

**SE 2AA4, CS 2ME3 (Introduction to Software
Development)**

Winter 2018

30 Introduction to Verification Continued (Ch. 6) DRAFT

Dr. Spencer Smith

Faculty of Engineering, McMaster University

December 15, 2017



30 Introduction to Verification Continued (Ch. 6) DRAFT

- Partially based on slides by Dr. Wassying, Ghezzi et al
- Administrative details
- Approaches to verification
- Goals of testing
- Test plan
- Types of test - white box, versus black box, manual versus automated, etc.

Administrative Details

TBD

Approaches to Verification

- What are some approaches to verification?
- How can we categorize these approaches?

Approaches to Verification

- Experiment with behaviour of product
 - ▶ Sample behaviours via testing
 - ▶ Goal is to find “counter examples”
 - ▶ **Dynamic** technique
 - ▶ Examples: unit testing, integration testing, acceptance testing, white box testing, stress testing, etc.
- Analyze product to deduce its adequacy
 - ▶ Analytic study of properties
 - ▶ **Static** technique
 - ▶ Examples: Code walk-throughs, code inspections, correctness proof, etc.

Does our Engineering Analogy Fail?

- If a bridge can hold 512 kN, can it hold 499 kN?
- If our software works for the input 512, will it work for 499?

Verification in Engineering

- Example of bridge design
- One test assures infinite correct situations
- In software a small change in the input may result in significantly different behaviour
- There are also chaotic systems in nature, but products of engineering design are usually stable and well-behaved

Modified Version Works for 512, but not 499

```
procedure binary-search (key: in element;  
                        table: in elementTable; found: out Boolean) is  
begin  
  bottom := table'first; top := table'last;  
  while bottom < top loop  
    if (bottom + top) rem 2  $\neq$  0 then  
      middle := (bottom + top - 1) / 2;  
    else  
      middle := (bottom + top) / 2;  
    end if;  
    if key  $\leq$  table (middle) then  
      top := middle;  
    else  
      bottom := middle + 1;  
    end if;  
  end loop;  
  found := key = table (top);  
end binary-search
```

if we omit this
the routine
works if the else
is never hit!
(i.e. if size of table
is a power of 2)



Testing and Lack of “Continuity”

- Testing samples behaviours by examining “test cases”
- Impossible to extrapolate behaviour of software from a finite set of test cases
- No continuity of behaviour - it can exhibit correct behaviour in infinitely many cases, but may still be incorrect in some cases

Goals of Testing

- If our code passes all test cases, is it now guaranteed to be error free?
- Are 5000 random tests always better than 5 carefully selected tests?

Goals of Testing

- To show the **presence** of bugs (Dijkstra, 1972)
- If tests do not detect failures, we cannot conclude that software is defect-free
- Still, we need to do testing - driven by sound and systematic principles
 - ▶ Random testing is often not a systematic principle to use
 - ▶ Need a test plan
- Should help isolate errors - to facilitate debugging

Goals of Testing Continued

- Should be repeatable
 - ▶ Repeating the same experiment, we should get the same results
 - ▶ Repeatability may not be true because of the effect of the execution environment on testing
 - ▶ Repeatability may not occur if there are uninitialized variables
 - ▶ Repeatability may not happen when there is nondeterminism
- Should be accurate
 - ▶ Accuracy increases reliability
 - ▶ Part of the motivation for formal specification
- Is a successful test case one that passes the test, or one that shows a failure?

Test Plan

- Given that no single verification technique can prove correctness, the practical approach is to use ALL verification techniques. Is this statement True or False?

Test Plan

- Testing can uncover errors and build confidence in the software
- Resources of time, people, facilities are limited
- Need to plan how the software will be tested
- You know in advance that the software is unlikely to be perfect
- You need to put resources into the most important parts of the project
- A risk analysis can determine where to put your limited resources
- A risk is a condition that can result in a loss
- Risk analysis involves looking at how bad the loss can be and at the probability of the loss occurring

Test Plan

- Risks cannot be eliminated, but the development process can reduce the probability of loss associated with risks to an “acceptable” level
- One approach to risk analysis is FMEA - Failure Mode Effect Analysis
- Consider the capstone project of the autonomous rescue robots (A3)
 - ▶ The final grade is mostly based on the final documentation and the final demonstration/presentation in front of an industry panel
 - ▶ What are the most significant risks?
 - ▶ How do you test the robot?
 - ▶ How should calibration be handled?

Test Plan For Autonomous Rescue Robot

- Consider the capstone project of the autonomous rescue robots
 - ▶ Largest risk, robot fails during final demonstrations
 - ▶ Test to improve reliability
 - ▶ Test results of great interest to IBM judges
 - ▶ Think about test cases, think about testing environment versus final environment

White Box Versus Black Box Testing

- Do you know (or can you guess) the difference between white box and black box testing?
- What if they were labelled transparent box and opaque box testing, respectively?

