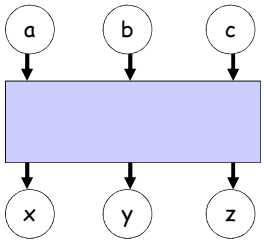
4. Test case design

Inputs and outputs
Dependencies
Test plan
Test Design Specification
Test Case specification

The fundamental problem of testing software

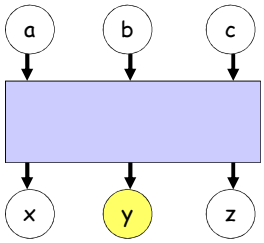
- We cannot test for everything
- No system can be completely tested
- The need to have a clever testing methodology
- Tests must be carefully designed

Inputs and outputs



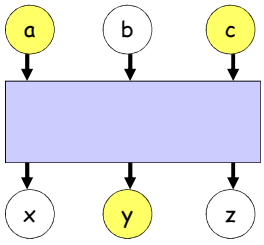
- Do you base your tests on inputs or outputs?
- Start from critical **OUTPUTS** (cf. risk analysis)

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**
  - For each output, select all the inputs that Should Not Matter (SNMs)
  - Set each of the other inputs to the same (random) value within the valid range
  - Run the code once and save the answer
  - Run a testing harness which randomly varies the SNMs within the valid ranges
  - Compare the outputs. If the values of SNMs really do not matter, you should always get the same output
  - 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

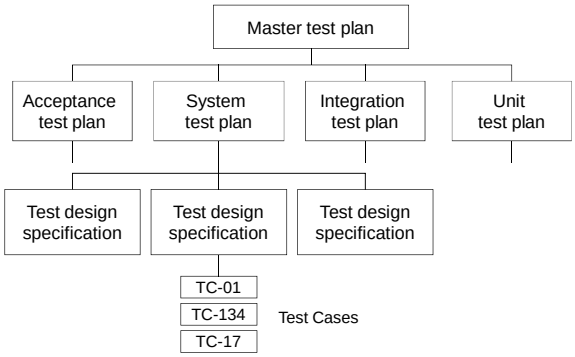
```
CASE 3
    Dialled_area_code = <user-entered area code>
    Phone_number = <user-entered destination phone number>
    User_area_code = <obtained automatically>
    Full_destination = concatenate( Dialled_area_code, Phone_number)
    Call_cost_rate = ComputeRate( User_area_code, Dialled_area_code)
```

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

Test planning - context

- Test plan – a major communication channel with all project participants
- Test planning – a PROCESS that leads to a document which describes testing priorities and their execution
- The goal of test planning – NOT a list of test cases, but the MEANS of dealing with issues of testing strategy, resource utilisation, responsibilities, risks and priorities

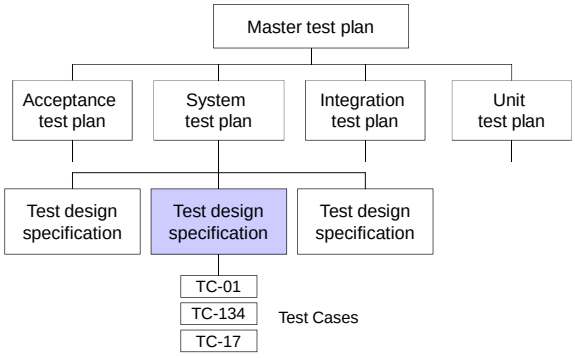
Test plan



Master Test Plan

- Place in the Software Lifecycle Model
  - To be developed alongside with the requirement specifications and the project plan
  - Accept that it may change during the course of the software development and testing
- Contents of the Test Plan
  - Use the templates
  - IEEE Standard 829-1998 for Software Test Documentation
  - see various documents at the module web site at http://www.cs.bham.ac.uk/~exc/Teaching/STesting/Web_resources.html
    - http://www.cs.bham.ac.uk/~exc/Teach/STesting/StandardTestPlan.doc
    - http://readysetpro.com/whitepapers/testplans.html
  - Links from http://www.aptest.com/resources.html

