Announcements

- Design document due Today
- Last day to discuss design document is Friday, February 10
- Sprint 1 begins next Monday, February 13
  - Planning document also due then
Lecture 12

- Desk Inspection
- Software Testing
Desk inspections

- Usually done by a team member
- Someone at the same ability level as the coder
- Done on a single module
- Can be done daily
- Usually 10 to 20 minutes
Benefits

- Reduce both omission and commission defects
- Provide an easy method for keeping team members up to date on project effort
- Provide an early check on software quality
- Good way to provide peer training of new team members
Members

- Only one role in desk inspection: the reviewer
- Coder has no involvement
Software Testing

- What is testing?
- When should we test?
- Functional testing
- Structural testing
Is it the process we use for fixing faults (bugs)?

Is it the means by which we confirm that the system is working correctly?

Is it a demonstration that errors are not present?
Is it the process which we use to search for defects?
Is it the activity of preventing faults?
Is it something else?
Purpose of testing

- Boris Beizer classifies our attitude on the purpose of testing into 5 phases
Phase 0

- There is no difference between testing and debugging
  - Other than in support of debugging, testing has no purpose
- Student and novice approach
Phase 1

- The purpose of testing is to show that the software works
  - This approach may not find very many faults
  - This approach is used by developers during acceptance tests
- Favored by sales and marketing
Phase 2

- The purpose of testing is to show that the software doesn’t work
  - Tends to cause tension between developers and verifiers
- Favored by the **customer** during acceptance tests
- Favored by the **legal** department
Phase 3

- The purpose of testing is not to prove anything, but to reduce the perceived risk of not working to an acceptable value.
- Requires an understanding of the expected operational environment.
Testing is not an art. It is a mental discipline that results in low-risk software without much testing effort

- This is the goal of quality driven software development teams
- Testing is a continuous process which starts at the time the product is conceived and only ends after product retirement
- Favored by development teams with adequate time to really test a product
Testing paradox

- **Software development engineers**
  - Do their very best to produce quality products on time and within budget

- **Test engineer**
  - You must be unyielding in your pursuit of faults in the product. Time and budget should not be limiting factors
What is testing?

- It is a collection of guidelines, methodologies, and tools for examining our products for inconsistencies in stated requirements, implied requirements, and missing requirements.

- It is one of the processes that must be used to ensure we are producing high quality products.
Observation

- It is everyone’s job
  - Customers and developers both have responsibilities
- It starts with the conception of the product
- It is an important part of every development phase
- It ends (maybe) with the retirement of the product
What about debugging?

- It is the activities associated with diagnosing the precise nature of a known fault and the removal of the fault.

- Testing uncovers faults, debugging attempts to remove faults.
What kinds of faults?

- **Omission**
  - Requirements we forgot to include
    - They were implied
    - Never thought about

- **Commission**
  - Requirements we failed to implement correctly
    - Stated requirement
    - Implied requirement
    - Non-requirement
Testing process

- Should be **blind and blameless**
- Not very important to know who inserted it, how it was inserted, when it was inserted, or why the fault occurred
  - Record it, though. Can be used to improve the software development process
- What is important: early detection of faults, prevention of faults from being shipped to the customer
Test classification

- **Functional**
  - Correctness
  - Performance
  - Stress (Reliability)

- **Structural**
  - Parts and statement coverage
  - Decision coverage
  - Path coverage
Functional testing

- Black box approach
- Concerned with system functionality and features
- Not concerned with implementation details
- Takes the view point of the user
Correctness testing

- To be sure that the system meets all of its stated and implied requirements
Performance testing

- To determine how well the system meets all of its stated and implied requirements
- Also determines the resource limitations and inefficiencies of the system
  - Include equipment, time, environment, data, algorithms, etc
Observation

- Functional testing is theoretically capable of detecting all faults; but may take infinite time and dollars to accomplish
Stress testing

