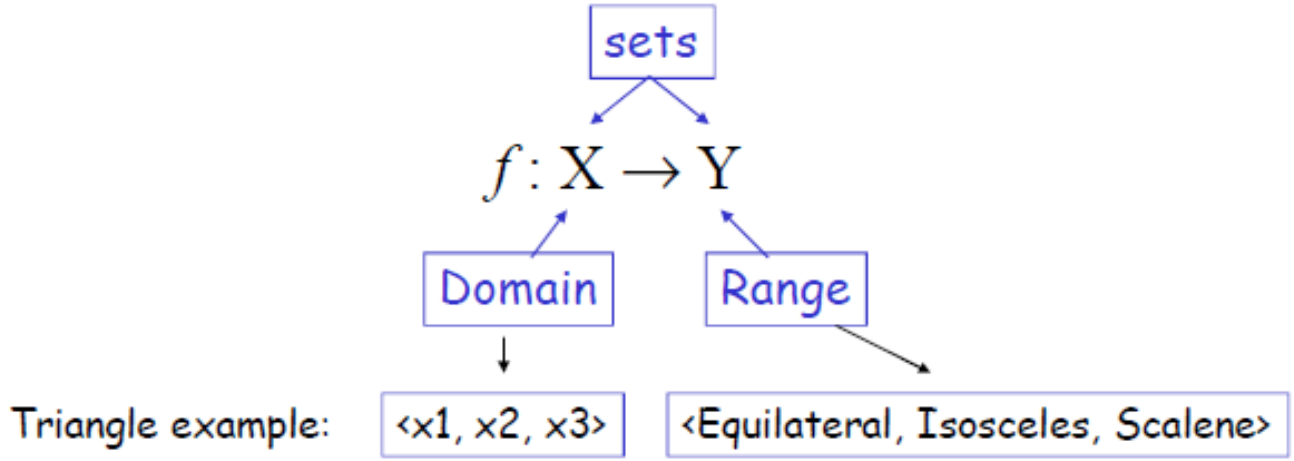


The Triangle Example

- The triangle program accepts three integers, a, b, and c, as input. These are taken to be sides of a triangle. The integers a, b, and c must satisfy the following conditions (c):
  - c1. $1 \leq a \leq 200$
  - c2. $1 \leq b \leq 200$
  - c3. $1 \leq c \leq 200$
  - c4. $a < b + c$
  - c5. $b < a + c$
  - c6. $c < a + b$
- The output of the program is the type of triangle determined by the three sides: Equilateral, Isosceles, Scalene, or NotATriangle.
  - If an input value fails any of conditions c1, c2, or c3, the program notes this with an output message, for example, “Value of b is not in the range of permitted values.”
  - If values of a, b, and c satisfy conditions c4, c5, and c6, one of four mutually exclusive outputs is given:
    1. If all three sides are equal, the program output is Equilateral.
    2. If exactly one pair of sides is equal, the program output is Isosceles.
    3. If no pair of sides is equal, the program output is Scalene.
    4. If any of conditions c4, c5, and c6 is not met, the program output is NotATriangle.

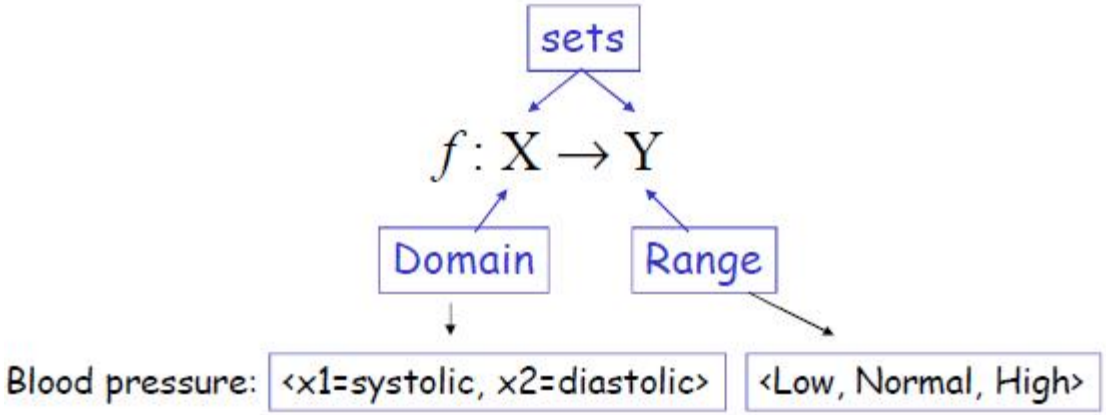
Boundary Value Analysis

- Any software to be tested can be thought of as a function that associates its inputs with its outputs

