SOFTENG 254:
Quality Assurance

Lecture 5b: Input Space Partitioning

Paramvir Singh
School of Computer Science

Potential Assessment Question

Consider the following method:

```
    public static boolean isPrime(int n) {
A       boolean prime = true;
B       int i = 2;
C       while (i < n) {
D           if (n % i == 0) {
E               prime = false;
            }
F           i++;
        }
G       return prime;
    }
```

Which one of the following is a du-path for prime?

(a) A, B, C, D, F, G

(b) A, B, C, D, E, F, C, G

(c) A, B, C, D, F, C, G

(d) E, F, C, G

Justify your answer.

Agenda

- PAQ
- Developing test suites based on sets of inputs

Previously in SOFTENG 254

- To improve test suites, we need to consider both the artifact we are testing (code) and the quality attribute we are trying to establish (correctness)
- White (clear, glass) box testing — performing tests based on **how** the component is implemented ⇒ artifact focus

— decide on tests based on the code itself
  - E.g., statement coverage

— not good at detecting faults relating to requirements

- Black box testing — performing tests based on **what** the component is supposed to do ⇒ quality attribute focus

— decide on tests based on the requirements independent of the code

— not good at detecting faults in implementation decisions

Input space partitioning

- all test suite creation is about choosing inputs
- for any given test requirement, there will often be a number of inputs that can satisfy it — all of those inputs are essentially equivalent
- why not choose the test requirements according to inputs that are "essentially equivalent"?
- principles:
  - minimise the number of tests
  - maximise the usefulness of each test
- modelling the input space

Example: Triangle Program

- Are these two test cases different?
  - Test Case 1: $(3, 3, 4)$
  - Test Case 2: $(4, 4, 5)$
- If we have test case 1, will adding test case 2 increase the probability of causing the IUT to fail (assuming the presence of a fault in it)?

Example: Triangle Program

- Are these two test cases different?
  - Test Case 1: $(3, 3, 4)$
  - Test Case 2: $(4, 4, 5)$
- If we have test case 1, will adding test case 2 increase the probability of causing the IUT to fail (assuming the presence of a fault in it)?
- Observations:
  - The expected output in both cases is the same
  - It is quite likely that the IUT will behave exactly the same in both cases

Equivalence Partitioning

- Partition possible input values into *equivalence classes*
  - every possible input value is in at most one equivalence class (classes are disjoint)
  - every possible input value is in at least one equivalence class (union of all classes is all possible input values)
- expect all input values in one class to have the “same behaviour” by the IUT, so
  - if one member of the class produces correct behaviour, then expect all other members to do so
  - if one member of the class produces incorrect behaviour, then same
- coverage criterion: there is at least one input value tested from each equivalence class
- if find set of largest possible equivalence classes, then
  - each test case will effectively test the whole class (*maximise effect of test*)
  - need no more tests than the number of classes (in theory) (*minimise number of tests*)

Example: Grading 1

Input: mark expressed as percentage (i.e., integers between 0 and 100 inclusive)

Outputs: Course grade, computed as follows:

- if the mark is greater than 90, then the grade is “Adequate”
- if the mark is greater than 75 but no greater than 90, then the grade is “Ok”
- if the mark is greater than 50 but no greater than 75, then the grade is “Pass”
- if the mark is greater than 40 but no greater than 50, then the grade is “More effort required
- if the mark is no greater than 40 then the grade is “Oh dear”

- input partitions: $[0, 40]$, $[41, 50]$, $[51, 75]$, $[75, 90]$, $[91, 100]$
- Example test suite: 20, 45, 60, 80, 95
- What about < 0 and > 100? One partition or two (or none)?

Example: Grading 2

Inputs: exam score expressed as percentage (i.e., integers between 0 and 100 inclusive), and practical score expressed as percentage.

Outputs: Course grade, computed as follows: A combined score percentage is computed as:

