Testing Techniques Applied to Virt Devel

Cleber Rosa
Red Hat, Inc.
Agenda

• Software Testing Basics
• Equivalence Partitioning
• Boundary Value Analysis
• Combinatorial Testing
Glenford J. Myers’ Triangle Check

- Input: 3 lengths of the triangle’s sides
- Output: the triangle classification
  - Equilateral
  - Isoceles
  - Scalene
- How hard can it be to write a comprehensive set of test cases?
## Triangle Check Basic Test Cases

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 1, 1</td>
<td>Equilateral</td>
</tr>
<tr>
<td>2, 2, 3</td>
<td>Isoceles</td>
</tr>
<tr>
<td>3, 4, 5</td>
<td>Scalene</td>
</tr>
</tbody>
</table>
def triangle_check(a, b, c):
    if a == b == c:
        return "equilateral"
    elif a != b != c:
        return "scalene"
    else:
        return "isoceles"
class Triangle(Test):
    def test_equilateral(self):
        self.assertEqual(triangle_check(1, 1, 1),
                         "equilateral")

    def test_isoceles(self):
        self.assertEqual(triangle_check(2, 2, 3),
                         "isoceles")

    def test_scalene(self):
        self.assertEqual(triangle_check(3, 4, 5),
                         "scalene")
## Triangle Check Error Test Cases

<table>
<thead>
<tr>
<th>Input</th>
<th>Expected Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 1</td>
<td>Error</td>
</tr>
<tr>
<td>-1, 1, 1</td>
<td>Error</td>
</tr>
<tr>
<td>1, 1, 2</td>
<td>Error (not isosceles)</td>
</tr>
<tr>
<td>1, 2, 3</td>
<td>Error (not scalene)</td>
</tr>
</tbody>
</table>
Triangle Check Extended Test Cases

- Permutations of lengths order
  - “(A + B) <= C” vs. “(C + B) <= A”
- Input is not a number
  - Give me a side with length “π”
- More or less than 3 input values
  - AKA “what do you mean by triangles must have three sides?”
Lessons from a simple example

• Even experienced developers will only think of a subset of those test cases
• Most software is not that simple
• Choosing good input data is key
  – Some input can be no better than other input already being used
  – Not all input are created equal, some will have a better shot at finding issues
  – We’ll explore some techniques next
• Don’t let the name scare you
• Think of groups of input that **should** generate similar **outcome**
  - A good pick is worth at least other two individual inputs
  - It usually tells us about what would happen (errors?) when values above or beyond itself would be used
// snippets from qemu/hw/acpi/cpu_hotplug.c

/* The current AML generator can cover the APIC ID range [0..255], */
/* inclusive, for VCPU hotplug. */
QEMU_BUILD_BUG_ON(ACPI_CPU_HOTPLUG_ID_LIMIT > 256);

... 

if (pcms->apic_id_limit > ACPI_CPU_HOTPLUG_ID_LIMIT) {
  error_report("max_cpus is too large. APIC ID of last CPU is \%u",
               pcms->apic_id_limit - 1);
  exit(1);
}
## Input Classes - # of CPUs

<table>
<thead>
<tr>
<th>Invalid (smaller than minimum required)</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>256</td>
</tr>
</tbody>
</table>
## Input Classes – CPU IDs

<table>
<thead>
<tr>
<th>Invalid (smaller than minimum required)</th>
<th>Valid</th>
<th>Invalid</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td>256</td>
<td></td>
</tr>
</tbody>
</table>
Boundary Analysis

- Also not scary
- When input classes are ordered, you can easily spot them
- These values are usually very good bets for tests
// snippets from tp-qemu/qemu/tests/cfg/cpu_add.cfg
smp = 4
vcpu_maxcpus = 255

Variants:
- cpuid_outof_range:
  cpuid_hotplug_vcpu0 = 256
  qmp_error_recheck = Unable to add CPU:.*, max allowed:.*
- invalid_vcpuid:
  cpuid_hotplug_vcpu0 = -1
  qmp_error_recheck = Invalid parameter type.*, expected:.*
- cpuid_already_exist:
  cpuid_hotplug_vcpu0 = 1
  qmp_error_recheck = Unable to add CPU:.*, it already exists
qemu-img bench

- “Run a simple sequential I/O benchmark on the specified image.”
- “A total number of **count** I/O requests is performed”
## Number of I/O requests - Actual

<table>
<thead>
<tr>
<th>Invalid (smaller than minimum required)</th>
<th>Valid</th>
<th>Invalid (larger than maximum allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>INT_MAX</td>
</tr>
<tr>
<td></td>
<td>INT_MAX + 1</td>
<td></td>
</tr>
</tbody>
</table>
### Number of I/O requests - Suggested

<table>
<thead>
<tr>
<th>Invalid (smaller than minimum required)</th>
<th>Valid</th>
<th>Invalid (larger than maximum allowed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1..UINT_MAX</td>
<td>UINT_MAX + 1</td>
</tr>
</tbody>
</table>
Combinatorial Testing

• Also known as “pair-wise”
• Principle is to have at least a pair of unique values in a test case
• Good values can use Equivalent Classes and Boundary Analysis
• Combinatorial can optimally test all values on a single test plan execution
Combinatorial Testing

Source: https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software
// qemu-img convert command line options

[--object objectdef] [--image-opts]

[-c]

[-p]

[-q]

[-n]

[-f fmt]

[-t cache]

[-T src_cache]

[-O output_fmt]

[-o options]

[-s snapshot_id_or_name]

[-l snapshot_param]

[-S sparse_size]

[-m num_coroutines]

[-W] filename [filename2 [...]] output_filename
// qemu-img convert command line options

[--object objectdef]  [--image-opts]
[-c]
[-p]
[-q]
[-n]
[-f fmt]
[-t cache]
[-T src_cache]
[-O output_fmt]
[-o options]
[-s snapshot_id_or_name]
[-l snapshot_param]
[-S sparse_size]
[-m num_coroutines]
[-W] filename [filename2 [...]] output_filename