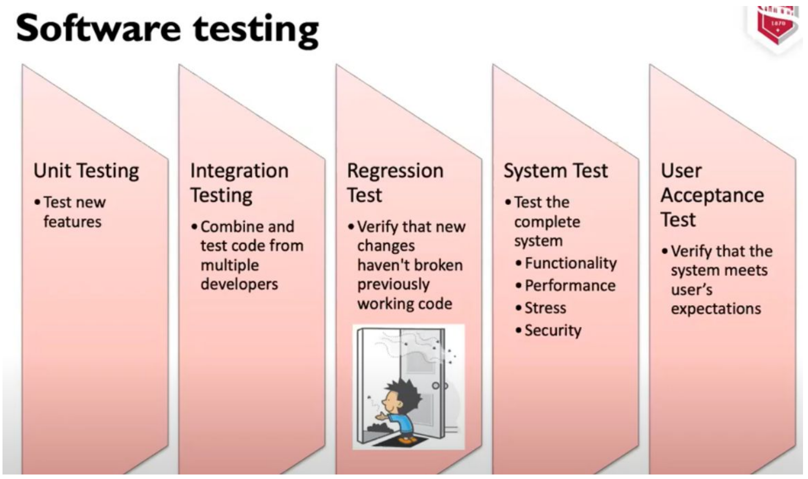
Lecture 4
SSW-555

Software Testing
The goal of software testing is to determine if the implementation meets the specifications

- Does the system do what it is supposed to do?
- **Testing** and debugging are related, but different tasks
- **Testing identifies problems - debugging fixes problems**

Traditional Methods (waterfall) versus Agile Methods (proactively reduce quality issues)



Software testing

Unit Testing
- Test new features

Integration Testing
- Combine and test code from multiple developers

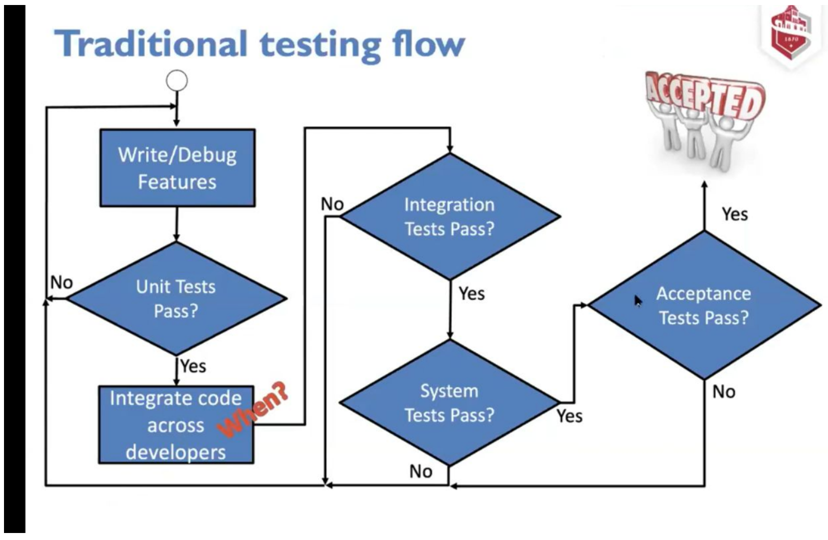
Regression Test
- Verify that new changes haven't broken previously working code

System Test
- Test the complete system
- Functionality
- Performance
- Stress
- Security

User Acceptance Test
- Verify that the system meets user's expectations

Traditional Software testing (Waterfall or Spiral)

- Code verification, design verification occurs at the end of the model. Usually, the requirements do not change very often. Take the military, if they are making a tank, then it will not change over the years.
- **Unit testing**: ensures that recently changed unit work correctly
- Who: Developers
- What: New features, bug fixes, code/branch coverage
- When: After code complete before integration
- **Integration testing:** ensures that components potentially written by different developers integrate cleanly and work together
- Who: developers and/or testers
- What: test multiple modules from potentially different developers
- When: After unit test is complete on relevant modules
- **Regression Testing:** verify that changes haven't introduced new problems in already tested code, existing release and things are not breaking
- Who: testers (NOT THE DEVS)
- What: After changes to code for bug fixes or new features
- **System testing:** ensures that complete system is correct; may include functionality, performance, stress testing, etc
- Who: System test team (not the devs)
- What: test complete system, performance and security, stress testing
- When: After all code modules are complete and tested