$$\text{combined} = 0.6 * \text{exam} + 0.4 * \text{practical}$$

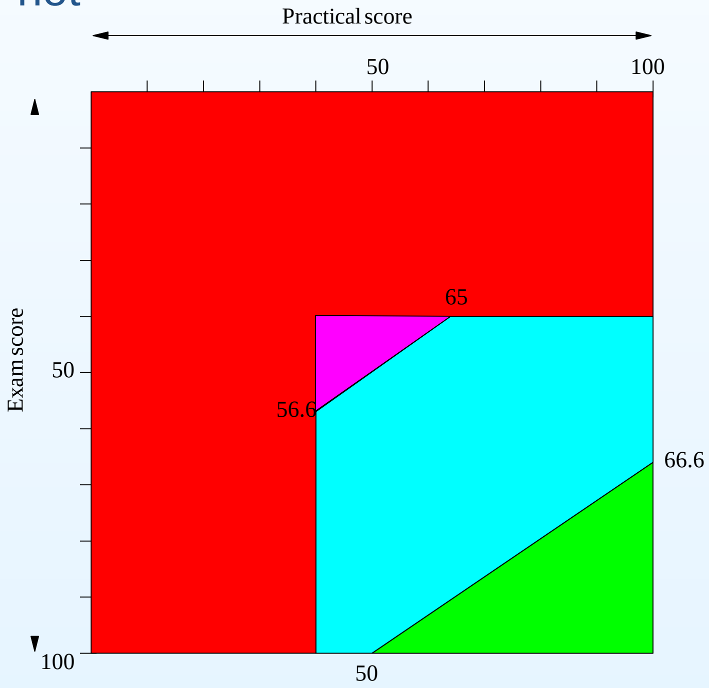
- For combined scores less than 50, the grade is “Fail”.
- For exam scores less than 40, the grade is “Component Fail” (no matter what the practical score is)
- For practical scores less than 40, the grade is “Component Fail” (no matter what the exam score is)
- For combined scores between 50 and 80 inclusive, the grade is “Pass”.
- For combined scores more than 80 and less than or equal to 100, the grade is “Pass with distinction”.
- For all other inputs, the program should report “Invalid Input”.

Based on “Software Metrics”, Fenton&Pfleeger Example 8.19

Grading 2 partitions

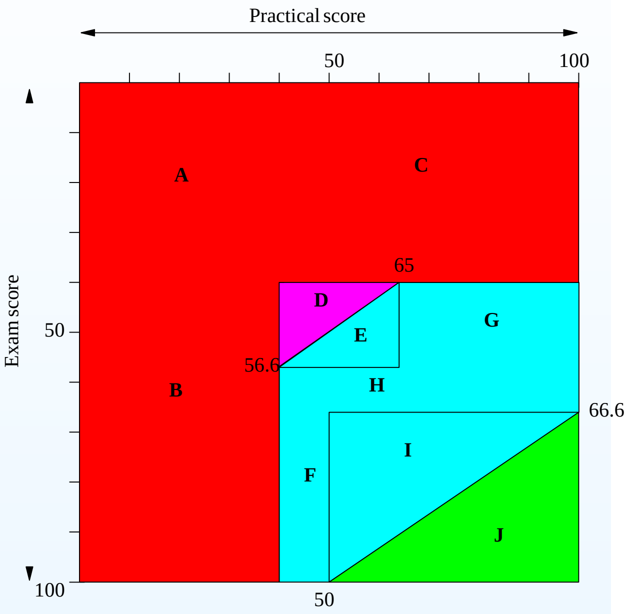
- 4 partitions ⇒ 4 test cases
- E.g. (exam,practical): (90,80), (50,80), (52,52), (30,80)
- And possibly: (-20,100)
- Is this enough? — Almost certainly not

- Pass with distinction
- Pass
- Fail
- Component fail



Grading 2 partitions, take 2

- Partitions:
  - A: $0 \leq \mathrm{e} < 40$ and $0 \leq \mathrm{p} < 40$
  - B: $40 \leq \mathrm{e} \leq 100$ and $0 \leq \mathrm{p} < 40$
  - C: $0 \leq \mathrm{e} < 40$ and $40 \leq \mathrm{p} \leq 100$
  - D: $40 \leq \mathrm{e} < 57$ and $40 \leq \mathrm{p} < 65$ and $\mathrm{combined} < 50$
  - E: $40 \leq \mathrm{e} < 57$ and $40 \leq \mathrm{p} < 65$ and $\mathrm{combined} \geq 50$
  - etc



- Coverage criterion determined by partitioning
- $\Rightarrow$ different partitioning gives different test requirements which gives potentially a different test suite
- Determining the partitioning can be hard!

