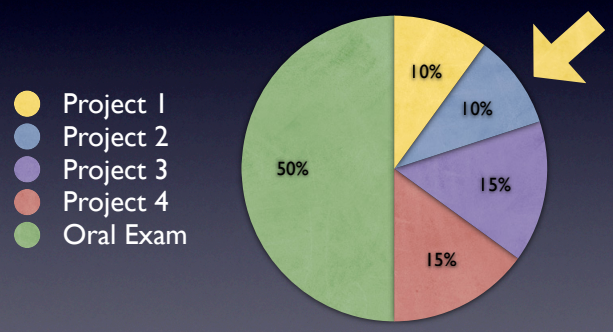
# Project 2
# Comparing Coverage

Andrzej Wasylkowski

1

# Grading




2

# Your Task

- Obtain coverage information
- Compare coverage information to detect anomalies
  - middle.py
  - XMLProc
- Implement an advanced method

3

# Obtaining Coverage

1. Set the tracing function
2. Invoke the program to be analyzed
3. Output coverage information

4

# Setting the Trace Function

```
def tracefunc (frame, event, arg):
  if event == "line":
    # get the location
    filename = frame.f_code.co_filename
    line = frame.f_lineno

    # make sure this is a program file
    if not filename.startswith (call_dir):
       return tracefunc

    # obtain the relative filename
    filename = filename[len (call_dir):]

    # IMPLEMENT: add to data structure

  return tracefunc
```

5

# Invoking the Program to Be Analyzed

```
# get the directory, where the tool was invoked
call_dir = os.path.abspath (sys.path[0])
if call_dir[-1] != os.sep:
  call_dir = call_dir + os.sep

# set the environment
sys.path[0] = os.path.abspath (os.path.dirname (program_to_analyze))
import __main__
sys.setrace (tracefunc)

# invoke the program
try:
  execfile (os.path.abspath (program_to_analyze), __main__.__dict__)
finally:
  # output the coverage obtained
```

6

# Input & Output

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

```
$ python get_coverage.py OUTPUT_FILE PROGRAM [ARGS]
```

- Output lines that were executed
- Omit lines not belonging to the program



# Obtaining Coverage: Example 1

```
$ python get_coverage.py middle.cov middle.py 2 1 3
```



# Obtaining Coverage: Example 1

```
$ python get_coverage.py middle.cov middle.py 2 1 3
middle: 1
$
```

# Obtaining Coverage: Example 1

```
$ python get_coverage.py middle.cov middle.py 2 1 3
middle: 1
$ cat middle.cov
```

10

# Obtaining Coverage: Example 1

```
$ python get_coverage.py middle.cov middle.py 2 1 3
middle: 1
$ cat middle.cov
middle.py:10
middle.py:11
middle.py:18
middle.py:20
middle.py:21
middle.py:22
middle.py:23
…
```

11

# Obtaining Coverage: Example 2

```
$ python get_coverage.py xmlproc.cov xmlproc/xpcmd.py input.xml
```

12

# Obtaining Coverage: Example 2

```
$ python get_coverage.py xmlproc.cov xmlproc/xpcmd.py input.xml
xmlproc version 0.70

Parsing 'input.xml'
Parse complete, 0 error(s) and 0 warning(s)
$
```

13

# Obtaining Coverage: Example 2

```
$ python get_coverage.py xmlproc.cov xmlproc/xpcmd.py input.xml
xmlproc version 0.70

Parsing 'input.xml'
Parse complete, 0 error(s) and 0 warning(s)
$ cat xmlproc.cov
```

14

# Obtaining Coverage: Example 2

```
$ python get_coverage.py xmlproc.cov xmlproc/xpcmd.py input.xml
xmlproc version 0.70

Parsing 'input.xml'
Parse complete, 0 error(s) and 0 warning(s)
$ cat xmlproc.cov
…
xmlproc/outputters.py:58
xmlproc/outputters.py:6
xmlproc/xml/__init__.py:1
xmlproc/xml/parsers/__init__.py:1
xmlproc/xml/parsers/xmlproc/__init__.py:1
xmlproc/xml/parsers/xmlproc/charconv.py:103
xmlproc/xml/parsers/xmlproc/charconv.py:105
…
```

15

# Comparing Coverage

1. Read the coverage data
2. Compare the coverage of passing and failing runs
3. Output the coverage comparison
4. (Optional) Output graphical coverage comparison

16

# Input & Output

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

```
$ python diff_coverage.py PASSING_SET FAILING_SET OUTPUT_FILE
```

- PASSING_SET contains names of passing runs coverage files
- FAILING_SET contains names of failing runs coverage files

17

# Plain Output Comparison

- Present information in a plain file
  - Percentage of test cases that covered this line (out of all test cases)
  - Percentage of failing test cases that covered this line (out of those test cases that covered this line)

18

# Plain Output Comparison: Example

- 5 test cases (3 passing, 2 failing)
- Line covered by 2 test cases (1 passing, 1 failing):
  - 2/5 test cases covered the line = 40%
  - 1/2 test cases that covered the line were failing = 50%

19

# Comparing Coverage: Example 1

```
$ cat middle_p.txt
middle_p1.cov
middle_p2.cov
middle_p3.cov
$ cat middle_f.txt
middle_f1.cov
middle_f2.cov
```

