

# Testing

CSE 308: Software Engineering

# Where We've Been

- [ Requirements elicitation
- [ Analysis
- [ System design
- [ Object design
- [ Implementation

# Where We're Going

- [ Testing
- [ Verification and Validation
- [ Rationale Management

**"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."**

**– Maurice Wilkes, designer of EDSAC, on programming, 1949**

**"A debugged program is one for which you have not yet found the conditions that make it fail."**

**– Jerry Ogdin**

# Testing

- [ The process of finding differences between expected and observed behavior
- [ Unit testing: object design model and components
- [ Structural testing: system design model and subsystems
- [ Functional testing: use case model and system
- [ Performance testing: nonfunctional requirements and system performance

# Testing Philosophy

- [ Testing is an attempt to show that the implementation is inconsistent with the system models
- [ Goal is to design tests that reveal problems
- [ Testing is aimed at breaking the system

# Testing Overview

- [ Reliability – measure of how well a system's observed behavior conforms to its specification
  - software reliability: probability that a software system will not cause system failure for a specified time period
- [ Failure: when observed behavior deviates from specification

# More Terminology

- [ Erroneous state (error) — situation where any further processing will lead to a failure
- [ Fault (“defect”) — a mechanical or algorithmic cause of an erroneous state
- [ Testing — systematic attempt to find faults in a planned way
  - this is NOT the same as ensuring that faults are not present!

# Reliability Techniques

- [ Fault avoidance — tries to prevent the insertion of faults into the system before it is released
- [ Fault detection — experiments used to identify erroneous states and underlying faults before system release
- [ Fault tolerance — deals with system failures by recovering from them at runtime
  - ex. modular redundancy

# Fault Detection Techniques

- [ Review — manual inspection of all aspects of a system without actually executing the system
  - walkthrough: developer presents API to reviewers
  - inspection: review team compares code to requirements
- [ Code reviews are very (85%) effective at detecting faults

# Debugging

- [ Debugging assumes that faults can be found by starting from an unplanned failure
- [ Correctness debugging: find any deviation between observed and specified functional requirements
- [ Performance debugging: find any deviation between observed and specified nonfunctional requirements

# Testing

- [ Testing tries to create failures or erroneous states in a planned way
  - a test is successful if it identifies faults
- [ A good test model contains test cases that identify faults
  - tests should include a range of input values, including invalid inputs and boundary cases

# Testing Activities

- [ Test planning: allocates resources, schedules the testing
- [ Usability testing: finds faults in the user interface design
- [ Unit testing: finds faults in participating objects/subsystems
- [ Integration testing: finds faults when testing components together (incl. structural testing)
- [ System testing: finds faults from scenarios and requirements

# Testing Concepts

- [ Component: system part that can be isolated for testing
- [ Fault: design/coding mistake
- [ Erroneous state: manifestation of a fault
- [ Failure: deviation between specification and actual behavior
- [ Test case: set of inputs and expected results
- [ Correction: change to a component

# Test Cases

- [ Set of input data and expected results
- [ A test case has five attributes: name, location, input, oracle (expected results), and log (actual results)
- [ Blackbox tests focus on I/O behavior of the component
- [ Whitebox tests focus on internal structure of the component

# Test Stubs and Drivers

- [ Used to substitute for missing parts of the system
- [ Test driver simulates the part of the system that calls the component being tested
- [ Test stub simulates components called by the tested component

# Corrections

- [ Correction: a change to a component for the purpose of repairing a fault
- [ Problem tracking: documentation of failure and response
- [ Regression testing: reexecution of all prior tests
- [ Rationale maintenance: documents rationale of change, including assumptions used when building the component

# Testing Activities

# Testing Activities

- [ Component inspection
- [ Usability testing
- [ Unit testing
- [ Integration testing
- [ System testing

# Component Testing

- [ Review component source code in a formal meeting

- [ Fagan inspection steps:

  - Overview

  - Preparation

  - Inspection meeting

  - Rework

  - Follow-up

# Fagan Inspections

- [ Preparation: reviewers become familiar with source code
  - they don't focus on finding faults yet
- [ Inspection: reader describes the source code, line by line
  - reviewers raise issues if they think there's a fault
  - solutions are not explored at this time!

