

# CAS 741, CES 741 (Development of Scientific Computing Software)

Fall 2017

## 10 Verification and Validation Continued

Dr. Spencer Smith

Faculty of Engineering, McMaster University

October 16, 2017



# Verification and Validation Continued

- Administrative details
- Questions?

# Administrative Details

- GitHub issues for colleagues
  - ▶ Assigned 1 colleague (see Repos.xlsx in repo)
  - ▶ Provide at least 5 issues on their SRS
- Reading week, no 741 classes
- V&V template updated in repo

# Administrative Details: Deadlines

|                        |              |                |
|------------------------|--------------|----------------|
| <b>SRS Issues</b>      | Reading week | Oct 10         |
| <b>V&amp;V Present</b> | Week 06      | Week of Oct 16 |
| <b>V&amp;V Plan</b>    | Week 07      | Oct 25         |
| MG Present             | Week 08      | Week of Oct 30 |
| MG                     | Week 09      | Nov 8          |
| MIS Present            | Week 10      | Week of Nov 13 |
| MIS                    | Week 11      | Nov 22         |
| Impl. Present          | Week 12      | Week of Nov 27 |
| Final Documentation    | Week 13      | Dec 6          |

# Administrative Details: Presentation Schedule

- V&V Present
  - ▶ **Tuesday: Steven, Alexandre P., Alexander S.**
  - ▶ **Friday: Geneva, Jason, Yuzhi**
- MG Present
  - ▶ Tuesday: Xiaoye, Shusheng, Devi, Keshav, Alex P, Paul
  - ▶ Friday: Yuzhi, Jason, Geneva, Alex S, Isobel, Steven
- MIS Present
  - ▶ Tuesday: Isobel, Keshav, Paul
  - ▶ Friday: Shusheng, Xiaoye, Devi
- Impl. Present
  - ▶ Tuesday: Alexander S., Steven, Alexandre P.
  - ▶ Friday: Jason, Geneva, Yuzhi

# Questions?

- Questions about SRS?
- Questions about V&V?

# White-box Testing

- Intuitively, after running your test suites, what percentage of the lines of code in your program should be exercised?

# White-box Coverage Testing

- (In)adequacy criteria - if significant parts of the program structure are not tested, testing is inadequate
- Control flow coverage criteria
  - ▶ Statement coverage
  - ▶ Edge coverage
  - ▶ Condition coverage
  - ▶ Path coverage

Examples that follow are from [\[1\]](#)



# Statement-Coverage Criterion

- Select a test set  $T$  such that every elementary statement in  $P$  is executed at least once by some  $d$  in  $T$
- An input datum executes many statements - try to minimize the number of test cases still preserving the desired coverage

## Example

```
read (x); read (y);  
if x > 0 then  
    write ("1");  
else  
    write ("2");  
end if;  
if y > 0 then  
    write ("3");  
else  
    write ("4");  
end if;
```

How would you write a test case?

What is the minimum number of test cases?

## Example

```
read (x); read (y);  
if x > 0 then  
    write ("1");  
else  
    write ("2");  
end if;  
if y > 0 then  
    write ("3");  
else  
    write ("4");  
end if;
```

**$\{ \langle x = 2, y = -3 \rangle, \langle x = -13, y = 51 \rangle, \langle x = 97, y = 17 \rangle, \langle x = -1, y = -1 \rangle \}$   
covers all statements**

**$\{ \langle x = -13, y = 51 \rangle, \langle x = 2, y = -3 \rangle \}$   
is minimal**

## Weakness of the Criterion

```
if x < 0 then  
    x := -x;  
end if;  
z := x;
```

$\{<x=-3>\}$  covers all statements. Why is this not enough?

## Weakness of the Criterion

```
if x < 0 then  
    x := -x;  
end if;  
z := x;
```

$\{x < -3\}$  covers all  
statements

it does not exercise the  
case when x is positive  
and the then branch is  
not entered

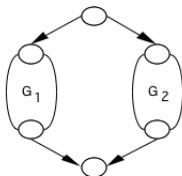
# Edge-Coverage Criterion

- Select a test set  $T$  such that every edge (branch) of the control flow is exercised at least once by some  $d$  in  $T$
- This requires formalizing the concept of the control graph and how to construct it
  - ▶ Edges represent statements
  - ▶ Nodes at the ends of an edge represent entry into the statement and exit

