Testing Objectives

- Informal view:
  - Testing: a process of executing software with the intent of finding errors
  - Good testing: a high probability of finding as-yet-undiscovered errors
  - Successful testing: discovers previously unknown errors
- Formal view
  - Testing is an experiment
  - Hypothesis: there are no faults
  - Independent variables: context and input
  - Dependent variables: output of test
  - Do the experiment: execute the model (program/system)
  - Analysis: are the outputs those predicted by the theory (requirements / logical structure of program)

Basic Definitions

- ➲ Test case: specifies
  - ⇘ Inputs + pre-test state of the software
  - ⇘ Expected results (outputs + new-state)
- ➲ White-box testing: uses knowledge of the internal structure of the software
  - ⇘ E.g., write tests to "cover" internal paths
  - ⇘ Typically used for unit testing
- ➲ Black-box testing: ignores the internal logic of the software, and looks at what happens at the interface (e.g., given this input, was the produced output correct?)
  - ⇘ Typically used for system testing

Testing Phases

- Unit testing
  - Initial testing on a developers component
- Integration testing
  - Testing of incrementally composed components
- System testing
  - Testing of a fully integrated system
  - Typically two phases: system and stress testing
- Alpha testing
  - Small set of friendly users – live context use
- Beta testing
  - Larger set of not necessarily friendly users – live context use
- Regression testing
  - Re-testing at unit, integration and system test levels to ensure evolution has not broken non-changed parts

Unit Testing

- Scope: one component from the design
  - Often corresponds to the notion of "compilation unit" from the programming language
- Responsibility of the developer
  - Not the job of an independent testing group
- Both white-box and black-box techniques are used for unit testing
- Maybe necessary to create stubs and drivers:
  - If related modules are not yet implemented or not yet tested

Stubs

- It may be difficult to test a method or class that interacts with other methods or classes
- The replacement of a method that has not yet been implemented or tested is called a stub
- A stub has the same header as the method it replaces,
  - but its body only displays a message indicating that the stub was called or
  - it performs some other hard coded action that allows you to proceed.

Drivers

- A driver program (aka harness)
  - declares any necessary object instances and variables,
  - assigns values to any of the method's inputs,
  - calls the method, and
  - displays the values of any outputs returned by the method
- You can put a main method in a class to serve as the test driver for that class's methods

Basic Strategy for Unit Testing

- Evaluate the tests using white-box techniques (test adequacy criteria)
  - How well did the tests cover statements, branches, paths, etc.?
  - Many possible criteria; at the very least need 100% branch coverage
- Create more tests for the inadequacies: e.g., to increase coverage of nested loops
- Create black-box tests
  - Based on the specification of the unit (as determined during design)
  - E.g. method interface, + preconditions

Integration Testing – Approach

- Integration testing: scope = set of interacting components
  - 2 general strategies: top-down and bottom-up
  - Focus: correctness of component interactions
  - Mixture of black-box and white-box techniques
- Goals
  - Ensure component expectations are met
    - Interfaces used match
    - Interfaces provided
  - Eliminate unwanted component interactions
    - Shared variables, race conditions, pointer problems, etc.
  - Replace "unit reality" with "integration reality"
    - Stubs at best "model" reality
- Infuse (change management + integration testing)
  - Systematic management of multiple developers making changes to a system
  - Add in integration testing for the recombination phase

System Testing

- Goal: find whether the system does what the customer expects to see
  - Black-box techniques
- In the spec created during requirements analysis, there should be validation criteria
  - How are the developers and the customers going to agree that the software is good enough?
- Many issues: functionality, performance, documentation, usability, portability, etc.

System Testing (cont)

- Initial part of system testing is done by the software producer
- Eventually, we need testing done by the customers (or surrogates)
  - Every time a customer runs the software he/she is testing it
  - Customers are good at doing unexpected things, which is great for testing
- If the software is built for a single customer: series of acceptance tests
  - Deploy the software in the customer environment and have end-users run it

System Testing (cont)

- If the software is produced for multiple customers: two phases
- Alpha testing: conducted at the vendor's site by a few customers
  - The vendor records any errors and usage problems
- Beta testing: the software is distributed to many end-users; they run it in their own environment and report problems
  - Often done by thousands of users

Stress Testing

- ➲ Form of system testing: check the behavior of the system under very heavy load conditions
- ➲ E.g., what if we have data sets that are an order of magnitude larger than normal?
  - ➥ Will we run out of memory?
  - ➥ Will the OS start writing memory pages to disk (thrashing)?
- ➲ E.g., what if our server gets 10 times more client requests than usual?
  - ➥ Will the system slow to a crawl ? Denial of service attacks ?

Stress Testing (cont)

- Goal: find how well the system can cope with defined load and overload
- Reason 1: determine failure behavior
  - If load goes above the intended (which often is a possibility) how gracefully does the system fail?
- Reason 2: expose bugs that only occur under heavy loads
  - Especially for system SW, middleware, servers, etc.
  - E.g., memory leaks, incorrect resource allocation and scheduling, race conditions

Regression Testing

- Basic idea: rerun old tests to make sure that nothing was "broken" by a change
  - Changes: bug fixes, module integration, maintenance enhancements, etc.
- To be able to do this regularly and efficiently, we need test automation tools
  - Load tests, execute them, check correctness
  - Everything has to be completely automatic
  - Test case database is required
- Could happen at any time: during initial development or after deployment