- **Acceptance Testing:** customer verifies that the system performs as expected and meets requirements..
- Who: customer test engineers
- What: complete system based on requirements
- When: After system test, frequently before final payment

Traditional Testing Gates

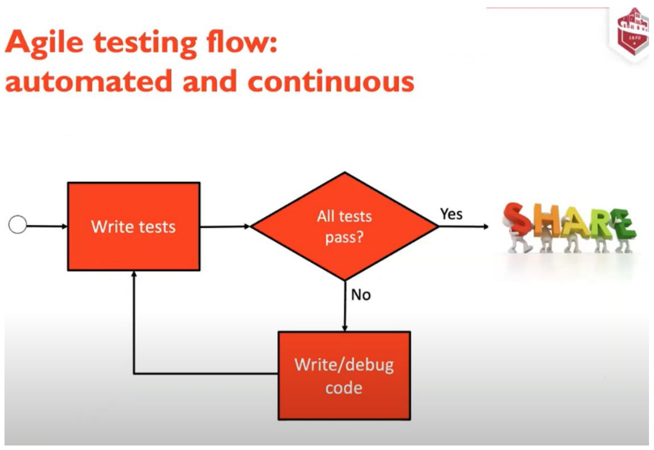
- Testing is blocked until development is "complete"
- Testing is performed by a separate test team
- Developers might not be trusted to test their own code
- Testing schedule may be compressed if development schedule slips

When to integrate?

- Agile: after every code commit, everyday
- Traditional every week/after all code complete
- Integration combines code from multiple developers

Agile Testing

- testing is not a separate phase with agile methods
- Testing is integrated into every step
- Focus on automated testing and continuous integration
- Expect frequent changes, so facilitate them
- Find problems as quickly as possible
- Agile testing is not just a passing phase
- **Unit Testing:** ensures that recently completed unit is correct
- Who: developers using automated testing platforms
- What: new features or bug fixes code / branch coverage
- When: tests written before the code is written run tests while developing code
- **All of the testing is automated. This is a fully automated process. The Developers are responsible for writing the code and the automated code.**
- Product owners do the acceptance testing. At the end of every sprint. The product owner reviews the user stories. The acceptance testing is done after every single sprint.



Agile testing flow: automated and continuous

Customer Bug Reports

- Agile methods focus on automated testing and continuous integration helps to reduce, but does not eliminate bugs found by customers.
- Scrum adds new features to each sprint.
  - How are bugs tracked? When are bugs fixed?
- Add bugs as user stories to the product backlog
- Product owner prioritizes new features and bug fixes

Test First or Test Driven Development (TDD)

- Motivation: programmers don't write tests because they don't like to, they don't have time, they have "more important" things to do
- Result: code breaks, debugging reduces productivity and doesn't improve testing
- Still no tests

- Alternative TDD scenario: write tests first, run the tests (they will probably fail), write some code, rerun the tests, debug until the tests pass (relatively little untested code at any one time so bugs are likely to be in the most recent code).
- TDD provides useful feedback: programmers get feedback when their tests pass and gets feedback when their tests fail
- Pace: write a few tests, write a few lines of code

xUnit

- Frame work for unit tests (jUnit)
- Structure

Architecture of a test

1. Fixture - creates objects and context for tests
2. Exercise - invokes methods under test
3. Result - asserts equality or something else about the results of the exercise

Collect all the tests for a class in a test suite

From TDD to Behavior-Driven Development (BDD)

Customers and some developers have trouble defining tests:

- Where to start, what to test, what to call the tests
- How can we improve the test process to include the customer?
- Change the syntax and simplify the process

Simplify testing

- Replace source code with natural language descriptions
- User story templates don't map easily to test

Behavior templates aid testing

Given, When, Then

- Evolving user stories to behavior stories.
-