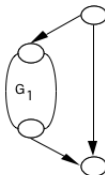
# Control Graph Construction Rules



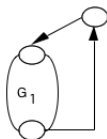
I/O, assignment,  
or procedure call



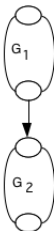
if-then-else



if-then



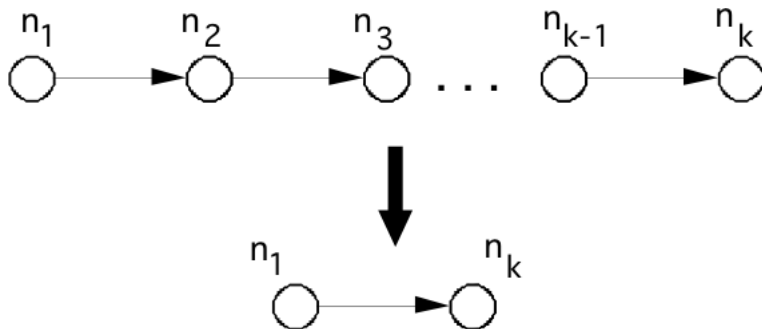
while loop



two sequential  
statements

# Simplification

A sequence of edges can be collapsed into just one edge





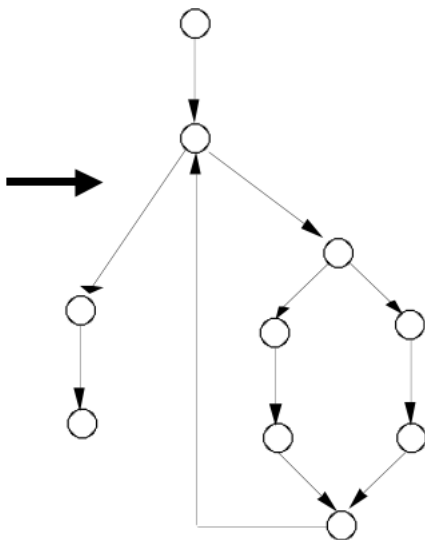
# Example: Euclid's Algorithm

```
begin
  read (x); read (y);
  while  $x \neq y$  loop
    if  $x > y$  then
       $x := x - y$ ;
    else
       $y := y - x$ ;
    end if;
  end loop;
  gcd := x;
end;
```

Draw the control  
flow graph

# Example: Euclid's Algorithm

```
begin
  read (x); read (y);
  while  $x \neq y$  loop
    if  $x > y$  then
       $x := x - y$ ;
    else
       $y := y - x$ ;
    end if;
  end loop;
  gcd := x;
end;
```



# Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
    if table(counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

# Weakness

```
found := false; counter := 1;
while (not found) and counter < number_of_items loop
    if table(counter) = desired_element then
        found := true;
    end if;
    counter := counter + 1;
end loop;
if found then
    write ("the desired element is in the table");
else
    write ("the desired element is not in the table");
end if;
```

test cases: (1) empty table, (2) table with 3 items, second of which is the item to look for

Do not discover the error ( $<$  instead of  $\leq$ )

```
if c1 and c2 then  
    st;  
else  
    sf;
```

*// equivalent to*

```
if c1 then  
    if c2 then  
        st;  
    else  
        sf;  
else  
    sf;
```

# Condition-Coverage Criterion

- Select a test set  $T$  such that every edge of  $P$ 's control flow is traversed and all possible values of the constituents of compound conditions are exercised at least once
- This criterion is finer than edge coverage

# Weakness

```
if  $x \neq 0$  then
     $y := 5$ ;
else
     $z := z - x$ ;
end if;
if  $z > 1$  then
     $z := z / x$ ;
else
     $z := 0$ ;
end if;
```

$\{ \langle x = 0, z = 1 \rangle, \langle x = 1, z = 3 \rangle \}$   
causes the execution of all edges,  
but fails to expose the risk of a  
division by zero

# Path-Coverage Criterion

- Select a test set  $T$  that traverses all paths from the initial to the final node of  $P$ 's control flow
- It is finer than the previous kinds of coverage
- However, number of paths may be too large, or even infinite (see while loops)
- Loops
  - ▶ Zero times (or minimum number of times)
  - ▶ Maximum times
  - ▶ Average number of times



# The Infeasibility Problem

- Syntactically indicated behaviours (statements, edges, etc.) are often impossible
- Unreachable code, infeasible edges, paths, etc.
- Adequacy criteria may be impossible to satisfy
  - ▶ Manual justification for omitting each impossible test case
  - ▶ Adequacy “scores” based on coverage - example 95 % statement coverage

## Further Problem

- What if the code omits the implementation of some part of the specification?
- White box test cases derived from the code will ignore that part of the specification!

# Testing Boundary Conditions

- Testing criteria partition input domain in classes, assuming that behavior is “similar” for all data within a class
- Some typical programming errors, however, just happen to be at the boundary between different classes
  - ▶ Off by one errors
  - ▶  $<$  instead of  $\leq$
  - ▶ equals zero

# Criterion

- After partitioning the input domain  $D$  into several classes, test the program using input values not only “inside” the classes, but also at their boundaries
- This applies to both white-box and black-box techniques
- In practice, use the different testing criteria in combinations

# The Oracle Problem

When might it be difficult to know the “expected” output/behaviour?

