6. Equivalence class testing

ROVs and GVs
Equivalence classes
Path coverage and code coverage
Strong and weak testing
Dependencies and their exploitation

ROVs and GVs

- Boundary value analysis *implicitly* assumed that variables are ROVs (Results Only Variables)
- Implications:
  - Variables were treated as independent
  - Single fault assumption likely to hold
- When dealing with GVs (Gate Variables) this assumption is not true

ROVs and GVs

Output variable: Total

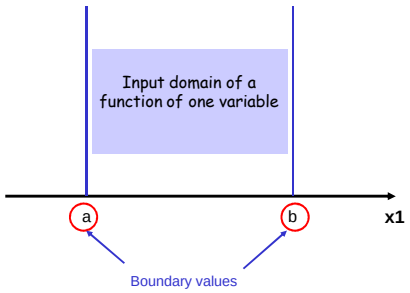
```
if (amount <= 1800)
      Total = amount
else if (amount > 1800 .AND. amount < 15000)
      Total = amount * tax_rate[1]
else
      Total = amount * tax_rate[2]
```

3 paths

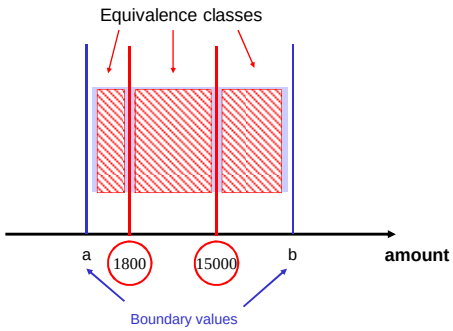
ROVs and GVs

- When dealing with GVs, single value testing is likely to be insufficient
- In the example above we need to test three cases:
  - amount <= 1800
  - 1800 < amount < 15000
  - amount >= 15000
- We say that the variable “amount” has three equivalence classes
- For the time being we ignore out-of-range values

Boundary Value analysis



Equivalence classes



Equivalence classes

Definition

Two input values are in the same equivalence class if they are treated in the same way according to the specification,

i.e they cause the same path to be executed.

Aims

- Equivalence class testing strategy is developed with the following aims
  - To test as few input values as possible
  - To spot and eliminate redundant tests
  - To tell how many tests are necessary to test a given piece of software to a known level

Equivalence class testing

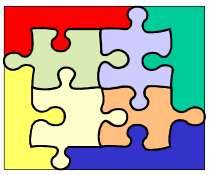
- We need to test only one value from each equivalence class; testing more would be redundant
- Equivalence classes help us to design tests which ensure
  - Completeness
  - Non-redundancy

Domain set A

Equivalence class testing

- We need to test only one value from each equivalence class; testing more would be redundant
- Equivalence classes help us to design tests which ensure
  - Completeness
  - Non-redundancy

Domain set A



Completeness

$A = A1 \cup A2 \cup \ldots \cup A8$

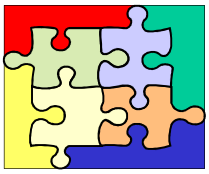
Non-redundancy

$i \neq j \Rightarrow Ai \cap Aj = \varnothing$

Equivalence class testing

Note that definition of the Equivalence Classes is based on the Range (i.e. in relation to the outputs) but variables to be tested are drawn from the Domain (i.e. are the input values).

Domain set A



Example problem

14.3.5 Calculating the Correction

The Correction is determined by the Base, by whether the Base is new, and by the value of Type.

If the Base is less than 10,000, use Method 1 to calculate the Correction. If the Base is at least 10,000 but not greater than 50,000 use Method 2. Otherwise, use Method 3.

Each Method will use procedures that depend on whether the Base is new or is a previous Base. (That is, "New" = Y or N).

The value of Type can be A, B, C, or X. Each Method uses the Type to determine what process to use for computing the Correction.

Example problem

Calculating the Correction

Base	New	Type

Example problem

Calculating the Correction

Base	New	Type
<10,000	Y	A
10,000 – 50,000	N	B
>50,000		C
		X

Example problem: How many test cases?