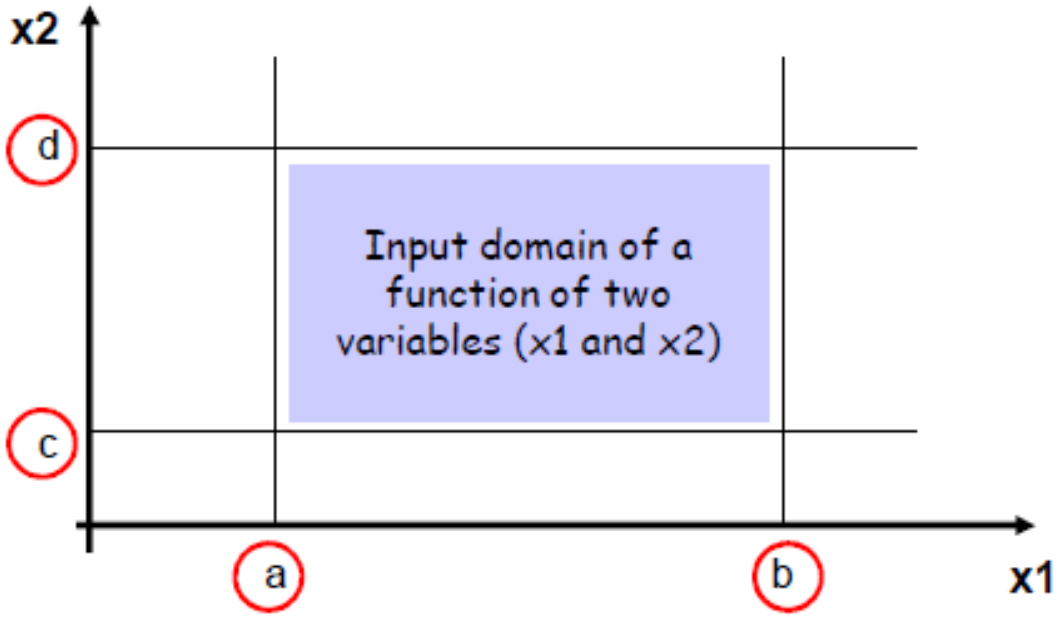


Boundary Value Analysis

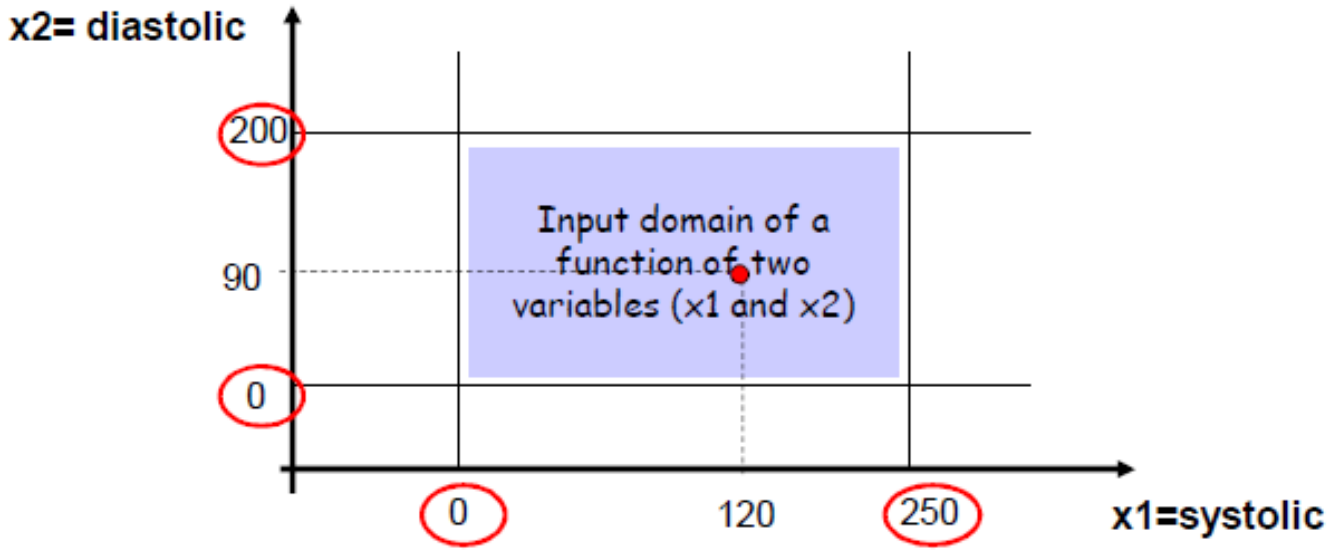
Blood pressure



Boundary Value Analysis



Boundary Value Analysis



Boundary Value Analysis

- A greater number of errors occurs at the boundaries of the input domain rather than in the “centre.” It is for this reason that boundary value analysis (BVA) has been developed as a testing technique.
- There are four variants of BVA:
  - Normal boundary value testing
  - Robust boundary value testing
  - Worst-case boundary value testing
  - Robust worst-case boundary value testing

