Week 5 - Wednesday

COMP 3100

Last time

- What did we talk about last time?
- Exam 1!
- And before that?
  - Review!
- And before that?
  - Scrum

Questions?

Project 2

Software Quality Assurance

Software quality

- **Software quality** is how well the software meets the needs of its stakeholders
- That's a pretty frustrating definition, since "stakeholders" can mean a lot of people
  - Some of whom have conflicting desires
- Also, stakeholder needs and desires change over time
  - Especially in a field that changes as quickly as technology
- To be more precise, the International Standards Organization (ISO), defined eight dimensions of software quality

Eight dimensions of software quality

- **Functional suitability**
  - How much the product satisfies user needs
- **Performance efficiency**
  - Processing time and resources used
- **Compatibility**
  - How well the product can co-exist and interoperate with other products
- **Usability**
  - How easy the product is to learn and use
- **Reliability**
  - The extent to which the product does certain functions under given conditions and recovers from interruptions
- **Security**
  - Confidentiality, integrity, authenticity, non-repudiation, and accountability
- **Maintainability**
  - How easy it is to modify, adapt, and reuse the product
- **Portability**
  - How easy it is to make the product work in a different computing environment

Quality assurance

- **Quality assurance (QA)** is a system for making sure the product satisfies stakeholder needs
- QA focuses on two distinct goals:
- **Validation**
  - Testing if the product satisfies stakeholder needs
  - "Are we building the right product?"
  - Example: Does the customer want steak and fries?
- **Verification**
  - Testing if the product satisfies needs properly
  - "Are we building the product right?"
  - Example: Are the steak and fries cooked well?

Validation vs. verification

- The distinction is important yet confusing
- Validation is always specific to the product
  - The details of whether the product is right depend on what you're making
- Some verification might be specific to the product and some might not be
  - Using an exponential algorithm or writing confusing comments is always bad
- Validation is associated more with customer satisfaction
- Verification is more about meeting specifications
- Both are important
  - It's hard to validate things if they can't be verified
  - Verification doesn't matter if the product is invalid

Defect elimination

- Defects are bad things
  - Which QA is trying to get rid of
- Saying "defect" is more general than "bug" because it includes mistakes in implementation as well as features might be correctly implemented but are not what the customer wanted
- There are two approaches to defect elimination
  - **Defect prevention:** Keep the defect from showing up in the first place
  - **Defect detection and removal:** Find the defect and remove it
    - Example: debugging

Defect prevention

- There is no one way to prevent defects
- Instead, preventing defects must be built into the software development processes that the entire organization uses
  - **Process improvement** is making a process better
  - Training and education are necessary
- **Process guides** such as documentation standards and style guides help
- Using well-studied **design methodologies** (such as OOP) can help

Reusing ideas

- Reusing **design architectures** that have been successful in the past can prevent defects
  - Examples: MVC and pipe-and-filter
- **Design patterns** are standard patterns for OOP classes
  - Examples: decorator and factory
- Using well-studied algorithms and data structures helps a great deal
- Reusing code (often from libraries) is smart, especially since those libraries have been tested thoroughly

Formal methods and prototypes

- **Formal methods** include systems for mathematically checking that code does what it's supposed to
  - Not all code can be modeled mathematically
  - Yet some of these systems have found bugs in real software, such as TimSort, the most commonly used sort in Python and Java
- Prototypes let us explore what defects might happen before putting them in the final product
  - The opposite end of the spectrum from formal methods, since prototypes are practical rather than theoretical

Tools

- Many tools help reduce defects
- Version control tools help track code over time
- Configuration management tools allow changes in one tool to automatically update other tools
  - Examples: Puppet and Ansible
- **Integrated development environments (IDEs)**, once called computer aided software engineering (CASE) tools, can integrate many useful tools for defect prevention
  - Syntax highlighting
  - Two-way translation between code and UML models
  - Style checking

Defect detection and removal

- A good process can't keep out all defects
- Some defects will show up and must be found and removed
- Defect detection and removal techniques fall into two categories:
  - **Review and correct**
  - **Test and debug**
- Review and correct methods look at the code while test and debug methods look at the product in operation

Review and correct methods

- There's a formal name for just looking at your code for errors: a **desk check**
- A **walkthrough** is when you explain your code to someone else
- An **inspection** is a more formal process with trained inspectors
- Inspection roles:
  - **Moderator** schedules and runs the meeting and distributes the code
  - **Author** of the code
  - **Reader** who guides the meeting
  - **Recorder** who takes notes
  - **Inspectors** who check code before and during the meeting

Inspection process

1. Readiness check
   - Moderator checks that the code has no known defects already
2. Overview meeting
   - Author distributes the code
3. Preparation
   - Each inspector reviews the code individually
4. Team inspection
   - The reader guides the inspectors through the code, and they comment on it
5. Corrections
   - The moderator gives the feedback to the author, who corrects defects
6. Follow-up
   - The moderator makes sure the defects were corrected

Guidelines for inspections

- Inspectors should have a good checklist of stuff to look for
  - Checklists should be improved over time
- Information given about defects is specific
- Inspectors attend at most one inspection per day
- Inspection meetings last at most two hours
- The moderator is not a manager
- Interactions are not judgmental: defects are the focus, not the author
- The report is given to the author within 24 hours

- It can be stressful to have an inspection, but they can really help find defects