20

# Comparing Coverage: Example 1

```
$ python diff_coverage.py middle_p.txt middle_f.txt diff.txt
```

21

# Comparing Coverage: Example 1

```
$ python diff_coverage.py middle_p.txt middle_f.txt diff.txt
$ cat diff.txt
```

22

# Comparing Coverage: Example 1

```
$ python diff_coverage.py middle_p.txt middle_f.txt diff.txt
$ cat diff.txt
middle.py:10 : 60% / 66%
middle.py:11 : 60% / 66%
middle.py:13 : 20% / 0%
middle.py:14 : 20% / 0%
middle.py:18 : 100% / 40%
middle.py:20 : 100% / 40%
middle.py:21 : 100% / 40%
middle.py:22 : 100% / 40%
middle.py:23 : 100% / 40%
…
```

That many test cases that covered this line were failing

That many test cases covered this line

23

# Graphical Output Comparison (1)

- Output a file coverage.html
- Use hue and brightness to highlight lines, in the style of the Tarantula tool
  - hue(s) = red hue + %passed(s) / (%passed(s) + %failed(s)) * hue range
  - bright(s) = max (%passed(s), %failed(s))

24

# Graphical Output Comparison (2)

- Red hue = 0
- Hue range = 0.33 for colors in the range from red to green
- Use Python's colorsys package and the hsv_to_rgb function (with 1.0 saturation and bright as the v parameter)



# Graphical Output Comparison (3)

- Output each source line as a separate HTML line, appropriately colored (or gray, if never executed)
- Remember to escape special HTML characters
  - Python's cgi module has an escape function



# Graphical Output Comparison: example

**middle.py**

```
#!/usr/bin/env python

import sys

def middle(x, y, z):
    m = z;
    if y < z:
        if x < y:
            m = y
        elif x < z:
            m = y
    else:
        if x > y:
            m = y
        elif x > z:
            m = x

    return m

if __name__ == "__main__":
    x = sys.argv[1]
    y = sys.argv[2]
    z = sys.argv[3]
    m = middle(x, y, z)

    print "middle:", m
```

# Test Data

- Apply your tools to two programs
  - The *middle* program shown in the lecture
  - The *XMLProc* parser from Project 1
- Where does coverage information point to as the reasons for the failures?

28

# The *middle* Program

- You do not need to create unit tests
- Contrast the failing run with the passing runs
- Passing runs
  - (1,2,3), (2,4,1), (3,2,1),(3,3,5),(5,3,4),(5,5,5)
- Failing run
  - (2,1,3)

29

# The *XMLProc* Parser

- The XMLdata archive contains passing and failing test input files
- When parsed, the failing files issue warnings and errors
- For each of the three failing inputs, demonstrate how their coverage differs from the coverage of passing inputs

30

# Implementing an Advanced Method

- Nearest Neighbour
  Renieris and Reiss, "Fault Localization with Nearest Neighbour Queries" (ASE 2002)
- Call / Location Sequences
  Dallmeier, Lindig, and Zeller, "Lightweight Defect Localization for Java" (ECOOP 2005)



# Nearest Neighbour

- Extend your get_coverage.py tool:

```
$ python diff_coverage.py -nn PASSING_SET FAILING_SET OUTPUT_FILE
```

- Output the nearest passing run
  - If many are nearest, output the first one
- Is this technique more effective?



# Nearest Neighbour: Example 1

```
$ cat middle_p.txt
middle_p1.cov
middle_p2.cov
middle_p3.cov
$ cat middle_f1.txt
middle_f1.cov
```

# Nearest Neighbour: Example 1

```
$ python diff_coverage.py -nn middle_p.txt middle_f1.txt diff.txt
```

34

# Nearest Neighbour: Example 1

```
$ python diff_coverage.py -nn middle_p.txt middle_f1.txt diff.txt
Nearest passing run: middle_p2.cov
```

35

# Nearest Neighbour: Example 1

```
$ python diff_coverage.py -nn middle_p.txt middle_f1.txt diff.txt
Nearest passing run: middle_p2.cov
$ cat diff.txt
```

36

# Nearest Neighbour: Example 1

```
$ python diff_coverage.py -nn middle_p.txt middle_f1.txt diff.txt
Nearest passing run: middle_p2.cov
$ cat diff.txt
middle.py:10 : 50% / 100%
middle.py:11 : 50% / 100%
middle.py:18 : 100% / 50%
middle.py:20 : 100% / 50%
middle.py:21 : 100% / 50%
middle.py:22 : 100% / 50%
middle.py:23 : 100% / 50%
…
```

Remember: compare only two runs

37

# Call / Location Sequences

- Add two new tools
- Collect sequences of locations / calls

```
$ python get_sequences.py [-stmt|-call] WINDOW_SIZE
OUTPUT_FILE PROGRAM [ARGS]
```

- Output sequences from the failing run only

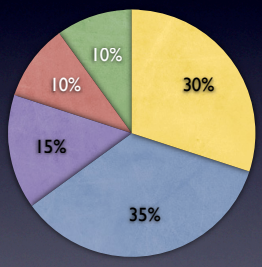
```
$ python diff_sequences.py
PASSING_SEQS FAILING_SEQS OUTPUT_FILE
```

- Look into the handout for details on format

38

# Project Grading

- Obtaining Coverage
- Comparing Coverage
- Graphical Coverage Comparison
- Nearest Neighbour
- Call / Location Sequences




39

# Submission

- 2008-12-19 23:59
- Send .zip archive to: wasylkowski@st.cs.uni-sb.de
  - Subject should start with [Project 2]
  - Input and output exactly as prescribed
  - Source code should be documented

40



41