INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007
Michele Rousseau
Set 9

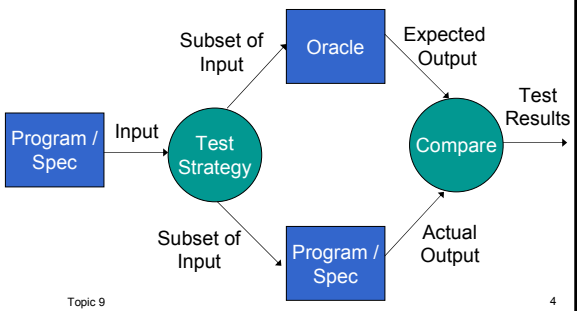
Previous Lecture

- **Testing**

Today's Lecture

- **More on Testing**
  - Static Analysis
    - Code Walkthroughs / Inspections
  - Formal Verification
  - Dynamic Testing

Typical Testing Process



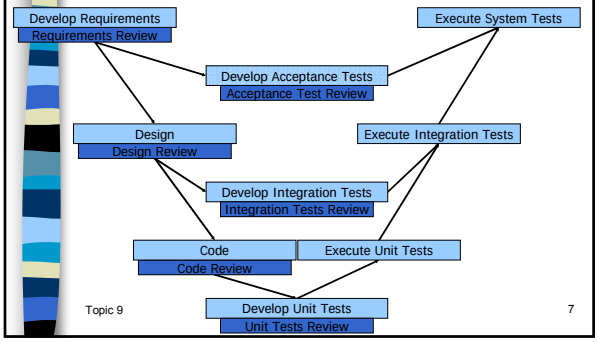
Software Testing

- **Exercising a system [component]**
  - on some predetermined input data
  - capturing the behavior and output data
  - comparing with test oracle
  - for the purposes of
    - identifying inconsistencies
    - verifying consistency between actual results and specification
      - to provide confidence in consistency with requirements an measurable qualities
      - to demonstrate subjective qualities
    - validating against user needs
- **Limitations**
  - only as good as the test data selected
  - subject to capabilities of test oracle

Remember the Diff Levels of Testing

- **System Testing**
  - Defined at Requirements -> Run after integration testing
- **Integration Testing**
  - Defined at Design -> Run after Unit Testing
- **Unit Testing**
  - Defined at Implementation -> Run after Implementation of each unit
- **Regression Testing (testing after Change)**
  - Defined throughout the process -> Run after modifcations

V-Model of Development & Testing (the big picture)



Goals of Testing

- **Reveal failures/faults/errors**
- **Locate failures/faults/errors**
- **Show system correctness**
- **Improve confidence that the system performs as specified (verification)**
- **Improve confidence that the system performs as desired (validation)**
- **Desired Qualities:**
  - Accurate
  - Complete / thorough
  - Repeatable
  - Systematic

Static Analysis

- **Examine & analyze source code**
- **Goal:**
  - Discovering anomalies and defects
- **May be used before implementation**
  - Execution is not Required
- **May be applied to any representation of the system**
  - Requirements
  - Design
  - Test data, etc…

Static Analysis

- **Very effective technique for discovering errors**
- **They reuse domain and programming knowledge**
  - reviewers are likely to have seen the types of error that commonly arise
- **Examples:**
  - Code Reviews &
  - Inspections

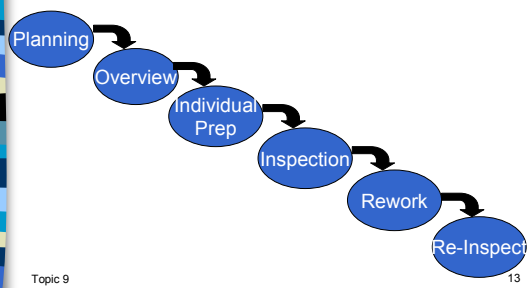
Code Reviews ("Walk-throughs")

- **Developer presents the code to a small group of colleagues**
  - Developer describes software
  - Developer describes how it works
    - "Walks through the code"
  - Free-form commentary/questioning by colleagues
- **Benefits**
  - Many eyes, many minds
  - Effective
- **Drawbacks**
  - Can lead to problems between developer and colleagues

Inspections

- **Small Team**
  - Author (Programmer)
    - Silent observer
    - Knows the code too well – might introduce bias
  - Reader
    - Presents the code
    - May have 1 or 2
  - Tester
    - Reviews the code "Testing point of view"
    - May have 1 or 2
  - Moderator
    - Conducts the inspection
    - Motivates other participants
    - Not directly involved with the product being inspected
    - Keeps the team focused and together

Inspection Process



Pre-Inspection Stages

- **Planning**
  - Select the team
  - Organize when and where
  - Ensure code and spec are complete
- **Overview**
  - Present general description of the material to be inspected
- **Individual preparation**
  - Each member inspects the code and the spec

Program Inspection

- **Should be short**
- ***Exclusively* focused on defects, anomalies, & non-compliance with standards**
- **Should not recommend changes or suggest corrections**
- **Paraphrase code → a few lines at a time**
  - Express meaning at a higher level of abstraction
- **Code is analyzed using a checklist**

Code Checklist

- **Wrong use of data**
  - Variables not initialized
  - Array index out of bounds
  - Dangling pointers
- **Faults in declaration / use of variables**
  - Duplicate use of variable names
- **Faults in computations**
  - Div by 0
  - Type mismatch of variables

Code Checklist (2)

