CITS5501 Software Testing and Quality Assurance
Input Space Partition Testing

Unit coordinator: Arran Stewart

March 13, 2018

Highlights

- This lecture looks at how we choose values for tests

Problem – how to choose test values

- An example method we might want to test:

```
public boolean findElement (List list, Object elem)
// Effects:
//  if list or elem is null throw NullPointerException
//  else return true if elem is in the list, false otherwise
```

- What are the possible values for `list`?
  For `elem`?

Input Domains

- The input domain for a program contains all the possible inputs to that program
- For even small programs, the input domain is so large that it might as well be infinite
- Testing is fundamentally about choosing finite sets of values from the input domain
- *Input parameters* define the scope of the input domain
  - Parameters to a method
  - Data read from a file
  - Global variables
  - User level inputs
- Domain for each input parameter is partitioned into regions
- At least one value is chosen from each region

Benefits of ISP

- Can be equally applied at several levels of testing
  - Unit
  - Integration
  - System
- Easy to adjust the procedure to get more or fewer tests

Partitioning domains

- Informally:

  partitions are a collection of disjoint sets of some domain D which *cover* the domain.

- They are pairwise disjoint (i.e. none overlap each other)

Using Partitions – Assumptions

- Choose a value from each block
- Each value is assumed to be equally useful for testing
- Application to testing
  - Find characteristics in the inputs : parameters, semantic descriptions, ...
  - Partition each characteristic
  - Choose tests by combining values from characteristics
- Example Characteristics
  - Input X is null
  - Order of the input file F (sorted, inverse sorted, arbitrary, ...)
  - Min separation of two aircraft
  - Input device (DVD, CD, VCR, computer, ...)

Choosing partitions

- Choosing (or defining) partitions seems easy, but is easy to get wrong
- Consider the "order of file F":

  b1 = sorted in ascending order
  b2 = sorted in descending order
  b3 = arbitrary order

Choosing partitions (2)

- But is this really a partitioning?

  What if the file is of length 1?
  The file will be in all three blocks . . .
  That is, disjointness is not satisfied

Choosing partitions (3)

Solution:
Each characteristic should address just one property

- File F sorted ascending
  - b1 = true
  - b2 = false
- File F sorted descending
  - b1 = true
  - b2 = false

Properties of Partitions

- If the partitions are not complete or disjoint, that means the partitions have not been considered carefully enough
- They should be reviewed carefully, like any design attempt
- Different alternatives should be considered
- We model the input domain in five steps . . .

Modeling the Input Domain – step 1

- Step 1: Identify testable *functions*
  - i.e., functions in the sense of mappings from a domain to results.
  - Can apply to methods, classes, programs
  - Individual methods have one testable function
  - In a class, each method often has the same characteristics
  - Programs have more complicated characteristics – modeling documents such as UML use cases can be used to design characteristics
  - Systems of integrated hardware and software components can use devices, operating systems, hardware platforms, browsers, etc

Modeling the Input Domain – step 2

- Step 2: Find all the parameters
  - Often fairly straightforward, even mechanical
  - Important to be complete
- Applied to different levels:
  - Methods: Parameters and state (non-local) variables used
  - Components: Parameters to methods and state variables
  - System: All inputs, including files and databases

Modeling the Input Domain – step 3

- Step 3: Model the input domain
  - We need to characterise the input domain, and partition into sets of blocks –
    where each block represents a set of values

Modeling the Input Domain – step 4

- Step 4: Apply a test criterion to choose combinations of values
  - Each test input has possible values, which we've partitioned
  - But even considering all the combinations of partitions, we end up with a very large number
  - *Coverage criteria* are criteria for choosing *subsets* of combinations (more later)

Modeling the Input Domain – step 5

- Step 5 : Refine combinations of blocks into test inputs
  - Choose appropriate values from each block

Two Approaches to Input Domain Modeling

1. Interface-based approach
   - Develops characteristics directly from individual input parameters
   - Simplest application
   - Can be partially automated in some situations
2. Functionality-based approach
   - Develops characteristics from a behavioral view of the program
   - under test
   - Harder to develop – requires more design effort
   - May result in better tests, or fewer tests that are as effective

Interface-Based Approach

- Mechanically consider each parameter in isolation
- This is an easy modeling technique and relies mostly on syntax
- Some domain and semantic information won't be used
  - Could lead to an incomplete IDM
- Ignores relationships among parameters

Functionality-Based Approach

- Identify characteristics that correspond to the intended functionality
- Requires more design effort from tester
- Can incorporate domain and semantic knowledge
- Can use relationships among parameters
- Modeling can be based on requirements, not implementation
- The same parameter may appear in multiple characteristics, so it's harder to translate values to test cases

Steps 1 and 2 – Identifying Functionalities, Parameters and Characteristics

- A creative engineering step
- More characteristics means more tests
- Interface-based : Translate parameters to characteristics
- Candidates for characteristics :
  - Preconditions and postconditions
  - Relationships among variables
  - Relationship of variables with special values (zero, null, blank, ...)
- Should not use program source – characteristics should be based on the input domain
  - Program source should be used with graph or logic criteria
- Better to have more characteristics with few blocks
  - Fewer mistakes and fewer tests

Steps 1 and 2 – Interface vs Functionality-Based

```
public boolean findElement (List list, Object elem)
// Effects:
//  if list or elem is null throw NullPointerException
//  else return true if elem is in the list, false otherwise
```

Interface-Based Approach:

- Two parameters : list, element
- Characteristics:
  list is null (block1 = true, block2 = false)
  list is empty (block1 = true, block2 = false)

Functionality-Based Approach:

- Two parameters : list, element
- Characteristics:
  number of occurrences of element in list
  (0, 1, >1)
  element occurs first in list
  (true, false)
  element occurs last in list
  (true, false)

Step 3 : Modeling the Input Domain

- Partitioning characteristics into blocks and values is a creative engineering step
- More blocks means more tests
- The partitioning often flows directly from the definition of characteristics and both steps are sometimes done together
  - Should evaluate them separately – sometimes fewer characteristics can be used with more blocks and vice versa
- Strategies for identifying values :
  - Include valid, invalid and special values
  - Sub-partition some blocks
  - Explore boundaries of domains
  - Include values that represent "normal use"
  - Try to balance the number of blocks in each characteristic
  - Check for completeness and disjointness

Using More than One IDM

- Some programs may have dozens or even hundreds of parameters
- Create several small IDMs
  - A divide-and-conquer approach
- Different parts of the software can be tested with different amounts of rigor
  - For example, some IDMs may include a lot of invalid values
- It is okay if the different IDMs overlap
  - The same variable may appear in more than one IDM

Step 4 – Choosing Combinations of Values

- Once characteristics and partitions are defined, the next step is to choose test values
- We use criteria – to choose effective subsets
- The most obvious criterion is to choose all combinations . . .

  *All Combinations (ACoC) : All combinations of blocks from all characteristics must be used.
- Number of tests is the product of the number of blocks in each