Test case design techniques II: Blackbox testing
Overview

- Black-box testing (or functional testing):
  - Equivalence partitioning
  - Boundary value analysis
  - Cause-effect graphing
  - Behavioural testing
  - Random testing
  - Error guessing etc…

- How to use black-box and white-box testing in combination
- Basics: heuristics and experience
Black box testing

SUT

requirements
input
events
output

domain testing

x
y
Black-box: Three major approaches

• Analysis of the input/output domain of the program:
  • Leads to a logical partitioning of the input/output domain into ‘interesting’ subsets

• Analysis of the observable black-box behaviour:
  • Leads to a flow-graph-like model, which enables application of techniques from the white-box world (on the black-box model)

• Heuristics
  • Techniques like risk analysis, random input, stress testing
Types of Testing

characteristics
- security
- reliability
- robustness
- performance
- user-friendliness
- functional behaviour

method
- unit
- module
- integration
- system

level of detail
white-box  black-box
V - Model

requirements → specification → architecture spec → detailed design → implementation code → unit-test

acceptance test spec → system test spec → integration test spec → module test spec → unit test spec

acceptance test → system test → integration test → module test → unit-test
Black-box : Equivalence Partitioning

• Divide all possible inputs into classes (partitions) such that
  • There is a finite number of input equivalence classes
  • You may reasonably assume that
    • the program behaves analogously for inputs in the same class
    • a test with a representative value from a class is sufficient
    • if representative detects fault then other class members
      will detect the same fault
Black-box : Equivalence Partitioning

Strategy :

• Identify input equivalence classes
  • Based on conditions on inputs / outputs in specification / description
  • Both valid and invalid input equivalence classes
  • Based on heuristics and experience
    • “input x in [1..10]” → classes : $x < 1$, $1 \leq x \leq 10$, $x > 10$
    • “enumeration A, B, C” → classes : A, B, C, not{A,B,C,}
    • ........

• Define one / couple of test cases for each class
  • Test cases that cover valid eq. classes
  • Test cases that cover at most one invalid eq. class
**Example: Equivalence Partitioning**

- Test a function for calculation of absolute value of an integer

- **Equivalence classes:**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Valid eq. classes</th>
<th>Invalid eq. Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>nr of inputs</td>
<td>1</td>
<td>0, &gt; 1</td>
</tr>
<tr>
<td>Input type</td>
<td>integer</td>
<td>non-integer</td>
</tr>
<tr>
<td>particular <em>abs</em></td>
<td>&lt; 0, &gt;= 0</td>
<td></td>
</tr>
</tbody>
</table>

- **Test cases:**

  - $x = -10, \quad x = 100$
  - $x = \text{“XYZ”}, \quad x = -10, 20$
A Self-Assessment Test [Myers]

“A program reads three integer values. The three values are interpreted as representing the lengths of the sides of a triangle. The program prints a message that states whether the triangle is scalene (uligesidet), isosceles (ligebenet), or equilateral (ligesidet).”

• Write a set of test cases to test this program.
A Self-Assessment Test  [Myers]

Test cases for:

1. valid scalene triangle ?
2. valid equilateral triangle ?
3. valid isosceles triangle ?
4. 3 permutations of previous ?
5. side = 0 ?
6. negative side ?
7. one side is sum of others ?
8. 3 permutations of previous ?
9. one side larger than sum of others ?
10. 3 permutations of previous ?
11. all sides = 0 ?
12. non-integer input ?
13. wrong number of values ?
14. for each test case: is expected output specified ?
15. check behaviour after output was produced ?
Example: Equivalence Partitioning

• Test a program that computes the sum of the first value integers as long as this sum is less than maxint. Otherwise an error should be reported. If value is negative, then it takes the absolute value.

• Formally:

Given integer inputs maxint and value compute result:

\[ \text{result} = \sum_{K=0}^{\left|\text{value}\right|} k \quad \text{if this} \leq \text{maxint}, \quad \text{error otherwise} \]
Example: Equivalence Partitioning

- Equivalence classes:
  - Condition
  - Nr of inputs: 2 < 2, > 2
  - Type of input: int int int no-int, no-int int
  - Abs(value): value < 0, value ≥ 0
  - maxint: \( \sum k \leq \text{maxint}, \sum k > \text{maxint} \)

- Test Cases:
  - Valid
  - maxint value result
  - 100 10 55
  - 100 -10 55
  - 10 10 error
  - Invalid
  - 10 - error
  - 10 20 30 error
  - "XYZ" 10 error
  - 100 9.1E4 error
Black-box: Boundary Value Analysis

