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

Fall 2017

09 Verification and Validation

Dr. Spencer Smith

Faculty of Engineering, McMaster University

October 3, 2018

Verification and Validation

- Administrative details
- Questions?
- 741 workflow
- Testing from SE perspective
- Testing from SC perspective
- V&V template
- V&V examples
  - SWHS
  - Mesh Gen
  - Rogue Reborn

Administrative Details

- SRS Presentation grades on Avenue
- Will provide some feedback on open issues
- GitHub issues for colleagues
  - Assigned 1 colleague (see `Repos.xlsx` in repo)
  - Provide at least 5 issues on their SRS
  - Grading
    - Not enough issues, or poor issues 0/2
    - Enough issues, but shallow 1/2
    - Enough issues and deep (not surface) 2/2
  - Due by Tuesday, Oct 9, 11:59 pm
- Reading week next week, no 741 classes
- V&V template will be updated in repo

Administrative Details: Deadlines

SRS or CA	Week 05	Oct 4
Syst. VnV Present	Week 06	Week of Oct 15
System VnV Plan	Week 07	Oct 22
MG Present	Week 08	Week of Oct 29
MG	Week 09	Nov 5
MIS Present	Week 10	Week of Nov 12
MIS	Week 11	Nov 19
Unit VnV or Impl. Present	Week 12	Week of Nov 26
Unit VnV Plan	Week 13	Dec 3
Final Doc	Week 14	Dec 10

Administrative Details: Presentation Schedule

- Syst V&V Plan Present
  - Wednesday: Malavika, Robert
  - Friday: Hanane
- MG Present
  - Wednesday: Karol, Malavika, Robert, Hanane
  - Friday: Brooks, Vajiheh, Olu, Jennifer
- MIS Present
  - Wednesday: Malavika, Robert
  - Friday: Hanane, Jennifer
- Unit VnV Plan or Impl. Present
  - Wednesday: Brooks, Vajiheh
  - Friday: Olu, Karol

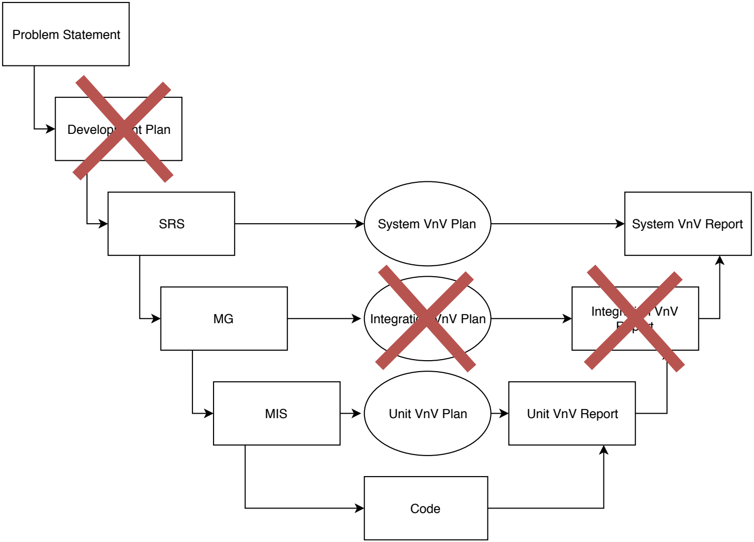
Invitation

- Malavika Srinivasan SE for SC Presentation
  - Friday, Oct 5, 3:30 pm – 5:00 pm in ITB/225
  - Motivational
  - We need your feedback
  - Will help Malavika with her MASc

Questions?

- Questions about SRS?
  - Where do variabilities go in CA template?
  - Functional versus nonfunctional requirements?

“Faked” Rational Design Process



Outline of Verification Topics

- What are the goals of verification?
- What are the main approaches to verification?
  - What kind of assurance do we get through testing?
  - Can testing prove correctness?
  - How can testing be done systematically?
  - How can we remove defects (debugging)?
- What are the main approaches to software analysis?
- Informal versus formal analysis

Incorrect Version of Delete

Using `s = new T[MAX_SIZE]`, for some type `T`

```
public static void del(int i)
{
  int j;

  for (j = i; j <= (length - 1); j++)
  {
    s[j] = s[j+1];
  }

  length = length - 1;
}
```

- What is the error?
- What test case would highlight the error?

Correct Version of Delete

```
public static void del(int i)
{
  int j;

  for (j = i; j < (length - 1); j++)
  {
    s[j] = s[j+1];
  }

  length = length - 1;
}
```

Avoids potential ArrayIndexOutOfBoundsException Exception

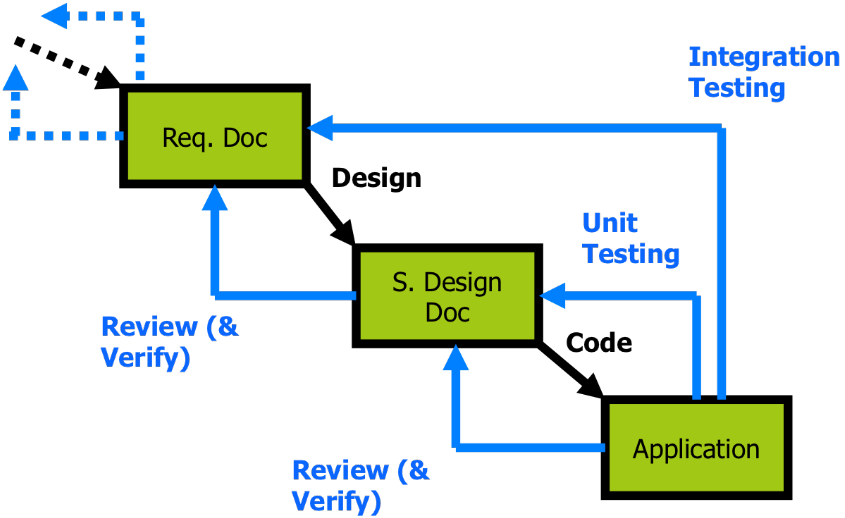
Verification Versus Validation

- What is the difference between verification and validation?