# Active Design Reviews

- [ Fagan inspections are seen as time-consuming
- [ Parnas proposed active design reviews as an alternative
- [ Active design reviews eliminate the inspection meeting
  - reviewers fill out questionnaires
  - code author meets individually with reviewers
- [ Both inspection types are more effective than testing

# Usability Testing

- [ Tests the user's understanding of the system
  - finds difference between system and users' expectations
  - works by studying user interactions with the system
- [ Developers formulate a set of test objectives, observe participants, and collect data measuring user performance

# Types of Usability Tests

## — [ Scenario tests

- scenario tests use paper mock-ups or prototypes

## — [ Prototype tests

- end users are presented with a partial implementation
- vertical prototype: completely implements a use case
- horizontal prototype: implements a single layer
- Wizard of Oz: human operates system behind the scenes

# Types of Usability Tests

## — [ Product test

- uses functional version of system instead of prototype
- can only be conducted after most of the system has been developed
- requires system to be easily modifiable (to adjust it based on user feedback/test results)

# Elements of Usability Testing

- [ Development of test objectives
- [ Representative sample of end users
- [ Actual or simulated work environment
- [ Controlled, extensive interrogation of users
- [ Collection and analysis of results
- [ Recommendations of how to improve the system

# Unit Testing

- [ Focuses on building blocks of the software system

- [ Motivations:

- reduces complexity of test activities
- makes it easier to pinpoint and correct faults
- allows parallelism in the testing process

# Unit Testing Techniques

— [ The four most important techniques for unit testing are:

— equivalence testing

— boundary testing

— path testing

— state-based testing

# Unit Testing Techniques

- [ Equivalence testing

- blackbox testing, minimizes the number of test cases
- possible inputs are partitioned into equivalence classes
- equivalence classes are determined based on coverage, disjointness, and representation
- for each equivalence class, we select a typical input and an invalid input

# Unit Testing Techniques

- [ Boundary testing

- focuses on the conditions at the boundary of the equivalence classes
- selects inputs from the “edges” of the classes
- Does not explore combinations of test data

# Unit Testing Techniques

- [ Path testing

- whitebox testing technique
- exercises all possible paths through the source code
- starts with a flow graph
  - nodes represent executable blocks
  - associations represent flow of control
- test cases are designed to traverse each transition

# Unit Testing Techniques

- [ State-based testing

- derives test cases from UML statechart diagram
- each test must include a sequence to put the class in the desired state before testing the transitions

# Integration Testing

- [ Tests faults by focusing on small groups of components
- [ Tests two or more components together
  - if no faults are detected, more components are added
- [ Different strategies can be employed to reduce testing overhead

# Integration Testing Strategies

- [ Big bang testing

- all components are tested together as a single system

- [ Bottom-up testing

- tests each component individually, then with components of the next layer up

# Integration Testing Strategies

- [ Top-down testing

- tests top layer components first, then moves down

- requires many test stubs

- [ Sandwich testing

- combines top-down and bottom-up testing strategies

# System Testing

- [ Ensures that the complete system complies with the functional and nonfunctional requirements
- [ Includes functional testing, performance testing, pilot testing, acceptance testing, and installation (beta) testing

# System Testing Activities

- [ Functional testing (requirements testing)

- finds differences between use case model and the system

- [ Performance testing

- finds differences between design goals and the system

- includes stress testing, volume testing, security testing, timing testing, and recovery tests

# System Testing Activities

## — [ Pilot testing

- system is installed/used by a small set of users
- includes alpha and beta tests

## — [ Acceptance testing

- client evaluates the system
- includes benchmark tests, competitor testing, and shadow testing

# Managing Testing

# Documenting Testing

- [ Test plan: documents scope, approach, resources, and schedule
- [ Test case specification: documents each test
- [ Test incident report: documents test execution results
- [ Test report summary: lists all failures discovered during testing

# Next Time

— [ System Validation and Formal Specification