"The first 90% of the code accounts for the first 90% of the development time. The remaining 10% of the code accounts for the other 90% of the development time."
– Tom Cargill, Bell Labs

"If debugging is the process of removing software bugs, then programming must be the process of putting them in."
– Edsger Dijkstra
CISC 327
Software Quality Assurance

Lecture 18
Continuous Testing
Continuous Testing

• Today, we look at the role of testing in **software maintenance**, and the need for **continuous testing methods**

• We'll look at:
  – Software maintenance and evolution
    • Corrective, adaptive, and **perfective** maintenance
  – Continuous testing methods
    • Maintaining **functionality, failure, operational** test suites
    • Regression testing (next Thursday)
Evolution = Software Maintenance

- Maintenance is the phase of development in which the software is in production day to day by real users.
- For successful software, this is almost all of its lifetime, and the software evolves in response to observed failures and new requirements.
- Usual estimate is that up to 85% of the total software effort is in maintenance.
- Three kinds of maintenance:
  - Corrective, adaptive, and perfective.
Corrective Maintenance

- Corrective maintenance is concerned with fixing reported failures (errors) observed in the software.

- Three varieties:
  - **Coding** errors: typically easy and inexpensive to correct since they are confined within one unit.
  - **Design** errors: more expensive since they may involve changes to several units.
  - **Requirements** errors: most expensive since they often involve extensive system redesign (re-architecting) to correct.
Adaptive Maintenance

• **Adaptive** maintenance involves changing the software to work in some **new environment**:
  – new platform
  – new operating system
  – new web browser

• Characterized by no change in **functionality**, just a move to the new environment
Perfective Maintenance

• Implementing **new or changed functionality** due to changes in requirements
• Normally generated **either** by users (customers) of the software
  – e.g., need to handle a new **transaction** or a new kind of bank card or service
• **Or** by changes in the business environment the software operates in
  – e.g., changes to tax laws, new information **interchange** formats, **competition** from other businesses, etc.
(Not an actual quiz)

• Corrective, Adaptive, Perfective?

1. switching from OS X to Windows because your former teammate, who had filibustered you into using OS X, dropped the course
(Not an actual quiz)

• Corrective, Adaptive, Perfective?

2. realizing late yesterday that your class hierarchy was nonsensical
(Not an actual quiz)

• Corrective, Adaptive, Perfective?

3. deciding that ctrl-C, ctrl-D, “quit”, “exit”, and “LET ME OUT” will all be valid ways to quit the Front End
(Not an actual quiz)

• Corrective, Adaptive, Perfective?

4. fixing a bug that let anyone view a “friends only” Facebook post
(Not an actual quiz)

• Corrective, Adaptive, Perfective?

5. fixing a bug that let anyone view a “friends only” Facebook post if they’re using the latest version of Firefox
(Not an actual quiz)

• Corrective, Adaptive, Perfective?

6. adding a feature called “Friends Groups” so that changing a “Group” retroactively affects who can see old posts
EVIL TIME

• A fun (or at least evil) digression...
EVIL STORY TIME: An İ for an i

- [REDACTED], like OS X (mostly), has a case-insensitive filesystem
- [Demonstrate how this works on OS X]
EVIL STORY TIME: An İ for an i

• [REDACTED], possibly unlike OS X, puts (or used to put) filenames into a canonical form (all caps)

• Which means there’s an OS-level function to make a string all caps
What could possibly go wrong?

• At some point, [REDACTED] started supporting “non-Western writing systems”, including Turkish

• Turkish has its own alphabet, which is mostly the Latin alphabet but not quite...
What could possibly go wrong?

- Turkish has two letters similar to the Latin ‘i’:

  Iı İı

- The **capital dotless letter** is represented perfectly well by the Latin capital I (ASCII 73)
- The **lowercase dotted letter** is represented perfectly well by the Latin lowercase i (ASCII 105)
- However, there is no room in ASCII, or even in the 8-bit space, for the other Turkish letters
What could possibly go wrong?

• So, you need to use an extended character set with more space, which needs 2 bytes rather than 1 byte...

• What happens when you *correctly* make a string “canonical”, by making it all caps, the string consists only of ASCII 105 (‘i’), and the [REDACTED] OS “locale” is set to Turkey?
EVIL STORY TIME

“And they all buffer-overflowed happily ever after.