• Based on experience / heuristics:
  • Testing *boundary conditions* of eq. classes is more effective
    i.e. values directly on, above, and beneath edges of eq. classes
  • Choose input boundary values as tests in input eq. classes
    instead of, or additional to arbitrary values
  • Choose also inputs that invoke *output boundary values*
    (values on the boundary of output classes)
  • Example strategy as extension of equivalence partitioning:
    • choose one (*n*) arbitrary value in each eq. class
    • choose values exactly on lower and upper boundaries of eq. class
    • choose values immediately below and above each boundary
      (if applicable)
Example: Boundary Value Analysis

- Test a function for calculation of absolute value of an integer
- Valid equivalence classes:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Valid eq. classes</th>
<th>Invalid eq. Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>particular abs</td>
<td>&lt; 0, &gt;= 0</td>
<td></td>
</tr>
</tbody>
</table>

- Test cases:
  - class $x < 0$, arbitrary value: $x = -10$
  - class $x >= 0$, arbitrary value $x = 100$
  - classes $x < 0$, $x >= 0$, on boundary: $x = 0$
  - classes $x < 0$, $x >= 0$, below and above: $x = -1$, $x = 1$
A Self-Assessment Test  [Myers]

Test cases for:

1. valid scalene triangle ?
2. valid equilateral triangle ?
3. valid isosceles triangle ?
4. 3 permutations of previous ?
5. side = 0 ?
6. negative side ?
7. one side is sum of others ?
8. 3 permutations of previous ?

9. one side larger than sum of others ?
10. 3 permutations of previous ?
11. all sides = 0 ?
12. non-integer input ?
13. wrong number of values ?
14. for each test case: is expected output specified ?
15. check behaviour after output was produced ?
Example: Boundary Value Analysis

• Given integer inputs \( \text{maxint} \) and \( \text{value} \) compute \( \text{result} \):

\[
\text{result} = \sum_{K=0}^{\text{|value|}} k \quad \text{if this} \leq \text{maxint}, \quad \text{error otherwise}
\]

• Valid equivalence classes:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Valid eq. Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Abs(value)} )</td>
<td>( \text{value} &lt; 0, \quad \text{value} \geq 0 )</td>
</tr>
<tr>
<td>( \text{maxint} )</td>
<td>( \sum k \leq \text{maxint}, \quad \sum k &gt; \text{maxint} )</td>
</tr>
</tbody>
</table>

• Should we also distinguish between \( \text{maxint} < 0 \) and \( \text{maxint} \geq 0 \) ?

| \( \text{maxint} \)       | \( \text{maxint} < 0, \quad 0 \leq \text{maxint} < \sum k, \quad \text{maxint} \geq \sum k \) |
Example: Boundary Value Analysis

• Valid equivalence classes:

  - $\text{Abs}(value)$: $value < 0$, $value \geq 0$
  - $\text{maxint}$: $\text{maxint} < 0$, $0 \leq \text{maxint} < \sum k$, $\text{maxint} \geq \sum k$

• Test Cases:

<table>
<thead>
<tr>
<th>maxint</th>
<th>value</th>
<th>result</th>
<th>maxint</th>
<th>value</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>55</td>
<td>10</td>
<td>55</td>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>54</td>
<td>10</td>
<td>error</td>
<td>100</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>56</td>
<td>10</td>
<td>55</td>
<td>100</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

• How to combine the boundary conditions of different inputs? Take all possible boundary combinations? This may blow-up.
Black-box : Cause Effect Graphing

- Black-box testing technique to analyse combinations of input conditions
- Identify *causes* and *effects* in specification
  \[ \downarrow \quad \downarrow \]
  inputs     outputs
  current state  new state
- Make Boolean Graph linking causes and effects
- Annotate impossible combinations of causes and effects
- Develop decision table from graph with in each column a particular combination of inputs and outputs
- Transform each column into test case
Black-Box: Cause Effect Graphing

\[ \sum k \leq \text{maxint} \]
\[ \sum k > \text{maxint} \]
\[ \text{value} < 0 \]
\[ \text{value} \geq 0 \]

**Causes**
- \( \sum k \leq \text{maxint} \):
  - \( \sum k \)
  - \( \text{and} \)
  - \( \text{xor} \)
  - \( \text{and} \)
  - \( \text{error} \)
- \( \sum k > \text{maxint} \):
  - \( \sum k \)

**Inputs**
- \( \sum k \leq \text{maxint} \):
  - \( 1 \)
  - \( 1 \)
  - \( 0 \)
  - \( 0 \)
- \( \sum k > \text{maxint} \):
  - \( 0 \)
  - \( 0 \)
  - \( 1 \)
  - \( 1 \)
- \( \text{value} < 0 \):
  - \( 1 \)
  - \( 0 \)
  - \( 1 \)
  - \( 0 \)
- \( \text{value} \geq 0 \):
  - \( 0 \)
  - \( 1 \)
  - \( 0 \)
  - \( 1 \)

**Effects**
- \( \sum k \):
  - \( 1 \)
  - \( 1 \)
  - \( 0 \)
  - \( 0 \)
- \( \text{error} \):
  - \( 0 \)
  - \( 0 \)
  - \( 1 \)
  - \( 1 \)
Black-box: Cause Effect Graphing