- **All combinations**

  3 x 2 x 4 = 24

- Testing including all the combination is called Path coverage
  - Equivalent to setting the "gates" in all possible combinations
  - Referred to as "strong testing"

Example problem: How many test cases?

- **Minimum number of cases to "get away with"**

  max (3, 2, 4) = 4

- Called Code coverage
  - Ensures "visiting" each line of code once, but without considering how you got there and where you are going next.

Definitions

- **Path Coverage**

  A set of test achieves Path Coverage if it contains at least one test that goes down **each path** in the piece of code under test

- **Code coverage**

  A set of tests achieves Code Coverage if it contains at least one test that executes **each line of code** in the piece of code under test.

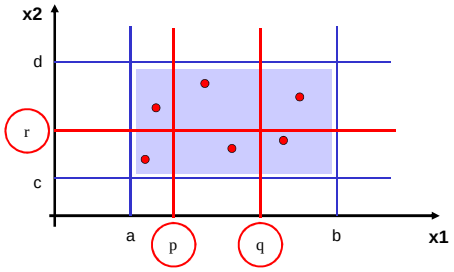
Strong and weak testing

- Strong testing = Path Coverage
- Maximum number of tests (excluding out-of-range values)

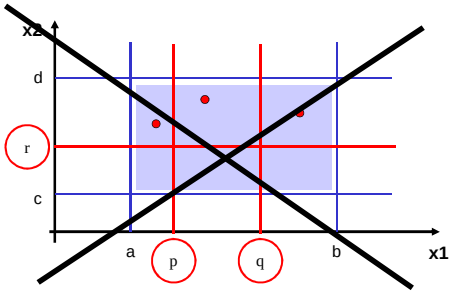
- Weak testing = Code Coverage
- Minimum number of tests (excluding out-of-range values)

- Defines the bounds for the number of tests (excluding out-of-range values)

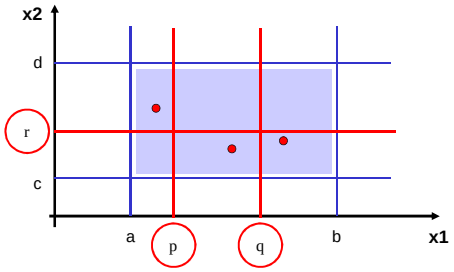
Strong normal equivalence class test cases (Path Coverage)



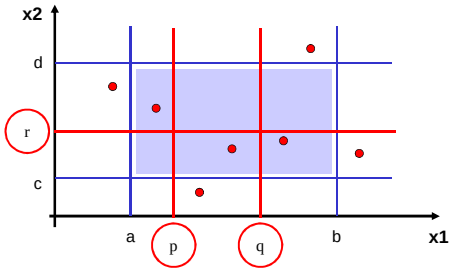
Weak normal equivalence class test cases (Code Coverage)



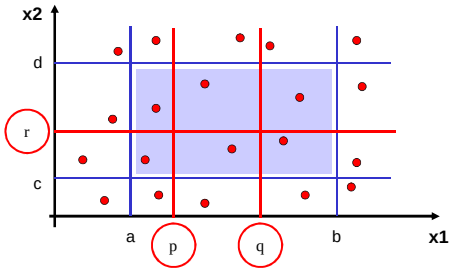
Weak normal equivalence class test cases (Code Coverage)



Weak robust equivalence class test cases



Strong robust equivalence class test cases



Equivalence classes - benefits

Help to determine

- How many test cases we need
  - Strong and weak cases
- When we have redundant tests
  - The number of tests exceeds the number defined by the Strong Test
- How to allocate testing effort based on Risk
  - We can see, for each output, how many tests would be required to test it thoroughly.
  - We can estimate how thoroughly we have tested each output:
  - ratio of the # tests planned/executed and the # tests ideally required for a given output.

Equivalence classes - weaknesses

- The use of strong equivalence may sometimes cause redundancy

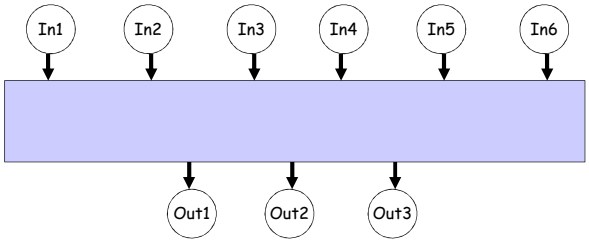
In what cases?

