Boundary Value Testing

Mohammad Mousavi

Eindhoven University of Technology, The Netherlands

Software Testing, 2012
Announcements

- Groups are announce on the web page.
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Robustness</th>
<th>Worst-Cases</th>
<th>Worst-Cases + Robustness</th>
<th>Special Values</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Introduction</td>
<td>Robustness</td>
<td>Worst-Cases</td>
<td>Worst-Cases + Robustness</td>
<td>Special Values</td>
</tr>
</tbody>
</table>

Mousavi: Boundary Value Testing
Assumptions

- functional testing: program is an input from a certain domain to a certain range
- impossible to check all input/output combinations: need to choose some
- applicability: domain comprises (product of)
  1. independent values
  2. physical (not logical) quantities (ordered, in an interval, taking all values in the interval)
- rationale: most errors occur at extremes (< instead of ≤, counters off by one)
  also called: stress testing
- technique also applicable to range boundaries
Idea

- Choose 4 candidate values for each input in the range \([a, b]\):
  1. at the 2 extremes (\(a\) and \(b\)),
  2. near the 2 extremes (predecessor of \(a\) and successor of \(b\)).

- Choose **nominal values** for all other variables.

- **Single-fault** assumption: each failure is the result of a single bug (and a single error)

- Assuming n input variables, \(4n + 1\) test-cases.
Functional Testing: Mortgage Example

Spec. Write a program that takes three inputs: gender (boolean), age([18-55]), salary ([0-10000]) and output the total mortgage for one person.

Mortgage = salary * factor,
where factor is given by the following table.

<table>
<thead>
<tr>
<th>Category</th>
<th>Male</th>
<th>Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>Young</td>
<td>(18-35 years) 75</td>
<td>(18-30 years) 70</td>
</tr>
<tr>
<td>Middle</td>
<td>(36-45 years) 55</td>
<td>(31-40 years) 50</td>
</tr>
<tr>
<td>Old</td>
<td>(46-55 years) 30</td>
<td>(41-50 years) 35</td>
</tr>
</tbody>
</table>
An Implementation

Mortgage (male:Boolean, age:Integer, salary:Integer): Integer
if male then
    return ((18 ≤ age < 35)?(75 * salary) : (31 ≤ age < 40)?(55 * salary) : (30 * salary))
else {female}
    return ((18 ≤ age < 30)?(75 * salary) : (31 ≤ age < 40)?(50 * salary) : (35 * salary))
end if
Boundary Value Testing

**Mortgage** (male:Boolean, age:Integer, salary:Integer): Integer

if male then

return ((18 \leq age < 35)?(75 \times salary) : (31 \leq age < 40)?(55 \times salary) : (30 \times salary))

else {female}

return ((18 \leq age < 30)?(75 \times salary) : (31 \leq age < 40)?(50 \times salary) : (35 \times salary))

end if

- **age**: extremes: 18, 55(?). near extremes: 19, 54. nominal: 25.
- **salary**: extremes: 0, 10000. near extremes: 1, 9999. nominal: 5000.
- **male**: ??
Boundary Value Testing

\textbf{Mortgage} (\texttt{male:Boolean, age:Integer, salary:Integer}): Integer

\begin{verbatim}
if male then
    return \(((18 \leq \texttt{age} < 35)\?((75 \times \texttt{salary}) \? (31 \leq \texttt{age} < 40)\?((55 \times \texttt{salary}) \? (30 \times \texttt{salary}))))
else {female}
    return \(((18 \leq \texttt{age} < 30)\?((75 \times \texttt{salary}) \? (31 \leq \texttt{age} < 40)\?((50 \times \texttt{salary}) \? (35 \times \texttt{salary}))))
end if
\end{verbatim}

- \textbf{salary:} extremes: 0, 10000. near extremes: 1, 9999. nominal: 5000.
- \textbf{male:}

No boundaries: define artificial boundaries (e.g., 0 and MAXINT for integers).
Boundary Value Testing

**Mortgage** (male:Boolean, age:Integer, salary:Integer): Integer

if male then
  return ((18 ≤ age < 35)?(75 * salary) : (31 ≤ age < 40)?(55 * salary) : (30 * salary))
else {female}
  return ((18 ≤ age < 30)?(75 * salary) : (31 ≤ age < 40)?(50 * salary) : (35 * salary))
end if

