Verification and Validation

Objectives

- To introduce software verification and validation and to discuss the distinction between them
- To describe the code inspection process and its role in V & V

Verification vs. Validation

- Verification

  "Are we building the product right?"

  - The software should conform to its specification.

- Validation

  "Are we building the right product?"

  - The software should do what the user really requires.

How could a system possibly pass verification, but not validation?

The V&V Process

- Has two principal objectives
  - The discovery of defects in a system
  - The assessment of whether or not the system is usable in an operational situation
- Is a whole life-cycle process
  - Examples
    - Peer document reviews Verification
    - Customer document reviews Validation
    - SDD requirements matrix Verification
    - Code inspections Verification
    - Customer meetings Validation
    - Prototyping Verification and Validation

Mainly verification or validation? Or both?

V&V Goals

- V&V should establish confidence that the software is fit for its purpose.
  - This does not mean completely free of defects.
  - Rather, it must be good enough for its intended use.

V&V Confidence

Depends on:

- System purpose
  - The level of confidence depends on how critical the software is to an organization (e.g. safety critical).
- User expectations
  - Users may have low expectations of certain kinds of software.
- Marketing environment
  - Getting a product to market early may be more important than finding defects in the program.

Code Inspection

- Visually examine the source code
- Goals
  - To discover logical anomalies and defects. Examples:
    - uninitialized variables
    - unreachable code
    - infinite loops
  - To confirm compliance with coding and commenting conventions
- Intended for defect detection, not correction
- Very effective technique for discovering errors
- Saves time and money
  - The earlier in the development process an error is found, the better.

Inspection Pros

- Many different defects may be discovered in a single inspection
  - With testing, one defect may mask another so that several executions/tests are required.
- Inspections reuse domain and programming knowledge
  - Reviewers are likely to have seen the types of errors that commonly arise.

Inspection and Testing

- Inspection and testing are complementary techniques.
- Inspection can check conformance with a specification (verification), but not conformance with the customer's real requirements.
  - Testing can do this (validation).
- Inspections cannot check non-functional characteristics.

Inspection Preparation

- A precise specification must be available.
- Team members must be familiar with the organization's coding and commenting standards.
- Syntactically correct code must be available.
- An error checklist should be prepared.
- Management must accept that inspection will increase costs early in the software process.

Something to Consider …

A manager decides to use the reports of program inspections as an input to the staff appraisal process. These reports show who made and who discovered program errors.

- What do you think about this practice?
- Might this make a difference in the inspection process itself?

Inspection Procedure

- The inspection procedure is planned.
- A system overview is presented to the inspection team.
- Code and associated documents are distributed to the inspection team in advance.
- Inspection takes place and all discovered errors are noted.
- All modifications are made to repair discovered errors.
- Re-inspection may or may not be required.

Inspection Teams

- Made up of:
  - **Author** of the code being inspected
  - **Inspector** who finds errors, omissions, and inconsistencies
  - **Reader** who reads the code to the team
  - **Moderator** who chairs the meeting
  - **Scribe** who makes detailed notes regarding errors
- Roles may vary from these (e.g., Reader).
- Multiple roles may be taken on by the same member.

Inspection Checklist

- Checklist of common errors is used to drive the inspection
  - Is programming language dependent

How would a C or Phython checklist differ from a Java checklist?

Sample of a Partial Inspection Checklist

Category	Inspection Check
Data	**Are all variables initialized before they are used?** **Have all constants been named?**
Control	**For each conditional statement, is the condition correct?** **Will each loop terminate?** **In case statements, are all possible cases accounted for?**
Input/output	**Are all input variables used?** **Are all output variables assigned a value before they are output?**
Interface	**Do formal and actual parameters match in:** **- number?** **- data type?** **- what they represent?**
Exception management	**Have all possible error conditions been taken into account?**

Automated Static Analysis

- Static analyzers are software tools for source text processing.
  - They parse the program text and try to discover potentially erroneous conditions.
  - They find many of the errors relevant to code inspection.
- Very effective as an aid to inspections. A supplement to, but not a replacement for, inspections.

Sample Static Analysis Checks

Fault Class	Static Analysis Check
Data	**Undeclared variables** **Variables used before initialization** **Variables declared but never used** **Possible array bounds violations**
Control	**Unreachable code**
Input/Output	**Variables output twice with no intervening assignment**
Interface	**Parameter type mismatches** **Parameter number mismatches** **Non-usage of the results of functions** **Uncalled functions**
Storage management	**Unassigned pointers**

LINT Static Analysis Example

```
138% more lint_ex.c

#include <stdio.h>
printarray (Anarray)
  int Anarray;
{
  printf("%d",Anarray);
}
main ()
{
  int Anarray[5]; int i; char c;
  printarray (Anarray, i, c);
  printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) ::
lint_ex.c(11)
printf returns value which is always ignored
```

References

- Sommerville, Ian, *Software Engineering*, 6th ed, 2000. New York: Addison Wesley.