Session 8: Testing
Special importance of testing
Stages of testing
Manual vs. automated testing
Continuous testing strategies

Why is testing critical to project management?
- It's when many projects first fall behind schedule and go over budget
  - Everything looks good until we get into heavy late-stage testing, which reveals unexpected bugs.
  - Results may require substantial rework, thus adding many unplanned tasks to the plan!
  - In extreme case we may lose control of the project!
- An I.T. project manager really needs to understand all this.

What's a bug?

Why doesn't our SDLC have a testing phase?

What's wrong with this?
The programming is done. Now we're ready to begin testing.

Programming
- Programming = internal component design + coding + testing
- A single task on the project plan
  - Develop (or program) module glob33 means
    1. Design glob33
    2. Code & unit test glob33
    3. Deliver thoroughly tested glob33 for integration into the application system

Programming
Stages of software testing

For all major software development:
- Unit test
- Integration test
- System test
- Volume (stress) test
- Acceptance test

For a software product also (optional)
- Alpha test
- Beta test

When does each occur?  
Who does it?

Unit testing

- Validates a single component (MUT: module under test)
  - a non-trivial subroutine or function
  - an object-oriented class
  - other (anything worth assigning and keeping track of)

- It is inseparable from coding
  - It's (almost) always done by the same programmer
  - as part of a single task "Develop xyz"

- Automated vs. manual unit testing

Manual unit testing

- In a manual test, the programmer
  - launches execution
  - enters test-case data
  - examines results
    - if unexpected, make corrections and redo

- How do we assure adequate test coverage?

- How do we know a future change won't invalidate a previous test result?

Automated unit testing

- The programmer
  - writes code to invoke the MUT with specified inputs
  - writes code to receive the result and compare it with the expected value
  - launches execution which will:
    - Do nothing (or just report OK) if result was expected.
    - Issue alarm if not
  - reruns the same tests every time anything is changed.

- A related fad is TDD (test-driven development)

How does that work?
Bottom-up unit testing
- The programmer writes a special **test driver** program to invoke the MUT
  - Testing can be extremely thorough, since the driver can invoke the MUT with a wider range of inputs than the eventual application program will.
  - There’s no place in the eventual application program for the driver, so retesting the MUT may require special set-up. **Don't throw the driver away!**
- The MUT may invoke lower-level modules, which presumably have already undergone thorough test.

Top-Down testing
- Combines unit testing with integration testing.
- The MUT is invoked by the actual real higher-level module in the application, which has already been tested.
- The programmer codes dummy modules to simulate lower-level modules that the MUT needs to invoke. (They can be thrown away later.)
- Testing of the interfaces is more certain than with bottom-up unit testing, but range of test cases will be narrower.

Integration testing
- Validates the **interfaces** among the components in the SUT (system under test)
- May be continuous or ad hoc

Continuous integration testing
- Not really continuous but **recurring** on a regular schedule (usually overnight)
- A complete "build" of the current application
  - Dozens, maybe hundreds of compilations
  - May take several hours and use huge resources
  - Experiments with parallel tasks
  - Driven by Make file or equivalent control
- A single faulty module can invalidate a test!
  - How can we keep that from happening often?
  - How should we react when it happens?
  - **Advantages & disadvantages?**
**Automated integration testing**
- Just as we could automate unit tests by building test cases into the driver program, we can automate integration test by simulating user transaction entry.
- Following each complete system build, we rerun the tests and get a failure report.
- Some organizations do this every night! (That generates a lot of pressure on the programming staff.)

**Volume (or stress) test**
- Tests the limits of
  - transaction volume
  - number of simultaneous users
  - database size
  - bizarre combinations of data entry, including the most diabolical user data-entry errors anyone can think of.
- Tests **performance** as well as correctness
  - Transaction response times
  - Start-up & initialization and shut down
- This is the time for **professional testers**
  - sometimes (misleadingly) called "Q.A."

**What next?**
- If the project's end product is an application system for a sponsoring end user, proceed to **system test** and **acceptance test**.
- If the project's end product is a software product to be marketed, consider **alpha** and **beta tests**.

**System testing**
- For the first time tests all components of the new application system together
- Includes the **people** and **user documentation** components
  - Actual user personnel
  - No I.T. hand-holding or coaching
- Includes all processing cycles, such as
  - end-of-month accounting close
  - disaster recovery procedures
Preparing for the system test and the acceptance test

- The project plan must contain detailed tasks to assure that
  - Participating user personnel get appropriate training
  - All participants know exactly what they're going to do. **Never improvise during a test!**
- The systems analyst can play the major role in preparing the user organization
  - That gives the analyst something important to do after defining the requirements (ESD)
  - The analyst usually has the necessary knowledge to take charge of this area.

Acceptance testing

- May occur as the final stage of system test
- Does the system do what the specifications (ESD) call for?
  - If so, we're ready for installation and start-up
  - If not, we have a list of urgent corrections to be made, and (at least) the acceptance testing must be repeated.
  - Resist temptation to take shortcuts here!

Result of acceptance test

a. No problems exposed; users are 100% satisfied with the application system.
   - Proceed to installation & start-up

b. Trivial bugs exposed; user-management willing to "live with" them until corrected through routine maintenance.
   - Proceed to installation & start-up

c. Serious bugs or performance problems exposed.
   - Reschedule installation after thorough re-test

Proceeding to installation

- Installing a new application system usually involves running some one-time programs for:
  - Converting data from old system files to the new system's files or database
  - Initializing certain data.
- Of course, those one-time programs must themselves have been thoroughly tested.
  - Surprisingly, many serious failures have occurred because of unexpected problems encountered during installation!
Alpha and Beta testing

When a software product is going to be released to the market we can gain more confidence by first:
  - releasing it to internal users within our company (alpha test). Once that goes well, then
  - releasing it to friendly volunteer outside user who agree to report problems but promise not to complain or sue or write reviews (beta test).

No matter how thorough we thought the system and volume tests were, actual use by real people usually turns up problems.