# The Oracle Problem

- Given input test cases that cover the domain, what are the expected outputs?
- Oracles are required at each stage of testing to tell us what the right answer is
- Black-box criteria are better than white-box for building test oracles
- Automated test oracles are required for running large amounts of tests
- Oracles are difficult to design - no universal recipe

# The Oracle Problem Continued

- Determining what the right answer should be is not always easy
  - ▶ Scientific computing
  - ▶ Machine learning
  - ▶ Artificial intelligence

# The Oracle Problem Continued

What are some strategies we can use when we do not have a test oracle?



# Strategies Without An Oracle

- Using an independent program to approximate the oracle (pseudo oracle)
- Method of manufactured solutions
- Properties of the expected values can be easier than stating the expected output
  - ▶ Examples?

# Strategies Without An Oracle

- Using an independent program to approximate the oracle (pseudo oracle)
- Method of manufactured solutions
- Properties of the expected values can be easier than stating the expected output
  - ▶ Examples?
  - ▶ List is sorted
  - ▶ Number of entries in file matches number of inputs
  - ▶ Conservation of energy or mass
  - ▶ Expected trends in output are observed (metamorphic testing [5, 4, 6])
  - ▶ etc.

# Challenges Specific to Scientific Computing

- Unknown solution
- Approximation of real numbers
- Nonfunctional requirements
- Parallel computation

# Mutation Testing for SC

- Generate changes to the source code, called mutants, which become code faults
- Mutants include changing an operation, modifying constants, changing the order of execution, etc.
- The adequacy of a set of tests is established by running the tests on all generated mutants
- Need to account for floating point approximations
- See [3]

# Specific SC V&V Approaches

Summary in [10]

- Compare to closed-form solutions
- Method of manufactured solutions [8]
- Interval arithmetic [2]
- Convergence studies
- Compare to other program (parallel testing)

Can also consider using code inspection [7, 9]

# Specific SC V&V NonFunctional

- Installability, consider VMs
- Portability, consider VMs, Docker, CI
- Describe (rather than specify) impact of changing inputs
  - ▶ Accuracy
  - ▶ Performance
  - ▶ Relative comparison

# Validation Testing Report for PMGT

- Prepared by Wen Yu
- Do not know the correct solution, but know properties of the correct solution
- Automated correctness validation tests
  - ▶ The area of each element is greater than zero
  - ▶ The boundary of the mesh is closed
  - ▶ Vertices in a clockwise order
  - ▶  $nc + nv - ne = 1$
  - ▶ ...
- Visual correctness validation tests
  - ▶ No vertex outside the input domain
  - ▶ No vertex inside a cell
  - ▶ No dangling edges
  - ▶ All cells connected
  - ▶ The mesh is conformal

# Validation Testing Report for PMGT (Continued)

- List and description of test cases
- Test cases are labelled and numbered
- Traceability to SRS requirements
- Traceability to MG
- Summary of results
- Analysis of results
  - ▶ Focus on nonfunctional requirements
  - ▶ Speed



# Test Plan From BlankProjectTemplate

# References I



Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.

*Fundamentals of Software Engineering.*

Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.



Timothy Hickey, Qun Ju, and Maarten H. Van Emden.

Interval arithmetic: From principles to implementation.

*J. ACM*, 48(5):1038–1068, September 2001.



Daniel Hook and Diane Kelly.

Testing for trustworthiness in scientific software.

In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 59–64, Washington, DC, USA, 2009. IEEE Computer Society.

# References II



U. Kanewala and J. M. Bieman.

Techniques for testing scientific programs without an oracle.

*In Software Engineering for Computational Science and Engineering (SE-CSE), 2013 5th International Workshop on*, pages 48–57, May 2013.



Upulee Kanewala, James M. Bieman, and Asa Ben-Hur.

Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels.

*Software Testing Verification and Reliability*, preprint, 2015.

# References III



Upulee Kanewala and Anders Lundgren.

Automated metamorphic testing of scientific software.

In Jeffrey C. Carver, Neil Chue Hong, and George Thiruvathukal, editors, *Software Engineering for Science*, Chapman & Hall/CRC Computational Science, chapter Examples of the Application of Traditional Software Engineering Practices to Science, pages 151–174. Taylor & Francis, 2016.



Diane Kelly and Terry Shepard.

Task-directed software inspection technique: an experiment and case study.

In *CASCON '00: Proceedings of the 2000 conference of the Centre for Advanced Studies on Collaborative research*, page 6. IBM Press, 2000.

# References IV



Patrick J. Roache.

*Verification and Validation in Computational Science and Engineering.*

Hermosa Publishers, Albuquerque, New Mexico, 1998.



Terry Shepard and Diane Kelly.

How to do inspections when there is no time.

In *Proceedings of the 23rd International Conference on Software Engineering*, page 718. IEEE Computer Society, 2001.

# References V



W. Spencer Smith.

A rational document driven design process for scientific computing software.

In Jeffrey C. Carver, Neil Chue Hong, and George Thiruvathukal, editors, *Software Engineering for Science*, chapter Section I – Examples of the Application of Traditional Software Engineering Practices to Science, pages 33–63. Taylor & Francis, 2016.