Test plan



Test Design Specification

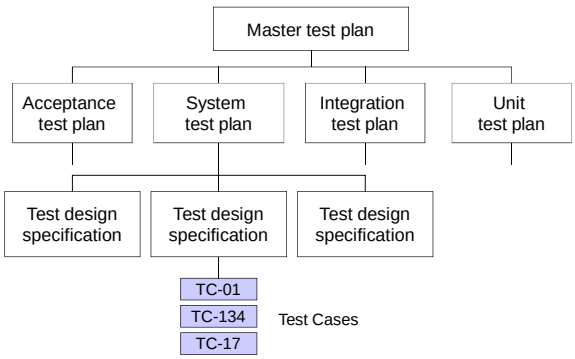
IEEE Template

1. Test design specification identifier
2. Feature(s) to be tested
3. Approach refinement
4. Test identification
5. Feature pass / fail criteria

Test Design Specification - example

- Test design specification identifier
  - TDS-04-10-11-B
- Feature(s) to be tested
  - Withdraw cash
  - Check account balance
- Approach refinement
  - Withdrawal not allowed between 2.00 a.m. and 4.00 a.m.
- Test idenfication
  - TC01 – withdraw £20 from a valid account with £200
  - TC17 – withdraw £100 from a valid account with £100
  - . . .
- Feature pass / fail criteria
  - All withdraw cash tests must pass
  - 90% of check account balance must pass

Test plan



Test Case Specification

IEEE Template

1. Test case specification identifier
2. Test items
3. Input specifications
4. Output specifications
5. Environmental needs
6. Special procedural requirements
7. Inter-case dependencies

Test Case Specification

- Test case specification identifier
  - TC01: withdraw £20 from a valid account with £200
- Test items
  - Requirement spec RS-04-09-11
  - Software code: Cash_withdraw(), Balance_Check()
- Input specifications
  - 20
- Output specifications
  - 180
- Environmental needs
  - Driver_D_11, Stubs: Set_account_balance()
- Special procedural requirements
  - Convert pounds into pence
- Inter-case dependencies

Simple Test Case Specification

Spreadsheet based

Test cases	Special Notes	Inputs				OK Results			
		Var 1	Var 2	Var 3	. . .	Var 1	Var 2	Var 3	. . .
TC001									
TC002									
TC117									
TC023									
. . .									

Test Design Specification Triangle example

- Test design specification identifier
  - TDS-04-10-11-B
- Feature(s) to be tested
  - Input validation
  - Triangle type determination
- Approach refinement
  - N/A
- Test idenfication
  - TC01 – First side of triangle in correct range
  - TC17 – Triangle is equilateral
  - . . .
- Feature pass / fail criteria
  - All conditions specified for input variables must be passed
  - All triangle types must be determined correctly

Test Case Specification Triangle example

- Test case specification identifier
  - TC17 – Triangle is equilateral
- Test items
  - Requirement spec RS-04-09-11
  - TriangleType()
- Input specifications
  - First side = 20; Second side = 20; Third side = 20
- Output specifications
  - 'Equilateral'
- Environmental needs
  - Driver_D_11, Stubs: ValidateInput()
- Special procedural requirements
  - Round input to the nearest integer
- Inter-case dependencies

Simple Test Case Specification Triangle example

Spreadsheet based

Test cases	Special Notes	Inputs				OK Results
		a	b	c	. . .	TriangleType
TC017		20	20	20		"Equilateral"
TC018						
TC117						
TC023						
. . .						

Home exercises

- Access the "IEEE Standard 829-1998 for Software Test Documentation" on the course resources web page
- Study 'Test design specification' (p.6) and the related example (A.2.2, p.35)
- Study 'Test case specification' (p.7) and the related example (A.2.3, p. 38)
- Prepare the Requirement Specification and the Test Case Specification for the TriangleType problem based on the handout "The Triangle Problem 2" using the appropriate IEEE standard templates.
- Test the Triangle program using the Test Case Specification (see also the class exercise on "Test Case Design – the triangle problem").

  The program Naive.exe (executable on a PC) can be downloaded from the course web page:

  http://www.cs.bham.ac.uk/~exc/Teaching/STesting/index.html

Further reading

- Craig and Jaskiel "Systematic Software Testing"
  - Chapter 3: Master Test Planning
  - Chapter 5: Analysis and Design, in particular section on "Test Design Documentation"

Next lecture

Boundary value testing