Dynamic Testing Methods

- Black Box (*today’s lecture*)
  - Equivalence Class & Boundary Tests
- White Box (*next lecture*)
  - Statement, Branch and Path

**References:**
- Bruegge & Dutoit, *Object-Oriented Software Engineering*, Prentice Hall 2004

---

**Test Planning Exercise**

- Two rectangles are identified by 8 integer values, giving the Cartesian co-ordinates of the lower left hand corner and upper right hand corner of each rectangle.
- A program is required to calculate whether these two rectangles: overlap, just touch, do not intersect or the input data is invalid.
- 1. Draw 2 or 3 interesting cases for this problem
- 2. Compare your cases with those around you
- 3. When would you consider two different cases for the rectangle problem to be “the same” or “significantly different”?

---

**Equivalence Class Testing**

- Method:
  - Identify the entire input data space of the unit under test
  - Partition this input space into different classes.
  - Select a data element from each class and execute the unit using this input
  - Check that the output is as expected

- The goal of equivalence class testing is to select classes so that the behaviour of the SUT can be assumed to be correct for all data in the class if it is known to be correct for one data element of the class.

---

**Equivalence Class Test Examples**

- Input space: integers from 1 to 100
- Partitions:
  - normal input: [1,100]
  - error input: [-100,0] [101,1000]
- Test cases one value from each class: -1, 42, 200
- Input space: calendar dates
- Partition: several dimensions needed here - valid and invalid dates, 30/31 day months, Februarys in leap years
- Test cases: one from each class

---

**Boundary Value Analysis**

- Special case of equivalence class testing
- Choose equivalence classes as above
- Test data is taken from the extremes of partitions of input and output ranges
- Tests normally performed for values at and either side of each boundary
- Example (see previous page)
  - Input space: integers from 1 to 100
  - Boundary values 0,1,2 and 99,100,101
Unit Test: Equivalence Classes

- **Test Purpose:**
  - to detect output errors for given inputs using representatives of the entire input space

- **Evidence:**
  - a set of inputs offered and outputs observed

- **Assumptions:**
  - if behaviour of SUT is correct for one element in an input class then it is correct for all elements

- **Deduction:**
  - oracle: is output correct for a given input?

- **Verdict:**
  - Code correct for all inputs OR code revision required

Test Planning Exercise Revisited

- Reconsider the classes you identified for the rectangle problem at the beginning of the lecture. Can you now translate these into test data for the rectangle problem using equivalence class testing?
- Invent a faulty implementation to solve this problem.
- Would your test data detect your fault(s)? Swap test data with the people around you - who has the best set of test cases?

Rectangle problem

- Coordinate system: origin at the lower left corner, x-axis points to the right, y-axis points up
- 8 input values: x1, y1, x2, y2, x3, y3, x4, y4
- where (x1,y1) = LL corner of rectangle 1;
  (x2,y2) = UR corner of rectangle 1;
  (x3,y3) = LL corner of rectangle 2;
  (x4,y4) = UR corner of rectangle 2

  => valid inputs: x2 > x1 and x4 > x3 and y2 > y1 and y4 > y3

  => 2^4 = 16 combinations

  => 15 possible combinations – equivalence classes for invalid input data

Rectangle problem (cont.)

- **Equivalence classes for rectangles overlapping at a point:**
  - x3 is between x1 and x2, or
  - x4 is between x1 and x2, or
  - x1 is between x3 and x4, or
  - x2 is between x3 and x4 and
  - y3 is between y1 and y2, or
  - y4 is between y1 and y2, or
  - y1 is between y3 and y4, or
  - y2 is between y3 and y4.

- **Equivalence classes for just touching rectangles:**
  - x3 or x4 is between x1 and x2
  - y3 or y4 is between y1 and y2

  => Cover all input data other than those listed above.

Rectangle problem (cont.)

- **Equivalence classes for rectangles not intersecting:**
  - x3 or x4 is between x1 and x2
  - y3 or y4 is between y1 and y2

- **Equivalence classes for do not intersect rectangles:**
  - Cover all input data other than those listed above.
Recall from the previous lecture…

Unit Testing by Code Inspection

- **Test Purpose:**
  - to detect logical faults in implemented units

- **Evidence:**
  - program code and its specification

- **Assumptions:**
  - correct operating environment for calling the unit

- **Deduction:**
  - Testers reason about the behaviour of the code to determine whether the code produces the correct results.

- **Verdict:**
  - Code ready (for other tests) OR code revision required

Real World Examples:

- Recall the *Place Bid* function from the first assignment:

  **Students** are allocated quantities of tokens that they may use to make bids on tutorial timeslots where they wish to confirm a place. Initially, the price of all tutorials shall be 0. Students wishing to attend that timeslot must now bid tokens of a higher value than the tutorial price.

  - Propose a set of equivalence classes to cover the possible scenarios (your classes may depend on the state of the system as well as the input).

  - Compare your classes to the people around you.