- **Faults in relational expressions**
  - Incorrect operator use (> instead of >)
- **Faults in Control Flow**
  - Infinite loops
  - Off by 1 errors
- **Faults in Interfaces**
  - Incorrect number of parameters
  - Passing the wrong type
  - Inconsistent use of global variables

Rework & Re-inspection

- **Rework**
  - Author corrects code
- **Re-inspection**
  - Can be done by team or moderator
  - Can either check for new problems that may have arisen
  - Can verify errors were corrected

Length of Inspection

- **Can cover up to 500 statements per hour**
  - Depending on experience of team
  - Usually more like 125/hor
- **Should not go for more than 2 hours**
- **Should be done frequently**

Inspections

- **Cons:**
  - Can be too shallow
  - Programmers can be defensive
    - Evaluations of the programmer should not be determined by reviews
  - Team may have insufficient knowledge of the domain

Inspections and Testing

- **Inspections and testing are complementary and not opposing verification techniques**
- **Both should be used during the V & V process**
- **Inspections can check conformance with a specification**
  - Can't check conformance with the customer's real requirements
  - Cannot validate dynamic behaviour
- **Inspections cannot check non-functional characteristics such as performance, usability, etc.**

Tools for Static Analysis

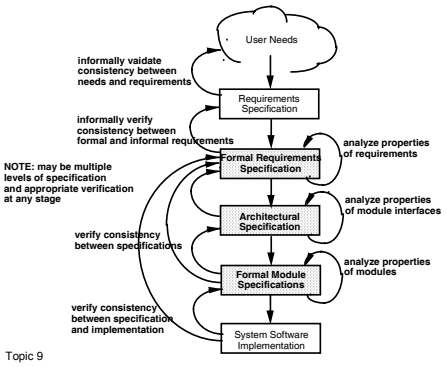
- **Scan source text & detect possible faults / anomalies**
  - Look for possible erroneous situations such as:
    - Unused variables
    - Undeclared variables
    - Unreachable code
    - Variables used before initialization
    - Parameter type mismatches
    - Parameter number mismatches
    - Uncalled functions or procedures
    - Non-usage of function results
    - Possible array bound violations
    - Misuse of pointers

Formal Verification

- **Techniques for proving consistency between two software descriptions**
  - to prove consistency of specification
  - to prove correctness of implementation

Correctness →
Correct with respect to the specification

Verification with Formal Specs



Formal Verification / Validation

- **Some shortcomings**
  - does not show other qualities
    - Performace, usability, etc..
  - May not scale up
  - only informal techniques for validating against user needs
  - subject to assumptions of proof system
  - only as good as formal specification
  - Not trivial → tedious
  - Not always cost effective
- **Generally used on a part of the system**
- **Example: Mathematically Based Verification**

Mathematically Based Verification

- **Must have formal specifications**
  - Notation must be consistent with mathematical verification techniques
- **The programming lang. must have formal semantics**
- **This is an intensive process but…**
  - Can verify correctness
- **Generally,**
  - Not cost effective for large systems

Tools for Mathematical Verification

- **Can it be automated?**
  - Theorem provers
    - Assist in developing proofs
  - Usually work with a subset of the program
  - Not completely automated

The problem with Testing

- Can't test exhaustively
  - Not feasible to run all those test cases
  - Not feasible to validate them once they are run
- Want to verify software →
- Need to test →

So,

- Need to decide on test cases →

But,

no set of test cases guarantees absence of bugs,

Testing Techniques

So,

- We need to find a **systematic approach** to selecting of test cases **that will lead to**:
  - **accurate,**
  - **acceptably thorough,**
  - **repeatable identification of errors, faults, and failures?**

Practical Issues

- **Purpose of testing**
  - Fault detection
  - High assurance of reliability
  - Performance/stress/load
  - Regression testing of new versions
- **Conflicting considerations**
  - safety, liability, risk, customer satisfaction, resources, schedule, market windows and share
- **Test Selection is a sampling technique**
  - choose a finite set from an infinite domain

Fundamental Testing Questions

- **Test Criteria: What should we test?**
- **Test Oracle: Is the test correct?**
- **Test Adequacy: How much is enough?**
- **Test Process: Is our testing effective?**

How to make the most of limited resources?

Test Criteria

- **Testing must select a subset of test cases that are likely to reveal failures**
- **Test Criteria provide the guidelines, rules, strategy by which test cases are selected**
  - actual test data
  - conditions on test data
  - requirements on test data
- **Equivalence partitioning is the typical approach**
  - a test of any value in a given class is equivalent to a test of any other value in that class
  - if a test case in a class reveals a failure, then any other test case in that class should reveal the failure
  - some approaches limit conclusions to some chosen class of errors and/or failures

Test Oracles

- **Where does "expected output" come from?**

*A **test oracle** is a mechanism for deciding whether a test case execution failed or succeeded*

- **Critical to testing**
- **Difficult to create systematically**
- **Typically done with a lot of guesswork**
  - Typically relies on humans
  - great dependence on the intuition of testers
- **Formal specifications make it possible to automate oracles**

What Does an Oracle Do?

- **Your test shows cos(0.5) = 0.8775825619**
- **You have to decide whether this answer is correct?**
- **You need an oracle**
  - Draw a triangle and measure the sides
  - Look up cosine of 0.5 in a book
  - Compute the value using Taylor series expansion
  - Check the answer with your desk calculator

Test Adequacy

- **Coverage metrics**
  - when sufficient percentage of the program structure has been exercised
- **Empirical assurance**
  - when failures/test curve flatten out
- **Error seeding**
  - percentage of seeded faults found is proportional to the percentage of real faults found
- **Independent testing**
  - faults found in common are representative of total population of faults