4478 INTRODUCTION TO INFORMATION TECHNOLOGY
8936 INTRODUCTION TO INFORMATION TECHNOLOGY G

Dr. Julio Romero

Faculty of Science and Technology

UNIVERSITY OF CANBERRA

Software Application Testing

Acknowledgment

- The following content has been adapted from Professor Roland Goecke software testing material. University of Canberra, October 2020.

Testing

The software coding is done.
We are trying to ensure it works correctly.

- Complementary requirements for correctness: ***Verification*** and ***Validation***
- ***Validation***
  - Are we building the right product?
  - Are the specifications right?
- ***Verification***
  - Are we building the product right?
  - Have the specifications been implemented?
- Up to 50% of development costs spent on validation & verification. By ***Testing***, we mean verification.

Tests

- Testing is the ***systematic*** attempt to find errors in a planned way
- An error causes a system to deviate from its intended behaviour
- Testing attempts to find differences between reality and requirements
- ***Tests are falsification attempts***
  - Or in plain English, we are systematically trying to break the programme code

Components of a Test

- A test has 3 components
  1. Data
  2. Operation
  3. Expected result
- The ***Operation*** may be a single function or an extended process of several functions
- In IIT / IIT G, it will just be a single function

Responsibility Based Testing

- Based on requirements
- Testing ***independent*** of implementation
  - In practise, there is often a testing team in a project that is completely separate from the programming team
  - This could be an in-house testing team or a one at the client's side
- **Tests**
  - Created as part of requirements analysis
  - Implemented ***before*** coding!
    - That is, before you even start writing a single line of code, the tests have already been written solely based on the requirements document

Testing Techniques

- *Example:* "The army will only enlist people between the ages of 18 and 40 inclusive."
- This gives three possibilities
  - Below 18 years of age
  - Between 18 and 40 years inclusive
  - Older than 40 years

Equivalence Partitions

- The three ranges [0, 17], [18, 40], [41, 99] are the ***equivalence partitions*** for this example
- Equivalence partitions depend on the business rules
- Each value in an equivalence partition tests the same business rule
  - For example, 15 and 16 are in the same equivalence partition because ages of 15 and 16 are both too young
- You should include at least one test from each equivalence partition
  - This is one point we will look for in the Software Testing lab test in week 14

Testing for Different Data Types

- When preparing tests, we need to think about what values we should use in our test cases
- These depend on the type of data our software takes
- Generally, we distinguish between
  - ***Ordinal data***, and
  - ***Non-ordinal data***

Ordinal Data and Non-Ordinal Data

- ***Ordinal*** data types are the ones where $<$, $\leq$, $>$ and $\geq$ have meaning
  - The most common ordinal data types are numbers (age, currency, distance, ...) and dates
- ***Non-ordinal*** data types include
  - Binary data
    - Raining? (yes / no)
    - Weekend? (yes / no)
    - Gender (f / m)
  - Discrete data
    - Colour (red, green, blue, yellow, brown, ...)
    - Dog breed (poodle, labrador, ...)
    - Pay rate (normal | overtime | Sunday)

Ordinal Data and Boundary Values

- If your data is ordinal, and has a range of values, you should ***push the boundaries*** in your tests
- In the army example, your tests should include 17 and 18, and 40 and 41. This gives

Description	Age	*Enlist()*
Just too young	17	No
Just old enough	18	Yes
Just young enough	40	Yes
Just too old	41	No

- You could also include 0 and 150 years (or whatever you want to use as maximum age) as boundary values

Planning Your Tests

- Analysis → description → test
- Coverage
  - Cover all equivalence partitions
  - Test special cases, such as boundary conditions (ordinal data)
- Description
  - Proof that you have planned your tests
  - Unique for each test
  - Avoid numbers in description
  - Before the data (numbers)
- Tests
  - Concrete – contains data (numbers)
  - Comes from description

How Many Tests?

- ***Tests are not for free!***
  - The two most common costs associated with tests are
    - Time (it costs to pay the testers)
    - Loss of revenue (because the s/w is not running or can't be sold yet)
- You are unlikely to ever be able to test all possibilities
- We want enough tests, but not too many
  - We want the tests that are absolutely necessary
  - We don't want unnecessary tests that test the same condition as another test
- Too many may indicate insufficient planning

How Many Tests?

- If two tests have same description, why do you need both?
- Diminishing returns. You are trying to find errors.
  - If the army application gives correct results for ages 15 and 17, do you need to test age 16?
- *Example*:
  Army: Which is a better set of tests?
  - Ages 4, 10, 13, 16, 23, 27, 33, 36, 45, 46, 61, 67, 78, 81, 82
  - Ages 17, 18, 40, 41

Testing Framework

- After every change to the software, all previous tests should be re-applied
- Changing software can – and does – introduce *new errors*
- Re-applying previous tests is called ***regression testing***