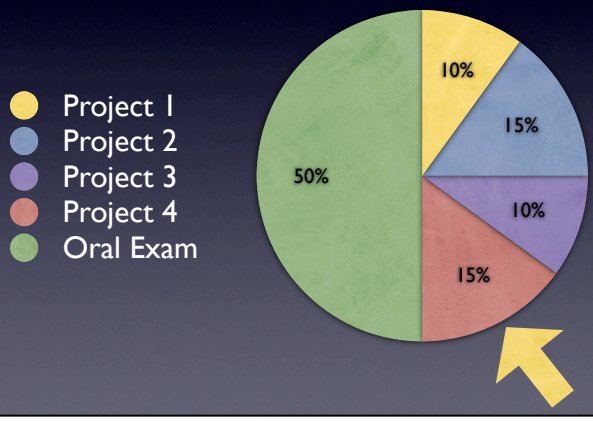
Project 4
Cause-Effect Chains

Andrzej Wasylkowski

Grading



Your Task

- Collect program states
- Compare states of a passing and failing run
- Find failure-inducing state differences
- Use the differences to locate and fix the defect in XMLProc
- Implement extensions

Input & Output

- Your tool must be called howcome.py and be runnable as follows:

```
$ python howcome.py UNIT_TEST PASSING_TEST FAILING_TEST LOCATIONS_TXT
```

- For each location output failure-inducing state differences

Sample Test Case (1)

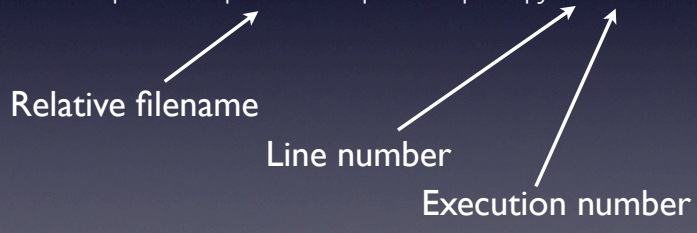
```
class TestMiddle (unittest.TestCase):

    def run_middle (self, elements):
      m = sorted (elements)[1]
      self.assertEqual (middle.middle (*elements), m)

    def testPass (self):
      self.run_middle ([1, 2, 3])

    def testFail (self):
      self.run_middle ([2, 1, 3])
```

Sample Test Case (2)

```
class TestXMLProc (unittest.TestCase):

    def runXMLProc (self, input):
      try:
        p = xmlproc.XMLProcessor ()
        p.reset ()
        p.feed (buffer)
        p.flush ()
      except UnboundLocalError:
        self.fail ("Failed with UnboundLocalError")

    def testPass (self):
      self.runXMLProc ('<?m?><!DOCTYPEl PUBLIC"""><a>&#10;</a>')

    def testFail (self):
      self.runXMLProc ('<?m?><!DOCTYPEl PUBLIC"""><a>&#xa;</a>')
```

Sample Locations File

```
xmlproc/xml/parsers/xmlproc/xmlutils.py:185:1
xmlproc/xml/parsers/xmlproc/xmlproc.py:85:4
xmlproc/xml/parsers/xmlproc/xmlproc.py:391:1
xmlproc/xml/parsers/xmlproc/xmlproc.py:400:1
```



Collecting Program States

- Use the tracing function to trace all lines
- Collect states at given locations
  - State = Backtrace
  - Backtrace = set of Frames
  - Frame = location + variables (global & local)

Pitfalls To Avoid (1)

- Cut the backtrace at the test function
- Keep track of how often each line was executed
- Put deep copies of variables to the Frames
  - Take aliasing into account

Creating Deep Copies

```
def get_deep_copy (value, memo):
  # avoid recursion depth problems
  if type (value) == types.ModuleType:
    return value
  try:
    # try to create a deep copy
    result = copy.deepcopy (value, memo)
    return result
  except:
    # didn't work, return the value itself
    return value
```

Comparing States

- StateDelta = set of Deltas
  - Delta = difference of a single variable
- Compare states of base variables only
  - Do not investigate objects' structure
- Have code to output Deltas in a human-readable way

Pitfalls To Avoid (2)

- Compare state at the same locations
  - Same location, same execution number
- Three types of Deltas
  - Addition
  - Change
  - Deletion

Finding Failure-Inducing State Differences

- Extend the Delta class with an apply method
- For each location
  - Use Delta Debugging on Deltas
  - Report minimal sets of Deltas found

Pitfalls To Avoid (3)

- Transform the passing run into a failing run
- Read the variables (e.g., frame.f_locals) once, then apply Deltas
  - Reading the variables after applying Deltas will undo the changes
- Be sure to again deep copy variables

Pitfalls To Avoid (4)

- Only changes to the top-level frame are transformed back to the program
  - Apply changes to the top-level frame
  - Defer other changes
  - Apply deferred changes when at the right frame

Test Data

- Apply your tool to the XMLProc parser
- Use the test case shown before
- Where does the problem with input lie?
- How can the XMLProc parser be fixed?

Sample Output For middle.py

```
$ cat locs.txt
middle.py:6:1
$ python howcome.py MiddleTests.TestMiddle testPass testFail locs.txt
Traceback (most recent call last):
…
AssertionError: 1 != 2

----------------------------------------
Location: middle.py:6 (1st hit)
Minimal deltas:
    Change at depth 0 local y: 2 -> 1
----------------------------------------
```

Extension 1: Finding Compatible States

- Output a file compatible_states.txt
- Output pairs of compatible states
  - Backtraces of the same length
  - Respective frames at the same locations (filename + line, not execution number)

Extension 1: Finding Compatible States

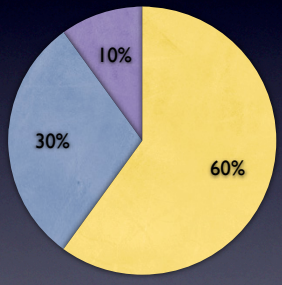
```
…
xmlproc/xml/parsers/xmlproc/xmlapp.py:146:1 == \
  xmlproc/xml/parsers/xmlproc/xmlapp.py:146:1
xmlproc/xml/parsers/xmlproc/xmlapp.py:169:1 == \
  xmlproc/xml/parsers/xmlproc/xmlapp.py:169:1
…
```

Extension 2: Identifying Cause Transitions

- See Cleve and Zeller, "Locating Causes of Program Failures" (ICSE 2005)
- How can compatible states be used for this?
- Sketch an algorithm

Project Grading

- Isolating Cause-Effect Chains
- Finding Compatible States
- Identifying Cause Transitions



Submission

- 2009-02-13 18:59
- Send .zip archive to: wasylkowski@cs.uni-saarland.de
  - Subject should start with [Project 4]
  - Input and output exactly as prescribed
  - Source code should be documented