Verification Versus Validation

- Verification - Are we building the product right? Are we implementing the requirements correctly (internal)
- Validation - Are we building the right product? Are we getting the right requirements (external)
- According to Capability Maturity Model (CMM)
  - Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. [IEEE-STD-610]
  - Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [IEEE-STD-610]
- We will focus on verification

Verification Activities



Testing Phases

1. Unit testing
2. Integration testing
3. System testing
4. Acceptance testing

Need for Verification

- Designers are fallible even if they are skilled and follow sound principles
- We need to build confidence in the software
- Everything must be verified, every required functionality, every required quality, every process, every product, every document
- For every work product covered in this class we have discussed its verification
- Even verification itself must be verified

Properties of Verification

From [1]

- May not be binary (OK, not OK)
  - Severity of defect is important
  - Some defects may be tolerated
  - Our goal is typically acceptable reliability, not correctness
- May be subjective or objective - for instance, usability, generic level of maintainability or portability
  - How might we make usability objective?
- Even implicit qualities should be verified
  - Because requirements are often incomplete
  - For instance robustness, maintainability
- What is better than implicitly specified qualities?

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 ≠ 0 then
            middle := (bottom + top - 1) / 2;
        else
            middle := (bottom + top) / 2;
        end if;
        if key ≤ 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 (V&V) 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 (V&V) 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

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

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

Sample Nonfunctional System Testing

- Stress testing - Determines if the system can function when subject to large volumes
- Usability testing
- Performance measurement

Sample Functional System Testing

- Requirements: Determines if the system can perform its function correctly and that the correctness can be sustained over a continuous period of time
- Error Handling: Determines the ability of the system to properly process incorrect transactions
- Manual Support: Determines that the manual support procedures are documented and complete, where manual support involves procedures, interfaces between people and the system, and training procedures
- Parallel: Determines the results of the new application are consistent with the processing of the previous application or version of the application

Theoretical Foundations Of Testing: Definitions

- P (program), D (input domain), R (output domain)
  - P: D $\rightarrow$ R (may be partial)
- Correctness defined by OR $\subseteq$ D $\times$ R
  - P(d) correct if $\langle$ d, P(d) $\rangle \in$ OR
  - P correct if all P(d) are correct
- Failure
  - P(d) is not correct
  - May be undefined (error state) or may be the wrong result
- Error (Defect)
  - Anything that may cause a failure
    - Typing mistake
    - Programmer forgot to test "x=0"
- Fault
  - Incorrect intermediate state entered by program

Definitions Questions

- A test case t is an element of D or R?
- A test set T is a finite subset of D or R?
- How would we define whether a test is successful?
- How would we define whether a test set is successful?

Definitions Continued

- Test case t: An element of D
- Test set T: A finite subset of D
- Test is successful if P(t) is correct
- Test set successful if P correct for all t in T

Theoretical Foundations of Testing

- Desire a test set T that is a finite subset of D that will uncover all errors
- Determining and ideal T leads to several undecideable problems
- No algorithm exists:
  - To state if a test set will uncover all possible errors
  - To derive a test set that would prove program correctness
  - To determine whether suitable input exists to guarantee execution of a given statement in a given program
  - etc.

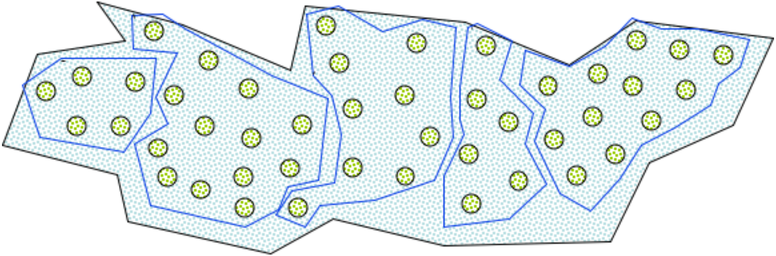
Empirical Testing

- Need to introduce empirical testing principles and heuristics as a compromise between the impossible and the inadequate
- Find a strategy to select significant test cases
- Significant means the test cases have a high potential of uncovering the presence of errors

Complete-Coverage Principle

- Try to group elements of D into subdomains D_1, D_2, ..., D_n where any element of each D_i is likely to have similar behaviour
- $D = D_1 \cup D_2 \cup \ldots \cup D_n$
- Select one test as a representative of the subdomain
- If $D_j \cap D_k = \emptyset$ for all $j \neq k$, (partition), any element can be chosen from each subdomain
- Otherwise choose representatives to minimize number of tests, yet fulfilling the principle

Complete-Coverage Principle



References I

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli.
Fundamentals of Software Engineering.
Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.