- salary: extremes: 0, 10000. near extremes: 1, 9999. nominal: 5000.
- male: true, false. nominal: true. (just to try the technique)

No boundaries: define artificial boundaries (e.g., 0 and MAXINT for integers).
Boundary Value Testing

```plaintext
if \text{(male)} \text{ then return } \\
((18 \leq \text{age} < 35)?(75 \times \text{salary}) : (31 \leq \text{age} < 40)?(55 \times \text{salary}) : (30 \times \text{salary})) \text{ else return } \\
((18 \leq \text{age} < 30)?(75 \times \text{salary}) : (31 \leq \text{age} < 40)?(50 \times \text{salary}) : (35 \times \text{salary}))
```

<table>
<thead>
<tr>
<th>Gender</th>
<th>Age</th>
<th>Salary</th>
<th>Output</th>
<th>Correct Out.</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td>18</td>
<td>5000</td>
<td>75*5000</td>
<td>75*5000</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>19</td>
<td>5000</td>
<td>75*5000</td>
<td>75*5000</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>5000</td>
<td>75*5000</td>
<td>75*5000</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>54</td>
<td>5000</td>
<td>30*5000</td>
<td>30*5000</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>55</td>
<td>5000</td>
<td>30*5000</td>
<td>30*5000</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>0</td>
<td>75*0</td>
<td>75*0</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>1</td>
<td>75*1</td>
<td>75*1</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>9999</td>
<td>75*9999</td>
<td>75*9999</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>10000</td>
<td>75*9999</td>
<td>75*10000</td>
<td>P</td>
</tr>
<tr>
<td>female</td>
<td>25</td>
<td>5000</td>
<td>75*5000</td>
<td>70*5000</td>
<td>F</td>
</tr>
</tbody>
</table>

Mousavi: Boundary Value Testing
Boundary Value Testing

Observations

▶ strange technique for booleans: decision-table-based technique (yet to come)
▶ not suitable due to the dependency between age and gender
▶ more combinations to be tested: wait for a few slides!
▶ finer partitioning needed: wait till next session
▶ exploiting more information from the code
Outline

Introduction

Robustness

Worst-Cases

Worst-Cases + Robustness

Special Values

Random
Idea

- In addition to the 4 candidates, choose 2 more candidates just beyond the extremes.
- Predicting the output: tricky.
- Suitable for PL’s with weak typing (testing exception handling).
- Assuming n input variables, \( 6n + 1 \) test-cases.
Robustness BV Testing: Example

```python
if (male) then return

((18 ≤ age < 35)?(75 * salary) : (31 ≤ age < 40)?(55 * salary) : (30 * salary))

else return ((18 ≤ age < 30)?(75 * salary) : (31 ≤ age < 40)?(50 * salary) : (35 * salary))
```

<table>
<thead>
<tr>
<th>Gender</th>
<th>Age</th>
<th>Salary</th>
<th>Output</th>
<th>Correct Out.</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td>17</td>
<td>5000</td>
<td>30*5000</td>
<td>nog even niet</td>
<td>F</td>
</tr>
<tr>
<td>male</td>
<td>56</td>
<td>5000</td>
<td>75*5000</td>
<td>too late</td>
<td>F</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>-1</td>
<td>75*-1</td>
<td>invalid salary</td>
<td>F</td>
</tr>
<tr>
<td>male</td>
<td>25</td>
<td>10001</td>
<td>75*10001</td>
<td>75*10000(?)</td>
<td>F</td>
</tr>
</tbody>
</table>
Outline

Introduction

Robustness

Worst-Cases

Worst-Cases + Robustness

Special Values

Random
Idea

- multiple-fault assumption: a fault may be the result of a combination of errors
- all combinations of 5 values for all variables
- $5^n$ test-cases
Worst-Case BV Testing

\[
\text{if } (\text{male}) \text{ then return } \left(\left((18 \leq \text{age} < 35)?(75 \times \text{salary}) : (31 \leq \text{age} < 40)?(55 \times \text{salary}) : (30 \times \text{salary})\right) \right)
\]

\[
\text{else return } \left(\left((18 \leq \text{age} < 30)?(75 \times \text{salary}) : (31 \leq \text{age} < 40)?(50 \times \text{salary}) : (35 \times \text{salary})\right) \right)
\]

