

INF 111 / CSE 121: Software Tools and Methods

Lecture Notes for Fall Quarter, 2007
Michele Rousseau

Lecture Notes 4 - Testing

Today's Lecture

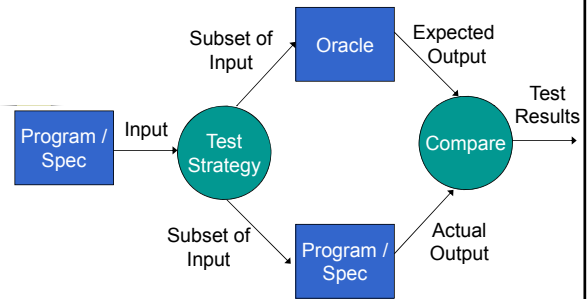
• More on Testing

- Static Analysis
 - ▣ Code Walkthroughs / Inspections
- Formal Verification
- Dynamic Testing

Previous Lecture

- Continue with XP
- No Silver Bullet
- Testing

Typical Testing Process

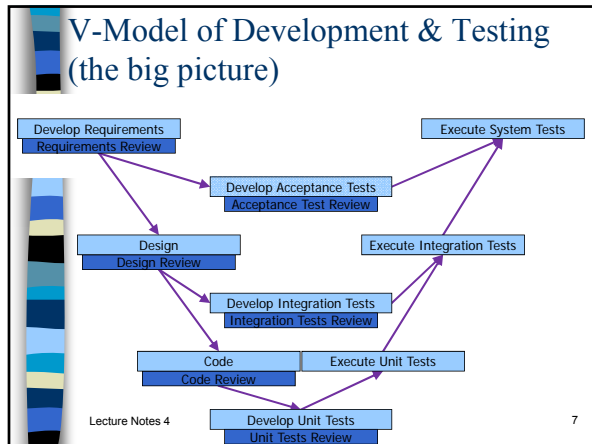


Quiz #1 Today

- Write in Pen if you want it to be regraded

Different 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 modifications



Motivation

- **People are not perfect**
 - We make errors in design and code
 - Goal of testing: given some code, uncover as many errors as possible
- **Important and expensive activity**
 - Not unusual to spend 30-40% of total project effort on testing

Lecture Notes 4 10

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 and 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

Lecture Notes 4 8

The Purpose of Testing

Design and coding are creative. but...

- **Testing is Destructive**
 - The primary goal is to “break” the software
- **Very often the same person does both coding and testing**
 - This is not ideal... why?
 - Need “split personality”:
 - ▣ when you start testing, become paranoid and malicious
 - Surprisingly hard to do: people don't like finding out that they made mistakes

Lecture Notes 4 11

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

Lecture Notes 4 9

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...

Lecture Notes 4 12

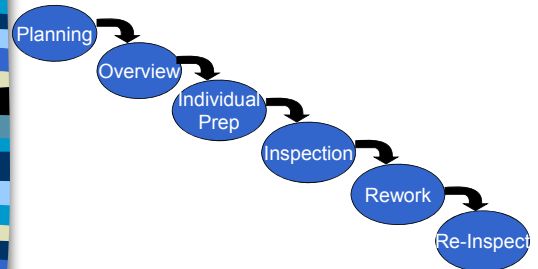
Static Analysis

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

Lecture Notes 4

13

Inspection Process



Lecture Notes 4

16

Code Reviews (“Walk-throughs”)

- o **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
- o **Benefits**
 - Many eyes, many minds
 - Effective
- o **Drawbacks**
 - Can lead to problems between developer and colleagues

Lecture Notes 4

14

Pre-Inspection Stages

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

Lecture Notes 4

17

Inspections

- o **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

Lecture Notes 4

15

Program Inspection

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

Lecture Notes 4

18

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

Lecture Notes 4

19

Length of Inspection

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

Lecture Notes 4

22

Code Checklist (2)

- **Faults in relational expressions**
 - Incorrect operator use (> instead of \geq)
- **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

Lecture Notes 4

20

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

Lecture Notes 4

23

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

Lecture Notes 4

21

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.**

Lecture Notes 4

24

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

Lecture Notes 4

25

Today's Lecture

- **More on Testing**
 - Static Analysis
 - Formal Verification
 - Coverage-Based Testing

Lecture Notes 4

28

Take a break!