Test and debug

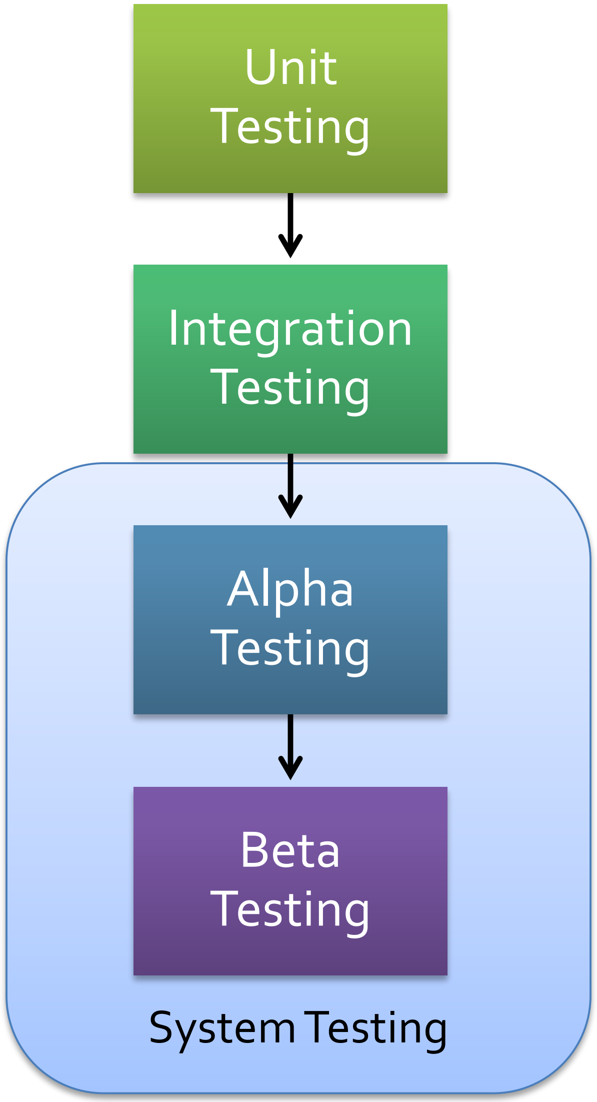
- **Testing** software helps find cases that are not obvious from looking at the code
- Software testing has some jargon:
  - A **failure** is a deviation between actual behavior and intended behavior
  - A **fault** is a defect that can give rise to a failure
  - A **trigger** is a condition that causes a fault to result in a failure
  - A **test case** is a set of inputs and program states
  - A collection of test cases is a **test suite**

Debugging

- **Debugging** is using trigger conditions to find and fix faults
- Testing is cheap
  - Just running a test can be easily automated
- Debugging is expensive
  - Fixing problems by coming up with tests and discovering the source of the problems is hard
- Debugging can be made cheaper in two ways
  - Debug small components
  - Debug as soon as you make a few changes to working code

Overview of testing

- **Unit tests** test a small piece of code (method or class) in isolation from other code
  - Often done by the author
- **Integration tests** test several small pieces of code together
  - By the author, a testing team, or both
- **Alpha and beta tests** test the whole product
  - Alpha tests usually have a testing team
  - Beta tests include users



Test and debug in waterfall

- Since requirements should be testable, each requirement should have at least one (and usually many) tests
- Unit testing happens at the implementation phase
- Integration usually happens later
  - It is often needed to make **stubs**, placeholders for code that hasn't been written yet
- System testing usually happens at the very end of the process
- Regression testing means rerunning all tests
  - This is done when any change is made the product
  - Fixing X might have broken Y

Test and debug in Scrum

- The biggest difference is that unit, integration, and system tests happen every sprint
- Unit tests are often done by authors
- However, unit tests can also be acceptance criteria for a user story
  - The user story is done when all the unit tests pass
  - These tests might be selected by people other than the authors
- Agile approaches often use **test-driven development (TDD)**
  - Write the tests *before* you write the code

Efficiency of detect and remove

- Some techniques for preventing or removing defects are more effective than others
- Inspections are often more effective than testing
- Different techniques find different bugs, so it's valuable to use them all
- The following table shows defect removal efficiencies for different techniques, from a 2013 study

Technique	Minimum (%)	Median (%)	Maximum (%)
Requirements review (informal)	20	30	50
Top-level design review (informal)	30	40	60
Detailed functional design inspection	30	65	80
Detailed logic design inspection	35	65	75
Code inspection or static analysis	35	60	90
Unit tests	10	25	50
Integration tests	25	45	60
System tests	25	50	65
External beta tests	15	40	75

Breaking it all down

Defect Elimination

Defect Prevention

- Process Guides
- Analysis and Design Methods
- Reference Architectures
- Design Patterns
- Data Structures and Algorithms
- Software Reuse
- Prototyping
- Version Control
- Configuration Management
- IDE Tools
- Training and Education

Defect Detection and Removal

Review and Correct

- Style and Standards Checkers
- Spelling and Grammar Checkers
- Reviews
  - Desk Checks
  - Walkthroughs
  - Inspections

Test and Debug

- Regression Testing
- Unit Testing
- Integration Testing
- System Testing
  - Alpha Testing
  - Beta Testing

Upcoming

Next time…

- User interaction design next Monday
- Work day on Friday

Reminders

- Read Chapter 6: User Interaction Design for Monday
- Work on the draft of Project 2
- **Office hours from 4-5 today are cancelled due to a meeting**