50 cases (5 \times 5 \times 2) in total;

discovered fault are quite similar to Robustness BV Testing (due to the same bugs)

(single-fault: reasonable in this case)
Outline

Introduction

Robustness

Worst-Cases

Worst-Cases + Robustness

Special Values

Random

Mousavi: Boundary Value Testing
Idea

- combination of worst case and robustness BV Testing
- all combinations of 7 values for all variables
- $7^n$ test-cases
Boundary Value Testing

\[
\text{if (male) then return}
((18 \leq age < 35)?(75 * salary) : (31 \leq age < 40)?(55 * salary) : (30 * salary))
\]

\[
\text{else return} \ ( (18 \leq age < 30)?(75 * salary) : (31 \leq age < 40)?(50 * salary) : (35 * salary))
\]

98 cases (7 * 7 * 2) in total;
A Brief Comparison

worst-case robustness

worst-case  robustness

boundary value
Outline

Introduction

Robustness

Worst-Cases

Worst-Cases + Robustness

Special Values

Random
Special Value Testing: Idea

- using domain knowledge
- finding corresponding boundaries for internal variables
- in combination with the techniques mentioned before
Special Value Testing: Example

```python
if (male) then return

((18 ≤ age < 35)?(75 * salary) : (31 ≤ age < 40)?(55 * salary) : (30 * salary))

else return ((18 ≤ age < 30)?(75 * salary) : (31 ≤ age < 40)?(50 * salary) : (35 * salary))
```

In addition to some of the worst-case robustness test-cases:

<table>
<thead>
<tr>
<th>Gender</th>
<th>Age</th>
<th>Salary</th>
<th>Output</th>
<th>Correct Out.</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td>18</td>
<td>1</td>
<td>75*1</td>
<td>75*1</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>35</td>
<td>1</td>
<td>55*1</td>
<td>75*1</td>
<td>F</td>
</tr>
<tr>
<td>male</td>
<td>36</td>
<td>1</td>
<td>55*1</td>
<td>55*1</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>45</td>
<td>1</td>
<td>30*1</td>
<td>55*1</td>
<td>F</td>
</tr>
<tr>
<td>male</td>
<td>46</td>
<td>1</td>
<td>30*1</td>
<td>30*1</td>
<td>P</td>
</tr>
<tr>
<td>male</td>
<td>55</td>
<td>1</td>
<td>30*1</td>
<td>30*1</td>
<td>P</td>
</tr>
</tbody>
</table>
### Test-cases (cont’d):

<table>
<thead>
<tr>
<th>Gender</th>
<th>Age</th>
<th>Salary</th>
<th>Output</th>
<th>Correct Out.</th>
<th>Pass/Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>18</td>
<td>1</td>
<td>75*1</td>
<td>70*1</td>
<td>F</td>
</tr>
<tr>
<td>female</td>
<td>30</td>
<td>1</td>
<td>35*1</td>
<td>70*1</td>
<td>F</td>
</tr>
<tr>
<td>female</td>
<td>31</td>
<td>1</td>
<td>50*1</td>
<td>50*1</td>
<td>P</td>
</tr>
<tr>
<td>female</td>
<td>40</td>
<td>1</td>
<td>35*1</td>
<td>50*1</td>
<td>F</td>
</tr>
<tr>
<td>female</td>
<td>41</td>
<td>1</td>
<td>35*1</td>
<td>35*1</td>
<td>P</td>
</tr>
<tr>
<td>female</td>
<td>50</td>
<td>1</td>
<td>35*1</td>
<td>35*1</td>
<td>P</td>
</tr>
</tbody>
</table>
Outline

Introduction

Robustness

Worst-Cases

Worst-Cases + Robustness

Special Values

Random

Mousavi: Boundary Value Testing
Random Testing

- randomly choosing input!
- range boundaries: determining the effectiveness of the test-cases
- statistical techniques
Conclusions

Boundary-Value Testing:

+ relatively straightforward test-case generation
- no sense of covering the input domain
- not enough white-box information exploited (apart from special-value testing)

Particularly suitable when the input data are:

1. independent
2. physical measures