“Now go to sleep.”
Maintenance Testing

• In practice, about 65-70% of maintenance is perfective, 15-20% adaptive, and 15-20% corrective

• In all three cases, but particularly for perfective maintenance, the biggest risk associated with maintenance is that some existing functionality is broken by the changes

• This is understandable - software typically has complex and intricate relationships between parts, so changing any one part often runs the risk of unexpected effects on the rest
Maintenance Testing

• Moreover, as time goes on, the software is often maintained by programmers not involved in the original design and development
  – More focussed on the changes than the whole product

• For this reason, testing has an even more important role in quality assurance in the maintenance phase than it does in initial development and delivery
  – Helps to make sure that changes have not broken anything
Continuous Testing Methods

• Testing as a Maintenance Activity
  – Thus testing is not a one-time thing - we're never "done" testing
  – As software is maintained, if we are to maintain consistent quality, we must continue testing: both the old existing functionality, and the new introduced functionality
  – Hence, XP calls for continuous testing ("every day")
  – At a minimum, we must re-test thoroughly after every set of changes, before the changed software is released
Test Suites

• Most projects maintain test suites, sets of tests to be run on every release of the software

• Maintained in parallel with the software - often at least as much effort as coding the software itself!
  – As we have already seen, automation is essential to make this practical
Kinds of Test Suites

• Three related kinds of continuous tests are normally performed and maintained continuously in software maintenance

• Functionality (or acceptance) tests, failure tests, and operational tests
Continuous Functionality Testing

• We have already seen functionality and acceptance testing suites (you've built one!)

• When used continuously over the evolution of the software, we maintain the functionality tests by:
  – Add a new feature ⇒ add new tests for that feature
  – Recall that in XP, we must have these new tests, since they form the specification for the new software capabilities
  – Every time a feature is changed or extended, we change/extend the corresponding functionality tests
Failure Suites

• **Failure tests** are suites of examples that have been observed to cause a failure of the software in the past.

• To be effective, failure tests must be maintained over the evolution of the software by:
  – Before correcting any observed failure, create a "failure test" that causes it.
  – Becomes the *specification* of the fix - the changed software must at a minimum correct the error for the test.
  – The failure test must *cause* the error in the old software and *not cause* the error in the new software.

• All these tests go in a **failure test suite**, re-run on all future versions of the software to ensure that the failure doesn’t *reappear*. 
Operational Testing

• There's No Substitute for the Real Thing
  – Operational tests are actual production runs observed in the use of the software
    • e.g., for a banking Front End, all of the transactions done at one or more bank terminals over a whole real day of operation
    • e.g., for a banking Back Office, all of the Transaction Summary Files from a set of real front ends
Operational Testing

• There's No Substitute for the Real Thing
  – Operational test suites must be created *early* in the production life by sampling actual production runs
    • e.g., *instrumenting* a bank machine to capture the actual transactions from customers over a day
  – Must be *updated* to add new real operational tests each time significant new or changed features are added to the software
  – These tests form a *sanity check* on the software to make sure that when we are about to release a new version, it will not only still run our artificial tests but will also still handle real customer input
    • Could be *embarrassing* otherwise!
Regression Testing

• Comprehensive Continuous Testing
  – Regression testing: an automated continuous testing strategy, whose purpose is to make sure that the software does not "regress" - that is, become less effective than it has been in the past
  – Regression test suites are normally comprehensive, including three major components
    • Functionality tests, failure tests, operational tests
Regression Testing

• Comprehensive Continuous Testing
  – Functionality tests, to make sure that we still meet the basic requirements
  – Failure tests, to make sure that we haven't recreated a past failure
  – Operational tests, to make sure that we can still process real production runs
  – Each of these is maintained, either together or separately, as previously described
Summary

• **Continuous Testing**
  – Software maintenance, consisting of **corrective**, **adaptive**, and **perfective** steps, is the longest phase of software development
  – Continuous testing is **essential** to maintain quality during software maintenance
  – Regression testing combines **functionality**, **failure**, and **operational** testing in an automated continuous testing framework

• **Reference**: Sommerville Ch. 8
Next Time

• Mini-Exam #2 is on Monday, Oct. 22\textsuperscript{nd}
• Next lecture
  – Practical regression testing