White Box Versus Black Box Testing

- White box testing is derived from the program's internal structure
- Black box testing is derived from a description of the program's function
- Should perform both white box and black box testing
- Black box testing
 - ▶ Uncovers errors that occur in implementing requirements or design specifications
 - ▶ Not concerned with how processing occurs, but with the results
 - ▶ Focuses on functional requirements for the system
 - ▶ Focuses on normal behaviour of the system

White Box Testing

- Uncovers errors that occur during implementation of the program
- Concerned with how processing occurs
- Evaluates whether the structure is sound
- Focuses on abnormal or extreme behaviour of the system

Dynamic Testing

- Is there a dynamic testing technique that can guarantee correctness?
- If so, what is the technique?
- Is this technique practical?

Dynamic Versus Static Testing

- Another classification of verification techniques, as previously discussed
- Use a combination of dynamic and static testing
- Dynamic analysis
 - ▶ Requires the program to be executed
 - ▶ Test cases are run and results are checked against expected behaviour
 - ▶ Exhaustive testing is the only dynamic technique that guarantees program validity
 - ▶ Exhaustive testing is usually impractical or impossible
 - ▶ Reduce number of test cases by finding criteria for choosing representative test cases

Static Testing Continued

- Static analysis
 - ▶ Does not involve program execution
 - ▶ Testing techniques simulate the dynamic environment
 - ▶ Includes syntax checking
 - ▶ Generally static testing is used in the requirements and design stage, where there is no code to execute
 - ▶ Document and code walkthroughs
 - ▶ Document and code inspections

Manual Versus Automated Testing

- What is the difference between manual and automated testing?
- What are the advantages of automated testing?
- What is regression testing?

Manual Versus Automated Testing

- Manual testing
 - ▶ Has to be conducted by people
 - ▶ Includes by-hand test cases, structured walkthroughs, code inspections
- Automated testing
 - ▶ The more automated the development process, the easier to automate testing
 - ▶ Less reliance on people
 - ▶ Necessary for regression testing
 - ▶ Test tools can assist, such as Junit, Cppunit, CuTest etc.
 - ▶ Can be challenging to automate GUI tests
 - ▶ Test suite for Maple has 2 000 000 test cases, run on 14 platforms, every night, automated reporting

Automated Testing at MapleSoft

- Three steps
 - ▶ Write the problem description
 - ▶ `result := solver(problem)`
 - ▶ `assert(result == expected)`
- Assert writes out code to reproduce any failures
- Track failures
 - ▶ Source code management (like CVS or Subversion)
 - ▶ Database of test cases, functions called
 - ▶ Database of source files, functions defined
 - ▶ Database of 40 days of timings and resources used
- Automatically sends an e-mail to the programmer and his/her boss

Continuous Integration Testing

- What is continuous integration testing?

Continuous Integration Testing

- Information available on [Wikipedia](#)
- Developers integrate their code into a shared repo frequently (multiple times a day)
- Each integration is automatically accompanied by regression tests and other build tasks
- Build server
 - ▶ Unit tests
 - ▶ Integration tests
 - ▶ Static analysis
 - ▶ Profile performance
 - ▶ Extract documentation
 - ▶ Update project web-page
 - ▶ Portability tests
 - ▶ etc.
- Avoids potentially extreme problems with integration when the baseline and a developer's code greatly differ

Continuous Integration Tools

- Gitlab
 - ▶ Example at [Rogue Reborn](#)
- Jenkins
- Travis
- Docker
 - ▶ Eliminates the “it works on my machine” problem
 - ▶ Package dependencies with your apps
 - ▶ A container for lightweight virtualization
 - ▶ Not a full VM

Fault Testing

- Common analogy involves planting fish in a lake to estimate the fish population
- T = total number of fish in the lake (to be estimated)
- N = fish stocked (marked) in the lake
- M = total number of fish caught in lake
- M' = number of marked fish caught
- $T = (M - M') * N / M'$
- Artificially seed faults, discover both seeded and new faults, estimate the total number of faults

Fault Testing Continued

- Method assumes that the real and seeded faults have the same distribution
- Hard to seed faults
 - ▶ By hand (not a great idea)
 - ▶ Independent testing by two groups and obtain the faults from one group for use by the other
- Want most of the discovered faults to be seeded faults
- If many faults are found, this is a bad thing
- The probability of errors is proportional to the number of errors already found

Fault Testing Usage Homework

After completion of a complex software project, two independent groups test it. The first group finds 300 errors and the second group finds 200 errors. A comparison of the errors discovered by the two teams shows that they found the same error in 50 cases. What is the range that estimates the number of errors in the original software project?

- A. 900–1000
- B. -17–25
- C. 600–1500
- D. 0–150
- E. 150–250