redhat.

Testing Techniques Applied to Virt Development

Cleber Rosa
Sr. Software Engineer
Oct. 26Th, 2017

AGENDA

- Software Testing Basics
- Equivalence Partitioning
- Boundary Value Analysis
- Combinatorial Testing
- Applying the Theory

Glenford J. Myers' Triangle Check

- Takes as input: lengths of a triangle's sides
- Outputs the triangle classification:
  - Equilateral
  - Isosceles
  - Scalene
- How hard can it be to write a comprehensive set of test cases?

Triangle Check – First (Naive) Version

```
def triangle_check(a, b, c):
    if a == b == c:
        return "equilateral"
    elif a != b != c:
        return "scalene"
    else:
        return "isoceles"
```

Triangle Check Basic Test Cases

INPUT	EXPECTED OUTCOME
1, 1, 1	Equilateral
2, 2, 3	Isosceles
3, 4, 5	Scalene

Triangle Check – Test for First (Naive) Version

```
class Triangle(Test):

    def test_equilateral(self):
        self.assertEqual(triangle_check(1, 1, 1), "equilateral")

    def test_isosceles(self):
        self.assertEqual(triangle_check(2, 2, 3), "isosceles")

    def test_scalene(self):
        self.assertEqual(triangle_check(3, 4, 5), "scalene")
```

Triangle Check - Another Basic Test Case

INPUT	EXPECTED OUTCOME
1, 1, 1	Equilateral
2, 2, 3	Isosceles
2, 3, 2	Isosceles
3, 4, 5	Scalene

Triangle Check – Extra Test for First Version

```
class Triangle(Test):

    def test_equilateral(self):
        self.assertEqual(triangle_check(1, 1, 1), "equilateral")

    def test_isosceles(self):
        self.assertEqual(triangle_check(2, 2, 3), "isosceles")
        self.assertEqual(triangle_check(3, 2, 3), "isosceles")

    def test_scalene(self):
        self.assertEqual(triangle_check(3, 4, 5), "scalene")
```

Triangle Check Error Test Cases

INPUT	EXPECTED OUTCOME
0, 1, 1	Error
-1, 1, 1	Error
1, 1, 2	Error (not isosceles)
1, 2, 3	Error (not scalene)

Triangle Check – Invalid Individual Lengths

```
class Triangle(Test):

    ...

    def test_no_length(self):
        self.assertEqual(triangle_check(0, 1, 1), "error")
        self.assertEqual(triangle_check(-1, 1, 1), "error")
```

Triangle Check – Invalid Lengths for a Triangle

```
class Triangle(Test):

    ...

    def test_sum_2_sides_larger_3rd(self):
        self.assertEqual(triangle_check(1, 1, 2), "error")
        self.assertEqual(triangle_check(1, 2, 3), "error")
```

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
- If the software we write is not this simple, we should do better than in this example
- 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

Equivalence Partitioning

- 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

Identifying Input Types and Equivalent Classes

```
// 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 Types and Classes

Number of CPUs

INVALID	VALID	INVALID
<= 0	1 .. 256	>= 257

CPU IDs

INVALID	VALID	INVALID
<= -1	0 .. 255	>= 256

Boundary Analysis

- Also not scary
- When input classes are ordered, you can easily spot them
- These values are usually very good bets for tests

Boundary Values

Number of CPUs

INVALID	VALID		INVALID
0	1	256	257

CPU IDs

INVALID	VALID		INVALID
-1	0	255	256

Boundary Values in Tests