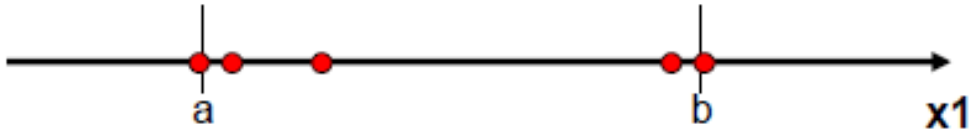
Normal Boundary Value Testing

Normal Boundary Value Testing

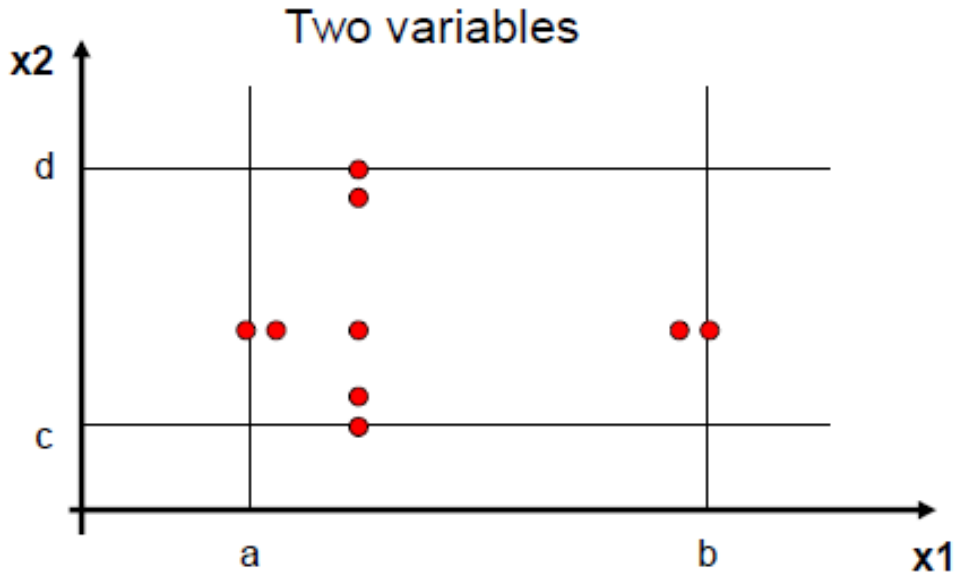
- The rationale behind boundary value testing is that errors tend to occur near the extreme values of an input variable.
- Loop conditions, for example, may test for < when they should test for ≤, and counters often are “off by one.” (Does counting begin at zero or at one?)
- The basic idea of boundary value analysis is to use input variable values…
  - at their minimum,
  - just above the minimum,
  - a nominal value,
  - just below their maximum,
  - and at their maximum.