- To uncover any hidden faults introduced by commission which may cause the system to be unreliable
- Also searches for faults of omission which may yield unsafe and unreliable operation
  - “Scenario testing”
Structural testing

- White box approach
- Concerned with implementation details
- Takes the viewpoint of the developer
- Tends to find errors of commission, not omission
Parts and statement coverage

- To be sure that all parts (hardware) and all statements (software) are used or executed at least once
- If you can’t use it (hardware) or execute it (software), then why did you include it?
- May be a good reason – is it part of a reuse package and too expensive to remove?
Decision coverage

- Check to be sure all decision points in the product are exercised at least once for each possible outcome
Path coverage

- Exercise the system in such a way that all possible paths through the decision points, both hardware and software, are used at least once
Observations

- Parts and statement coverage is a weak form of testing
- Decision testing is better but does not cover all possible paths
- 100% path testing can take forever
- 100% path coverage at source code level does not guarantee 100% path coverage at object code level
Structural tests cannot find all the faults even if all the possible paths are checked.

Structural tests assure us that what we produced is correct, but not that it is the correct product.
There are some things you might not want to test for, or that you are unable to test.

- It would be foolish to test to see if every nuclear device or car air bag worked in the factory prior to delivery to the customer.
Bad news

- Testing assumes the following (suspect) components:
  - Fault free input information
  - Fault free response information
  - Fault free observations
    - Uncertainty principle?
  - Fault free testing environment
    - Who is checking on the checker?
What should we do?

- If testing in the real world will never assure us (100%) that there are no more faults, then our goal should be to provide enough (risk based) testing to ensure ourselves that the probability of failure of the system is low enough to be acceptable to users.
Funny fact of life

- Why is it, when a bug is killed, that all of its relatives come to the funeral
Software testing techniques

- Path testing
- Transaction flow testing
- Data-flow testing
- Domain (random) testing
- Syntax testing
- State testing
- Mutation testing
Path testing

- Based on selecting paths through the software
  - Statement coverage
  - Branch coverage
  - Some % of full path coverage

- Good for module level testing

- Built-in self testing code can make it difficult to use at the system integration level and above
Transaction flow testing

- Traces the objects from their birth to their death, and possibly beyond, through the system
- Forms the basis of many code reviews and walkthroughs
- Hard part is finding the transaction flow
Data flow testing

- Based on tracing data through the system
- Very useful on multiprocessor and parallel programming systems
- Looks for initialization, and data flow anomalies
- In some sense, testing for possible hardware and/or compiler incompatibilities
Domain (random) testing

- Requires domain knowledge of how the system is to be used
- Input data is either pre-selected or drawn at random
- System response is checked for anomalous conditions
Requires a large number of tests (N) to reach a confidence level of $1 - e$, where $e$ is very small.

$$(1 - p)^N = e$$

$p$ is the failure rate.
\( p = 10^{-6} \)

<table>
<thead>
<tr>
<th>( N )</th>
<th>( e )</th>
</tr>
</thead>
<tbody>
<tr>
<td>500,000</td>
<td>0.606</td>
</tr>
<tr>
<td>1,000,000</td>
<td>0.368</td>
</tr>
<tr>
<td>2,000,000</td>
<td>0.135</td>
</tr>
<tr>
<td>4,000,000</td>
<td>0.0183</td>
</tr>
<tr>
<td>8,000,000</td>
<td>0.000335</td>
</tr>
</tbody>
</table>
Syntax testing

- GIGO - Garbage In, Garbage Out
- Used to test the program and data translation functions of the system

- Do you really think the compiler you are using is bug free?
State testing

- Based on finite state or petri net models
- Trace state transition behavior of the system
- Very useful for testing real time systems
Mutation testing

- Premise: test data sets are good if they can detect small random changes to the software
- Plant errors in the code and check to see if they are detected
- Good for build regression tests
  - Helps find “mutants” generated during maintenance
- Requires significant resources
Questions?