- When dependencies between the input and output variables are ignored.

Exploiting dependencies

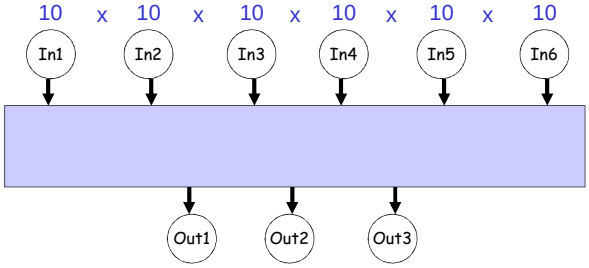
- Information about dependencies can be used to reduce the number of test cases
- This is in addition to the reductions achieved by
  - Not testing SNM variables
  - Using Equivalence Classes to test values from inputs that do matter.

Example



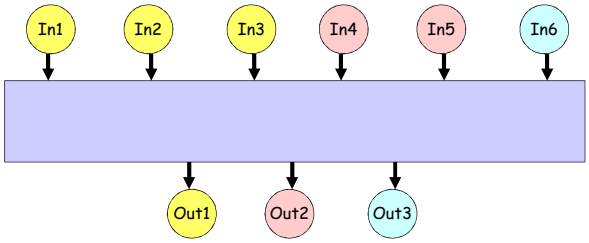
- Each input has 10 valid equivalence classes (boundary and out-of-range values ignored)

How many tests for complete coverage?



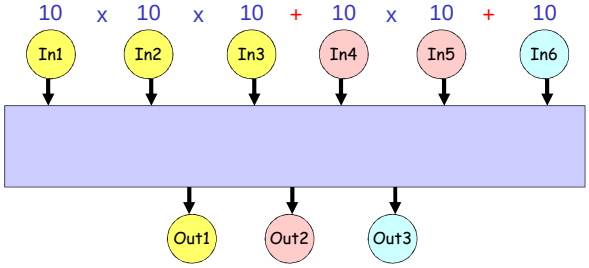
3 x 10 x 10 x 10 x 10 x 10 = **3,000,000**

Dependencies



Out1-:{In1, In2, In3}, Out2-:{In4, In5}; Out3-:{In6}

How many tests for complete coverage?



10 x 10 x 10 + 10 x 10 + 10 = **1,110**

Merits of knowing dependencies

- Outputs should be tested one at a time
- For a given output vary only those inputs that are relevant

- This dramatically reduces the number of cases

- In the example above from 3,000,000 to 1,110 – this is a reduction by 3 orders of magnitude!

Next lecture

Exploiting dependencies – Dependency Islands

Homework

- Define the equivalence classes and specify test cases for Strong Normal Equivalence Tests for the NextDate program.

- Study the problem description "Credit Approval on Orders"
  - Find the valid Equivalence Classes for all the input variables that contribute to the output variable Credit_Status. Assume that all amounts have been rounded off to the nearest pound.
  - How many tests will be required for the Path Coverage Testing?
  - How many tests will be required for the Code Coverage Testing?

- If an input is used to compute more than one output, do you think it could have different Equivalence Classes (ECs) for the different outputs, or are the ECs the same?

- Compute the percentage reduction of the number of test cases in the example above, when each of the input variables has 3 equivalence classes (this is more realistic than 10 equivalence classes we assumed).

Further reading

- Craig and Jaskiel "Systematic Software Testing"
  - Chapter 5: Analysis and Design, section "Equivalence partitioning"

- Search web pages for more information and interesting examples (e.g. http://www.kaner.com/pdfs/ has many interesting articles and slides, such as "What is a good test case?"). There is a lot of relevant and interesting information on the web and it is up to you to find out and explore. I have provided the initial links from which this information can be accessed, but you must do all the legwork (remember the first lecture and the mountain guide?)