Normal Boundary Value Testing

Single variable



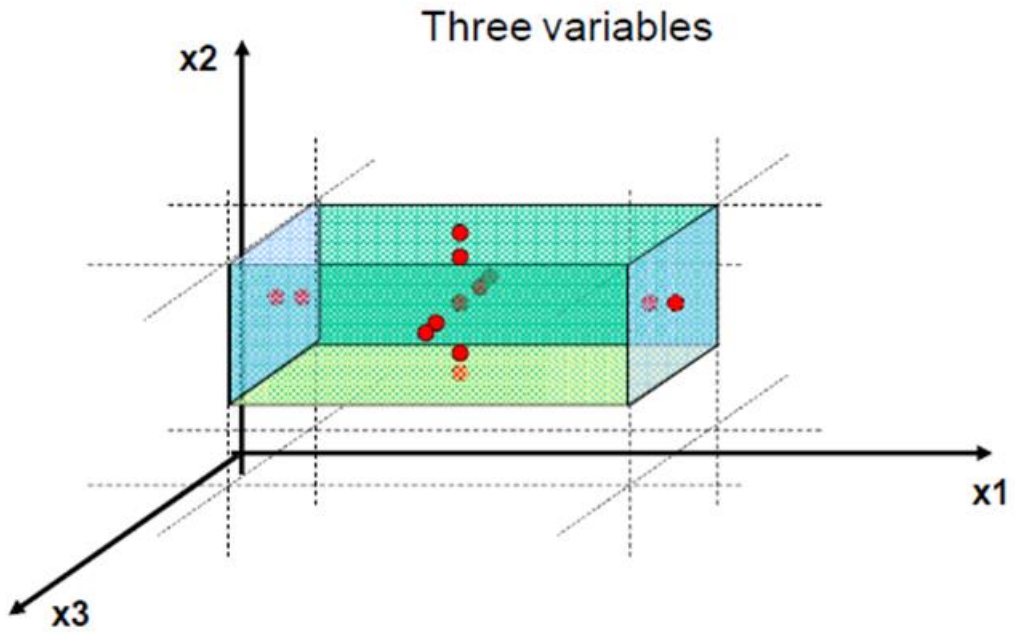
Normal Boundary Value Testing



How many boundary tests for 2, 3, …, n variables?

4n + 1

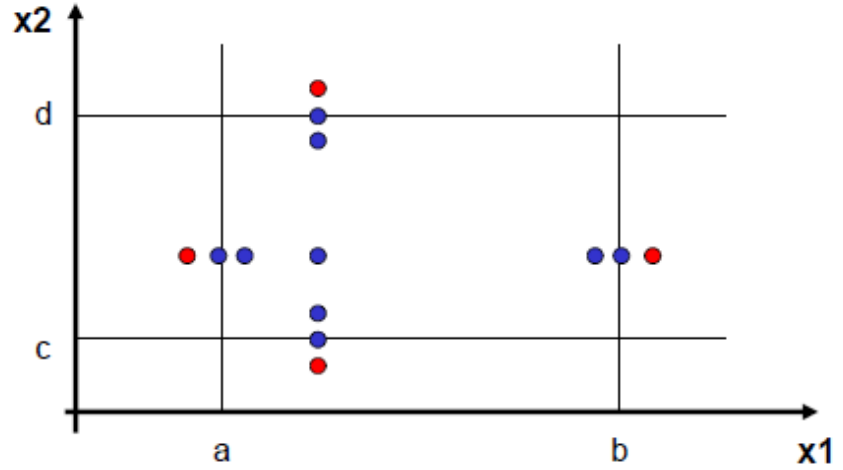
Normal Boundary Value Testing



Robust Boundary Value Testing

Robust Boundary Value Testing

- Robust boundary value testing is a simple extension of normal boundary value testing.
- In addition to the five boundary value analysis values of a variable, we see what happens when the extremes are exceeded with a value slightly greater than the maximum (max+) and a value slightly less than the minimum (min–).

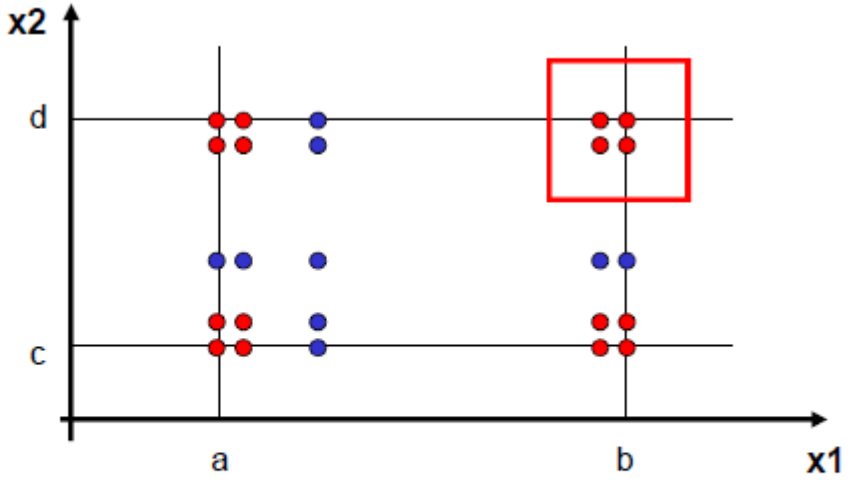


Worst-Case Boundary Value Testing

Worst-Case Boundary Value Testing

- Boundary Value Analysis makes the “single fault” assumption:
  - “Failures are only rarely the result of the simultaneous occurrence of two or more faults”.
- Worst case testing is interested in what happens when more than one variable has an extreme value simultaneously.
- Best used when variables may have complex interactions and where the impact of failure is large.

