Test Generation from Requirements

Lecture 5

Dr. Ilchul Yoon (icyoon@suny.korea.ac.kr)

Sources: (1) Foundations of Software Testing (textbook), (2) slides by Prof. Mathur

Boundary-Value Analysis

- Test-selection technique that targets faults in applications at the boundaries of equivalence classes

- Generally done in conjunction with eq. partitioning

- Method
  - Partition the input domain using eq. partitioning.
  - Identify the boundaries for each partition. Boundaries may also be identified using special relationships amongst the inputs.
  - Select test data such that each boundary value occurs in at least one test input.

Consider a function findPrice. Assuming that an item code must be in the range 99.999 and quantity in the range 1..100,

Equivalence classes for code:
- E1: Values less than 99.
- E2: Values in the range.
- E3: Values greater than 999.

Equivalence classes for qty:
- E4: Values less than 1.
- E5: Values in the range.
- E6: Values greater than 100.

Equivalence classes and boundaries for findPrice.
Boundaries are indicated with an x. Points near the boundary are marked with *.
Boundary-Value Analysis

- An example test set
  
  ```
  T = { t1: (code = 98, qty = 0),
        t2: (code = 99, qty = 1),
        t3: (code = 100, qty = 2),
        t4: (code = 999, qty = 99),
        t5: (code = 999, qty = 100),
        t6: (code = 1000, qty = 101) }
  ```
  
  Illegal values of code and qty are included.

  - Minimal, but not the best!
    - Relationships amongst the variables must be examined carefully while identifying boundaries along the input domain.
    - Additional tests may be obtained by taking the product of equivalence classes

  Additional tests may be obtained by taking the product of equivalence classes

- An example
  ```java
  public void process(int code, int qty) {
    if ((code == 99 && code == 999) == true) {
      display_error("Invalid code!"); return;
    }
    // Validity check for qty is missing!
    if (begin_processing(code, qty)) {
      t1: (code = 98, qty = 0),
      t2: (code = 99, qty = 1),
      t3: (code = 100, qty = 2),
      t4: (code = 999, qty = 99),
      t5: (code = 999, qty = 100),
      t6: (code = 1000, qty = 101)
    }
  }
  ```
  
  t1 and t6 cannot detect qty related problem.

  Replace with additional test cases
  ```
  t7 = (code = 98, qty = 45)
  t8 = (code = 1000, qty = 45)
  t9 = (code = 250, qty = 0)
  t10 = (code = 250, qty = 101)
  ```

Another example in the textbook

- Equivalence classes
  - Input space
    Equivalence classes for x: E1: empty string, E2: nonempty string.
    Equivalence classes for x: E3: empty string, E4: nonempty string.
  - For output
    Eq. classes for x: E5: p < 0, E6: p ≥ 0

Boundary-Value Analysis (BVA)

- Selected test inputs
  ```
  T = { t1: (code = 98, qty = 0)
        t2: (code = 99, qty = 1)
        t3: (code = 100, qty = 2)
        t4: (code = 999, qty = 99)
        t5: (code = 999, qty = 100)
        t6: (code = 1000, qty = 101)
  ```

Imposed by BVA

- Another suggested input
  ```
  t7: (code = 98, qty = 45)
  t8: (code = 1000, qty = 45)
  t9: (code = 250, qty = 0)
  t10: (code = 250, qty = 101)
  ```
Category-Partition Method (Ostrand, 1988)

- Similar to equivalence partitioning
  - Formal, systemized version of eq. partitioning and BVA
  - A mix of manual and automated steps

Steps

1. Analyze specification
2. Identify categories
3. Partition categories
4. Identify constraints

Tasks in solid boxes are difficult to automate

- (39) write test
- (30) Test specification
- (27) Test frames

Category-Partition Method

- Analyze specification
  - Identify each functional unit independently testable
  - Identify in a hierarchical way

- Identify categories
  - Analyze inputs and environment objects
  - Determine the characteristics of each input and env. obj.

  code: length, leftmost digit, remaining digits
  qty: integer quantity
  weight: float quantity
  database: contents

Example requirements

Function: TestProc

Syntax: (Price, quantity, weight)

Function:

TestProc takes three inputs: code, qty, and weight. The code is represented by a string of eight digits contained in variable code. The quantity purchased is contained in qty. The weight of the item purchased is contained in weight.

Variables: (P, requires a database to find and display the unit price, the description, and the total price of the item corresponding to code; P is required to display an error message and return, if either of the three values is in error. However, for this example, the specification of the environment of the code is not necessary. Example: how the values of qty and weight are to be used. code is an eight-digit string (this format used for the price). P is concerned with only the leftmost digit that is interpreted as follows:

- Leftmost digit
  - Interpretation
  - 0: Ordinary grocery items such as bread, margarine, and soup
  - 2: Vegetable-weight items such as meat, fruits, and vegetables
  - 3: Health-related items such as insulin, contraceptives, and tampax
  - 5: Cooper digit 2 (defined), 3 and 4 (meats)
  - Specify the document
  - 1, 4-9: Unused

Partition categories

- Multiple choices are expected for each category

- Input parameter examples
  - Code
    - length (valid/invalid)
    - Leftmost digit (0, 2, 3, 5, others)
    - Remaining digits (valid/invalid string)
  - Weight: float quantity
    - Valid/invalid (e.g., 0)

- Environments example
  - Database
    - Contents (exist/not-exist)
Category-Partition Method

- Identify constraints
  - Feasible or infeasible combinations
  - Relationships between inputs
  - Specified with properties and selector expressions
  - Suffix to choices

- Example

  ```
  # Leftmost digit of code
  2
  
  # Remaining digits of code
  [property ordinary-grocery]
  [property variable-weight]
  
  # Valid value of qty
  valid
  
  # Invalid value of qty
  invalid
  
  [≤ ordinary-grocery]
  [error]
  
  selector expression
  ```