- Systematic method for generating test cases representing combinations of conditions
- Combinatorial explosion of number of possible combinations
- Some heuristics to reduce this combinatorial explosion
- Starting point is effects (outputs) then working ‘backwards’
- ‘light-weight’ formal methods: transformation into semi-formal Boolean graph
- A technique: to be combined with others
Black-box: behavioural specifications

• Many systems are partly specified through the interaction with an environment, e.g.:
  • Phone switches (dialing sequences)
  • Typical PC applications (GUI dialogues)
  • Consumer electronics (mobile phones)
  • Control systems (cruise, navigation)

• Typical specification formalisms:
  • Use cases
  • Sequence diagrams
  • State machines

• In many situations, abstract test cases can be derived directly from such specifications

{ Will be elaborated later in this course }
Example: Use case

One test per use case:
1. Subscribe
2. Place call
3. Answer call
4. Unsubscribe
Example: sequence diagrams

Test:
1. Key-digit
2. Key-digit
3. Key-digit
4. Key-digit
5. key-on

Once a key has been pressed, the on key has up to five seconds to be pressed.

Once a code has been entered, the code must be pressed within three seconds.

During the exit time delay, the Red LED will flash on and off.
During this time, detection of doors opening and movement in rooms will be ignored.

LED will flash at rate of 2Hz.
Example: state machine

Tests:
1. evArm
2. evDoor
3. evDisarm
Black-box: syntax testing

• Many kinds of program inputs are syntax driven, e.g.:
  • Command line input
  • Web forms
  • Language definitions
• Normally, such inputs are analysed by standard parsers, however:
  • Boundary conditions may still be useful to apply in order to check correct error handling
• The techniques for behavioural testing can be used
Syntax testing example

• **Commands**: put | get

Some tests:

1. p,u,t
2. g,e,t
3. q,u,t
4. p,u
5. p,u,s
6. ........
Black-box: random/stochastic

- Basic idea: Drive the system through typical scenarios, extreme scenarios, and rare scenarios in a random way.
- Motivation: Increase the chance of ‘hitting’ system faults.
- Application areas:
  - Systems that run forever in some nondeterministic way, e.g. control systems and communication systems
  - Systems with huge input domains
- Examples:
  - Random mouse clicking/typing towards a GUI.
  - Typical browser-user behaviour: (click;read;)\* with a typical random distribution of waiting time
  - Random walk through a specification state model while testing
Black-box: stress testing

- Basic idea: Let the environment behave in an extreme way towards the system in order to identify faults.
- Examples:
  - Emulate an extreme number of web users of a given application
  - Denial of service attacks
  - Push ‘on/off’ on the cars cruise control a number of times followed by a turn-off of the motor and a ‘on’ push.
  - Send a huge amount of buffers on a network connection as fast as possible
  - Power off the washing machine in any state
Black-box: Error Guessing

• Just ‘guess’ where the errors are ……
• Intuition and experience of tester
• Ad hoc, not really a technique
• Strategy:
  • Make a list of possible errors or error-prone situations
    (often related to boundary conditions)
  • Write test cases based on this list
Black-box: Error Guessing

- More sophisticated ‘error guessing’: *Risk Analysis*
- Try to identify critical parts of program (high risk code sections):
  - parts with unclear specifications
  - developed by junior programmer while his wife was pregnant ……
  - complex code:
    - measure code complexity - tools available (McGabe, Logiscope, …)
- High-risk code will be more thoroughly tested
  ( or be rewritten immediately …..)
Black-Box Testing: Which One?

- Black-box testing techniques:
  - Equivalence partitioning
  - Boundary value analysis
  - Cause-effect graphing
  - Error guessing
  - Test derivation from formal specifications
  - ........

- Which one to use?
  - None is complete
  - All are based on some kind of heuristics
  - They are complementary
Black-Box Testing: Which One?

- Always use a combination of techniques
  - When a formal specification is available try to use it
  - Identify valid and invalid input equivalence classes
  - Identify output equivalence classes
  - Apply boundary value analysis on valid equivalence classes
  - Guess about possible errors
  - Cause-effect graphing for linking inputs and outputs
White-Box testing : How to Apply ?

• Don’t start with designing white-box test cases !
• Start with black-box test cases
  (equivalence partitioning, boundary value analysis,
  cause effect graphing, test derivation with formal methods, …..)
• Check white-box coverage
  ( statement-, branch-, condition-, ….. coverage )
• Use a coverage tool – maybe combined with a Unit framework
• Design additional white-box test cases for not covered code
A Coverage Tool: gcov

- Standard Gnu tool gcov
- Only statement coverage
- Compile your program under test with a special option
- Run a number of test cases
- A listing indicates how often each statement was executed and percentage of statements executed