Worst-Case Boundary Value Testing



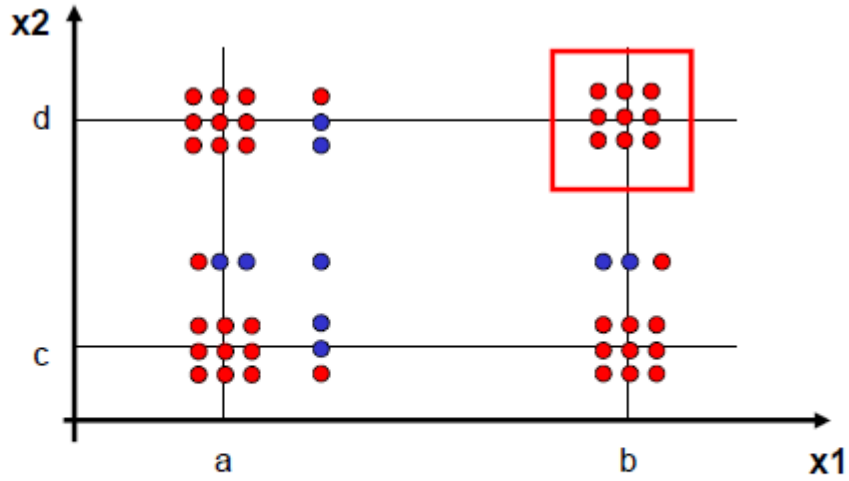
- How many boundary tests for 2, 3, …, n variables?

5^n

Robust Worst-Case Boundary Value Testing

Robust Worst-Case Boundary Value Testing

- It involves the Cartesian product of the seven-element sets we used in robustness testing.
- Also known as the paranoid testing.



- How many boundary tests for 2, 3, …, n variables?

7^n

Special Value Testing

Special Value Testing

- Tester uses the domain knowledge and experience to select test values
- Ad hoc and not based on any principles
- The least systematic and least uniform of the methods
- Can be of value when used in addition to one of the systematic methods

Random Testing

Random Testing

- Random values are picked from the specified ranges
- The main question: how many values are sufficient?

Limitations of Boundary Value Analysis

Limitations of Boundary Value Analysis

- Boundary value analysis works well when the program to be tested is a function of several **independent variables** that represent **bounded physical quantities**.
- Bounds may not be always obvious (e.g. an upper bound for the triangle problem)
- Bounds may not be appropriate (e.g. for Boolean values)
- No bounds in cases of enumerated variables (e.g. in bank transactions “deposit”, “withdrawal”, “query”; or sets of car colours)
- No testing for negative cases
- Inadequate when there are dependencies between the variables. (e.g. the month, day, and year in an analysis of test cases for NextDate problem).
- It provides **no consideration for the nature of the function, or the semantic meaning** of the variables. We see boundary value analysis test cases to be rudimentary because they are obtained with very little insight and imagination.

Next On

Take Away Points

- Verification and Validation?
  - What is the difference between verification and validation?
  - Activities included in verification and validation.
- Functional Testing
  - The abstract model of the traditional testing process.
- Boundary Value Analysis (BVA)
  - Comparison and contrast of each of the BVA variants.
  - What does the single fault assumption say?
  - Which are its limitations?

Acknowledgement: Part of the material in these slides is a courtesy of Dr Ela Claridge

Further Reading

- Ian Sommerville, “Software Engineering”
  - Chapter 8: Software Testing
- Pressman and Maxin, “Software Engineering: A Practitioner’s Approach”
  - Chapter 22: Software Testing Strategies
  - Chapter 23: Testing Conventional Applications
- Jorgensen, “Software Testing”
  - Chapter 2: Examples
  - Chapter 5: Boundary Value Testing
- Chat et al., “Handbook of Software Engineering”
  - Software Testing by Gordon Fraser and José Miguel Rojas

Next Lecture

- Equivalence classes