Boundary Value Analysis

- faults often occur "at the edges"
- *Boundary value analysis* — choose test cases from the boundaries of the equivalence classes
- Coverage criterion: inputs from *both sides* of every boundary
- E.g., if equivalence classes are: $[-2^{31}, 0]$, $[1..12]$, $[13, 2^{31}-1]$
  - test cases are: -10 (from first equivalence class), 9 (second), 397 (third), -2^{31} & 0 (either end), 1 & 12 (ditto), $13\&2^{31}-1$

Dealing with complex input spaces

- “boundary” and “side” (in “both sides”) can be difficult to identify
- When the class is an area, there can be many values “on the boundary”
- Make each boundary a separate equivalence class, apply BVA to them
- (repeat as necessary for multi-dimensional data)
- E.g. Grading 2
  - For equivalence class A: $0 \leq \mathrm{e} < 40$ and $0 \leq \mathrm{p} < 40$
  - For equivalence class $p = 0$, boundaries are $e = 0$ and $e = 39$, so choose test cases (0,0), (0,27) (covering equivalence class), (0,39).
  - Other equivalence classes $e = 0$ boundaries $p = 1$ and $p = 39$ because (0,0) is already in an equivalence class

Triangle Program Equivalence Classes

- Which should we choose?
  - “any value 0”

or

  - “$x = 0$” (others are “don’t care”)
  - “$y = 0$” (others are “don’t care”)
  - “$z = 0$” (others are “don’t care”)

Triangle Program

- Test case 1: $(0, 1, 2)$
- Test case 2: $(2^{31} - 1, 1, 2)$
- Test case 3: $(4, 4, 8)$

- PAQ
- Agenda
- Previously
- Input Partitioning
- Equivalence Partitioning
- Examples
- Boundary Value Analysis
- EQ & BVA
- IDM
- Assessment
- Key Points

Grading 2: Example Solution

```
  public static List grade(int exam, int practical) {
A.  String grade = "Fail";
B.  if (exam < 40 && practical < 40) {
C.      grade = "Component Fail";
    } else {
D       int combined = (60*exam+40*practical)/100;
E       if (combined < 50) {
F           grade = "Fail";
G       } else if (combined >= 50 && combined < 80)  {
H           grade = "Pass";
I       } else if (combined >= 80 && combined < 100) {
J           grade = "Pass with distinction";
        }
    }
K   return grade;
  }
```

- faults that may not have been detected by test suites developed using coverage criteria from CFGs
- but are very likely to be detected with test suites developed using coverage criteria from BVA
- faults due to missing requirements more likely to be caught using equivalence partitioning
    - e.g. having a partition $\mathrm{exam} < 0 || \mathrm{practical} < 0$ catches problems dealing with invalid scores

Input Domain Modelling

- equivalence classes can also be determined by more general means than just groups of input values
- An input domain model (IDM) represents the inputs to an IUT in an abstract way
- the inputs are described in terms of characteristics
- characteristics can then be used to partition inputs
- characteristics can be developed directly from parameters to the IUT — interface-based — or from a functional view of the IUT — functionality-based
- adding more (or better) characteristics and partitioning potentially improves test suite

Examples

```
public boolean findElement(List list, Object element);
```

(from *Introduction to Software Testing*, Ammann and Offutt)

interface characteristics

- list is null — **true** (partition n_1) or **false** (partition n_2)
- list is empty — **true** (partition e_1) or **false** (partition e_2)

Test cases: (n_1, e_1),(n_1, e_2),(n_2, e_1),(n_2, e_2)

functionality characteristics

- number of elements in list — **0** (ne_1), **1** (ne_2), $> 1\,(ne_3)$
- element is first in list — **true** (f_1), **false** (f_2)

Test cases: every combination from the two characteristics

- PAQ
- Agenda
- Previously
- Input Partitioning
- Equivalence Partitioning
- Examples
- Boundary Value Analysis
- EQ & BVA
- IDM
- Assessment
- Key Points

Assessment

- can be applied to different levels: e.g. unit, module, system
- can be applied to different artifacts: e.g. code, requirements, design
  - don't need implementation to being development of tests
  - don't need as much technical knowledge (e.g. programming)
- the better the model of the inputs, the better the resulting test suite is likely to be
- *there is no reason why the model cannot include information from an implementation*
- many (more sophisticated) variations on the basic idea
- As with paths (including du-paths), considering every possibility (every possible combination) can lead to impractical number of test requirements, so need coverage criteria that produce subsets

Key Points

- Another model of a implementation under test (IUT) is to describe it in terms of behaviour with respect to inputs
- Equivalence Partitioning — divide the input space into regions that we expect the IUT to “behave the same”. Test suite consists of inputs that are representatives of each region
- Boundary Value Analysis — same model as equivalence partitioning, test suite consists of inputs that are “on the boundary”
- Input Domain Model — abstract representation of inputs