```
// snippets from tp-qemu/qemu/tests/cfg/cpu_add.cfg
smp = 4
vcpu_maxcpus = 255
variants:
    - cpuid_outof_range:
        cpuid_hotplug_vcpu0 = 255
        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
```

Boundary Analysis in “qemu-img bench”

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

qemu-img bench – number of I/O requests

Actual

INVALID	VALID		INVALID
-1	1	INT_MAX	INT_MAX + 1

Perceived

INVALID	VALID		INVALID
0	1	UINT_MAX	UINT_MAX + 1

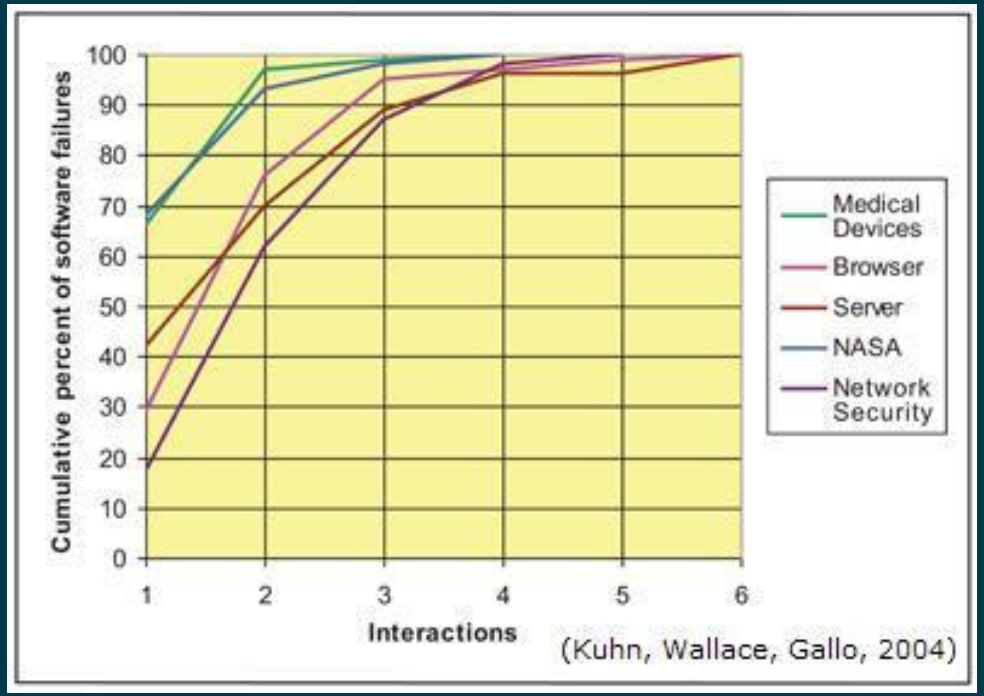
Combinatorial Testing

- Also known as “pair-wise”
- Basic principle is that the generated test cases will have **at least** a pair of unique values
- Good values can use Equivalent Classes and Boundary Analysis
- Combinatorial will **optimally** test all unique values on a single test plan execution

The more knobs you have...

Source: https://en.wikipedia.org/wiki/Cockpit#/media/File:Airbus_A380_cockpit.jpg

Combinatorial Testing Results in the Industry



Source: https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software

qemu-img convert knobs

```
[--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 knobs – let's pick some

```
[--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
```

Picking a tool - pict

- pict, as in “**P**airwise **I**ndependent **C**ombinatorial **T**esting”
- “PICT generates test cases and test configurations”
- “With PICT, you can generate tests that are more effective than manually generated tests and in a fraction of the time required by hands-on test case design.”
- https://github.com/microsoft/pict

qemu-img convert parameter file

```
# Interesting values (at boundaries): -1, 0b, 92233720368547758O8b (== 8EiB
exbibytes), 8EiB+1b
# Values that shows some issues: 0b
# Safe values: 2b, 1M, 1G
input_size: 2b, 1M, 1G

# qemu-img convert parameters
fmt: parallels, qcow, qcow2, qed, raw, vdi, vhdx, vmdk, vpc
output_fmt: parallels, qcow, qcow2, qed, raw, vdi, vhdx, vmdk, vpc
src_cache: off, writeback, unsafe, writethrough
cache: off, writeback, unsafe, writethrough
```

Picking a test runner - avocado

- Versatile test runner
- Long time history (heritage) in virtualization testing
  - KVM Autotest, virt-test, Avocado-VT
- Built-in support for “test variants”
- Proof of concept integration with pict:
  - https://github.com/avocado-framework/avocado/pull/2050
- More info:
  - http://avocado-framework.github.io/
  - https://github.com/avocado-framework/avocado
  - https://avocado-framework.readthedocs.io

What's Next?

- Make Combinatorial Testing (and other tools and techniques) easily available to Virt developers
  - Packaging pict in for Fedora and EPEL
  - Upstream work in QEMU
- Combinatorial Testing Implementation
  - Merge the pict integration into Avocado's next release
  - Another Implementation to be made native to Avocado
    - Currently in development
    - Collaboration with the Czech technical university in Prague
    - Let the best solution win!
- Special Interest Group?

redhat.

THANK YOU