Category-Partition Method

- Re(w)rite test specification in TSL (Test Specification Language) with a precise syntax
  - Extend partitioned category
  - Completely specify constraints or properties for choices
  - Examples
    - Remaining digits:
      - Valid string [single]
      - Invalid string [error]
    - Weight:
      - Valid weight [variable-weight]
      - Invalid weight [error]

Category-Partition Method

- Process specification
  - Automatic processing of TSL specification, resulting in a number of test frames
  - Generate (all) possible combinations of choices while satisfying constraints
  - Sample test case
    - List of choices used for the parameters
    - Parameters
      - Test case 2: (Key = 1.2.1.0.1.1)
      - valid
      - 2
      - valid string
      - 3.19
      - item exists
    - 0 means no choice.

Category-Partition Method

- Evaluate generator output
  - Examine test frames manually by tester
  - Check redundancy or missing cases

- Generate test scripts
  - Test cases are combined into test scripts where each is a group of test cases
  - Group test cases that need same environment setting
cause - effect graphing

- Focus on modeling dependency relationships among program input conditions (causes) and output conditions (effects).
  - Known as dependency modeling
  - Represents logical relationships in a cause-effect graph, which can also be represented as a Boolean expression.

- Select various combinations of input values as tests
- Avoid combinatorial explosion using heuristics

cause - effect graphing

- Generic procedure
  - Identify causes and effects by reviewing requirements
  - Express the relationship between causes and effects using cause-effect graph
  - Transform the graph into a limited entry decision table
  - Generate tests from the decision table

cause - effect graphing

- Example
  - Dispense food if DF switch is ON.

notations in cause - effect graphing

- Basic elements
  - If \( C \) then \( Ef \)
  - If \( \neg C \) then \( Ef \)
  - If \( C_1 \&\& C_2 \&\& C_3 \) then \( Ef \)
  - If \( C_1 \mid\mid C_2 \) then \( Ef \)

- Constraints between causes
  - Exclusive (0 or more)
  - Inclusive
  - Requires
  - One and Only-One constraint
Notations in Cause-Effect Graphing

- Possible Values of Causes constrained by E, I, R, O

<table>
<thead>
<tr>
<th>Constraint</th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C1, C2)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(C1, C2)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(C1, C2)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(C1, C2)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(C1, C3)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(C1, C3)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(C1, C3)</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(C1, C3)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- Constraints between effects: Masking
  - Ef1: Generate shipping invoice
  - Ef2: Generate an "Order not shipped" regret letter.

Creating Cause-Effect Graph

- Causes identified:
  - C1: Purchase CPU 1.
  - C2: Purchase CPU 2.
  - C3: Purchase CPU 3.
  - C4: Purchase PR 1.
  - C5: Purchase PR 2.
  - C7: Purchase M 23.
  - C8: Purchase M 30.

- Effects ("free") identified:
  - Ef1: RAM 256.
  - Ef2: RAM 512 and PR 1.
  - Ef3: RAM 1G and PR 2.
  - Ef4: No giveaway with this item.

Intermediate nodes are useful to represent complex conditions -- e.g., (C1 ∨ C2) ∨ C3

Creating Cause-Effect Graph

- Make a list of causes and effects by carefully reviewing requirements

Example:
- Requirements
- Sample purchase configurations and free window contents

<table>
<thead>
<tr>
<th>Items purchased</th>
<th>&quot;Free&quot; window</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 1</td>
<td>RAM 256</td>
<td>$499</td>
</tr>
<tr>
<td>CPU 1, PR 1</td>
<td>RAM 256</td>
<td>$628</td>
</tr>
<tr>
<td>CPU 2, PR 2, M 23</td>
<td>PR 1, RAM 512</td>
<td>$1255</td>
</tr>
<tr>
<td>CPU 3, M 30</td>
<td>PR 2, RAM 1G</td>
<td>$1540</td>
</tr>
</tbody>
</table>

Intermediate node

Decision Table from Cause-Effect Graph

- Decision Table
- N x M matrix where
  - N is the sum of the number of conditions and effects
  - M is the number of tests

Procedure
- Initialize an empty decision table
- For each effect,
  - Find combinations of conditions that causes the effect
  - Trace cause-effect graph backward
  - Try avoiding combinatorial explosion
  - Update the table by adding the combinations as new columns
  - Update the next available column in the table
Example: Cause-Effect Graphing

- Cause-effect graph

C_1, C_2, C_3 combinations that cause Ef_1 to be present
  - Ef_1 must be present when node 2 is in 1-state
  - Need also to consider C_3 requires C_1

C_1 \ C_2 \ C_3

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Append to the decision table

Example: Cause-Effect Graphing

- Repeat for Ef_2
  - C_4 has to be 1
  - Need also to consider exclusive between C_2 and C_4

V_1 1 0 1 1 1 1 1 1
V_2 1 0 0 1 1 1 1 1
V_3 0 0 0 1 1 1 1 1

Append to the decision table

Heuristics to Avoid Combinatorial Explosion

- Combinatorial explosion can happen, while tracking back a cause-effect graph.
  - Because we generate combinations of causes that set an intermediate node to 0-state or 1-state.
  - For n causes related to an effect e, in worst, 2^n combinations to bring e

Simple heuristics

Example: Cause-Effect Graphing

- Full vectors of causes and effects for Ef_1
  - (setting C_4 to 0)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>C_1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C_2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C_3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Append to the decision table

Heuristics to Avoid Combinatorial Explosion

- Because we generate combinations of causes that set an intermediate node to 0-state or 1-state.
  - For n causes related to an effect e, in worst, 2^n combinations to bring e

Simple heuristics