Testing Strategies

Sources:
Code Complete, 2nd Ed., Steve McConnell
Software Engineering, 5th Ed., Roger Pressman
• System Engineering
  – The software you're building may be only one part of a much larger system containing many hardware and software components
  – System Engineering deals with engineering the entire system, not just the software components, and goes beyond the scope of Software Engineering
Unit Testing

- Testing of individual software "units" (i.e., classes, routines)
- White box testing
- Usually done by the engineer who wrote the unit
Unit Testing: Test-Driven Development

- The traditional approach is to write unit test cases after the code is written

- Test-Driven Development is a relatively new approach that encourages writing unit test cases before the code is written
  - Writing test cases first forces you to think about the requirements and design before writing the code, which leads to a better design
  - Since your test cases will fail at first, they help you know when the code is done (i.e., when the tests work, you're done)
  - Writing test cases before the code doesn't take any more effort than writing them after the code
  - Writing test cases first will help you detect and remove bugs sooner
Unit Testing: Using stubs and drivers to isolate the class under unit test

Test cases

Driver

Results

Class to Test

Stub

Stub

Stub
Unit Testing: What do stubs do?

- Do nothing
- Validate the method inputs
- Validate method call sequence (including parameter values)
- Send a message to a log
- Return a hard-coded answer regardless of the input
- Select an answer from a pool of hard-coded answers
  - Cycle through the pool or randomly select one
- Randomly generate an answer
- Prompt the user for the answer
- Simple implementation of the module that is slower, less accurate, or somehow less capable than the real module
- Pause for awhile to simulate the time taken by the real module
- Generate errors (e.g., throw exceptions) that are hard to produce for real
Unit Testing: What do drivers do?

- Invokes the class with fixed inputs
  - If an oracle is available, inputs can be generated rather than fixed
- Compares actual outputs with expected outputs
- Records failure if expected and actual outputs don’t match
- Normally continues to execute even if a test case fails
- Generates report detailing what worked and what didn’t

- Drivers and stubs must be designed together
- Since stubs don’t produce “real” outputs, the expected results in the driver must take into account the “fake” behavior of the stubs
Integration Testing

• Integration involves combining the individual software units into larger functional units

• If the units work individually, why wouldn't they work when you put them together?
  – Ask the 2010 Miami Heat

• Example: Interactions between units may be flawed
  – Mars Orbiter disaster caused by one module assuming English units and another assuming Metric units
Integration Testing

• Big-Bang integration
  – Entire system is integrated at once
  – The system doesn't work, and you don't know why

• Incremental integration
  – Add a piece, retest the system, Add a piece, retest the system, …
  – If it breaks, you know what caused the problem (i.e., the last piece you added)

• Integration Testing is testing that's done during integration to ensure that the system continues to work each time a new piece is added
Top-Down Integration Testing

Top module is tested with stubs

Stubs are replaced one at a time, either breadth-first or depth-first

Tests are run each time new modules are integrated
Bottom-Up Integration Testing

Modules are integrated in a bottom-up fashion until the entire system has been assembled.

Tests are run each time new modules are integrated.

Drivers must be developed, but stubs are not needed.
Sandwich Integration

- Combination of top-down and bottom-up integration

- Integrate from the top and from the bottom as it makes sense, and meet somewhere in the middle
Continuous Integration

- Some projects wait several weeks between integrations
- Between integrations, engineers work on their pieces in relative isolation
- Each integration produces a new "build"
- The system may not build successfully between integrations

- A better approach for many projects is "continuous integration"
- A minimal working system is checked into source control
- Engineers check in new code every time they finish a meaningful unit of work
- Engineers must never break the build, and are responsible for adding new test cases to exercise the new code
- The system is built and tests are run every night to ensure that all is well
- The system is always integrated and build-able
Unit Testing or Integration Testing?

- Sometimes people who claim to be doing unit testing are actually doing bottom-up integration testing

- How do you tell the difference?
  - Unit testing uses stubs to isolate the module under test from its dependencies
  - Bottom-up integration testing does not (i.e., modules call their “real” dependencies)
System Testing

- Black box testing to ensure that the product meets all of the specified requirements

- Performed by independent testing group
  - The testing organization should be separate from the development organization to avoid conflict of interest
  - The testing organization creates a “test plan” that details how the product will be tested

- Development delivers periodic builds of the complete system

- Acceptance test is run on new builds to verify that they're stable enough to test
  - If a build fails the acceptance test, it's rejected and no testing resources are expended on it
  - If a build passes the acceptance test, tests are run and bugs are entered into the bug tracking system
System Testing

• Before shipping a product it's important to get feedback from real customers

• Alpha Testing
  – Product not yet feature-complete (some things are missing)
  – Give the product to trusted, enthusiastic customers for evaluation
  – Could be done at your place or theirs
  – Lots of hand-holding by the developers
  – Incentives to alpha testers could include influence on product features or early access to needed features

• Beta Testing
  – Product much closer to shipment than with alpha testing
  – Product is feature-complete (debugging and tuning still in progress)
  – Broader distribution to a less selective group of customers
  – Less hand-holding
  – Incentives could include cash, free software or customer support, early access to needed features
System Testing

• When are we done testing?

• "You're never done testing, the burden simply shifts from you to your customer."

• "You're done testing when you run out of time or you run out of money."

• You're done testing when:
  – All "show stopper" bugs have been fixed
  – The rate at which new bugs are being found falls below some acceptable threshold
    • If your testing is thorough, the rate at which you're finding new bugs is correlated to how many bugs still remain
System Testing

• Can we predict when we'll be done testing in advance?
  – An estimation and scheduling problem

• How about this?
  – Record the rate at which you're finding bugs each week
  – Once you've got several data points, fit a curve to the data that can be used to predict the date at which the bug rate will be low enough to ship
  – This might be better than guessing
Regression Testing (I)

- Any change to a software product, even a slight one, has the potential to cause bugs anywhere in the system.

- Basically, you have to assume that anything can break at any time.

- Regression testing is done on every build to ensure that new code (including bug fixes) did not break features that used to work.

- It's not feasible to re-run all prior tests on every build (especially manual tests).

- The regression test suite is a subset of tests that covers all product areas and can feasibly be run on every build.

- The regression test suite will evolve (i.e., grow larger) over time.
Regression Testing (II)

• In addition to having regression tests that cover all functional areas of the product, you should also have a regression test for every bug that has been fixed.

• Whenever a bug is discovered, if you don't already have one, design a test case that can be used later to verify that the bug has been fixed.

• After the bug is fixed, re-run the test case on each new build to ensure that the bug stays fixed.

• Experience has shown that fixed bugs often get “unfixed” later.
• Why?
  – Source control mistakes (i.e., somebody unintentionally overwrites the bug fix in the code repository).
  – The bug fix was “fragile” (i.e., it barely worked, and some later change pushes it over the edge).
  – The feature in which the bug was found is later redesigned and re-implemented. The original mistake is repeated, thus reintroducing the bug.
Customer Acceptance Testing

- After system testing is complete, the customer might perform a "customer acceptance test" before signing off on the product.

- The customer acceptance test is a suite of tests that will be run by the customer (or someone they hire) to ensure that the product meets requirements.