- **Stretch, Relax**
 - **Get some water, Use the restroom**
 - **Get to know your classmates...**
 - **Etc.....**
- When we return...
- **No Silver Bullet**
 - **Testing**

Lecture Notes 3

26

Verification & Validation (revisited)

- **Verification**

"Are we building the product right?" (Boehm)

 - The Software should conform to its specification
 - testing, reviews, walk-throughs, inspections
 - internal consistency; consistency with previous step
- **Validation**

"Are we building the right product?"

 - The software should do what the user really requires
 - ascertaining software meets customer's intent

Lecture Notes 4

29

Before the Break

- **Testing**
 - Static Analysis
 - Code Walkthroughs
 - Inspections

Lecture Notes 4

27

Quality Assurance : 5 Problems

- #1 : Eliciting the Customer's Intent**
 - Getting the Specs to meet the "real needs"
- #2 : QA is inherently difficult**
 - Systems can be complex making QA difficult to perform
 - Air Traffic Control → stringent performance
 - Medical Diagnosis System → Complex processing

Lecture Notes 4

30

Quality Assurance : 5 Problems

#3 : Management Aspects

- Who does what testing?
 - Are developers involved?
- How are bugs handled?
- What is the reward structure?

#4 : QA Team vs. Developers

- QA lays out the rules
- Uncovers faults
 - "image of competition"
- Viewed by Developers as Cumbersome
 - "let me just code"

#5 : Can't test exhaustively

Lecture Notes 4

31

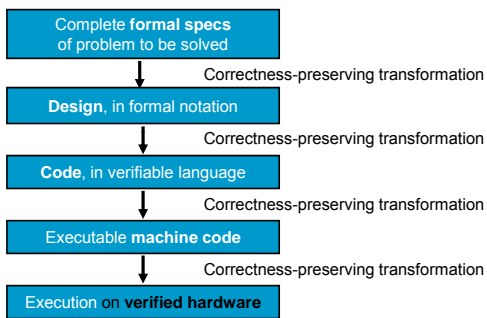
Unit Tests

- **Developer tests the code just produced**
 - Needs to ensure that the code functions properly before releasing it to the other developers
- **Benefits**
 - Knows the code best
 - Has easy access to the code
- **Drawbacks**
 - Bias
 - "I trust my code"
 - "I always write correct code"
 - Blind spots
- **Possible Solutions:**
 - Outside Testers
 - Walkthroughs / Inspections

Lecture Notes 4

34

How QA would like the world to be



Lecture Notes 4

32

Formal Verification

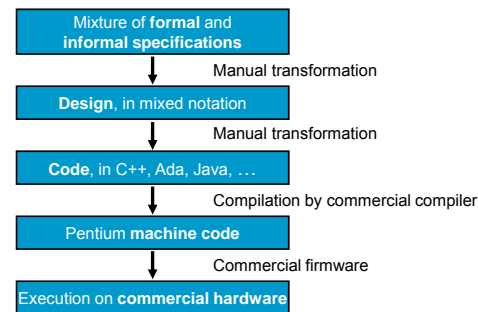
- **Techniques for proving consistency between two software descriptions**
 - to prove consistency of specification
 - to prove correctness of implementation

Correctness →
Correct with respect to the specification

Lecture Notes 4

35

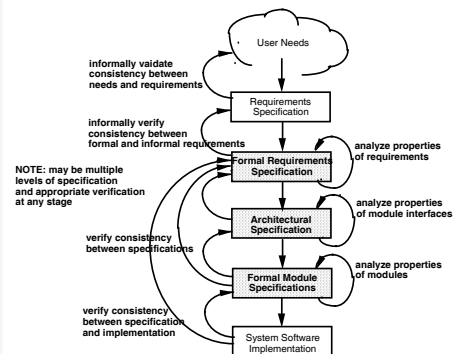
... but in reality



Lecture Notes 4

33

Verification with Formal Specs



Lecture Notes 4

36

Formal Verification / Validation

- **Some shortcomings**
 - does not show other qualities
 - Performance, 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**

Lecture Notes 4

37

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,**

Lecture Notes 4

40

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

Lecture Notes 4

38

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?**

Lecture Notes 4

41

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

Lecture Notes 4

39

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**

Lecture Notes 4

choose a finite set from an infinite domain

42

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?

Lecture Notes 4

43

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

Lecture Notes 4

46

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

Lecture Notes 4

44

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

Lecture Notes 4

47

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**

Lecture Notes 4

45