

CS1530, Lecture 11:
Quality Assurance and Quality Software

Bill Laboon

Hey Laboon, this is

CS1530 Software Engineering,

not

CS1632 Software Quality Assurance!

TRUTH

- .. but QA is a big part of the SDLC!

- Remember that our goal in software development is creating *quality software*.

- Quality Assurance helps *assure* you that you actually have that *quality*.

What does "software quality" mean, anyways?

"Most controversies would soon be ended, if those engaged in them would first accurately define their terms, and then adhere to their definitions."

-Tryon Edwards, *A Dictionary of Thoughts*

It varies by domain and project!

Quality is in the eye of the beholder.

Kinds of Quality - Internal and External

- External quality
  - Functional attributes - software is free from faults in design, implementation, returns correct results
    - Correct: Does what the user wants and expects (checking for this is validation)
    - Accurate: How well does it do that job? (checking for this is verification)
- Non-functional attributes (quality attributes) - Software is usable, efficient, reliable, scalable, adaptable, etc.

Kinds of Quality - Internal and External

- Internal Quality
  - Things programmers see that users do not - is the program ..
    - Understandable?
    - Maintainable?
    - Flexible?
    - Portable?
    - etc.

Defining Quality

- Sometimes internal and external quality overlap
  - For example, better software design may have the result of fewer defects
- But there are always, always, always tradeoffs, e.g.
  - More readable code —> lower performance
  - More security —> less usability
- Software engineering is the study of tradeoffs

Quality Assurance vs Software Testing

- There is a common misconception that software quality is the same as software testing.
- Not the truth - testing is simply a part of ensuring quality
- But also - ensuring processes are met, focusing on the right things, clarifying requirements, etc.

Achieving Quality

- Focus on your goals!
- Understand competing priorities, and which will win out if you need to make a trade-off

Is this a high-quality plane?

- Extremely expensive, custom-made parts
- Needs a "start cart" - could not start engines on its own
- Unpressurized cabin - pilots needed to wear a pressurized suit
- Very hard to maintain
  - Tires needed to be filled with nitrogen instead of regular air
  - Airframe was over 500° F (260° C) when it landed
- Leaked fuel on the runway - ruptures only sealed once it reached cruising altitude

Public domain photo, courtesy of Armstrong Flight Research Center of the United States National Aeronautics and Space Administration (NASA) under Photo ID: EC94-42883-4. https://en.wikipedia.org/wiki/File:Lockheed_SR-71_Blackbird.jpg

SR-71 Blackbird

- First deployed in 1964; still world record holder for fastest manned airplane (Mach 3.26, about 2200 miles/3,540 kilometers per hour)
- None lost to enemy action - was faster than the enemy's fastest missiles
- Developers at Lockheed's "Skunk Works" were told to care about speed above all other concerns

Developers will work for objectives

- Study: Five teams of developers told to work for a specific goal, different one for each team
  - Minimum memory use, most readable output, most readable code, least amount of code, minimum programming time
- In 4 / 5 cases, programmers were #1 in the category that they optimized (last case, they were #2). In all of the cases, at least one of the other categories, they were the worst.

Ways To Improve Quality of Code

- **Pair programming**
  - Person "looking over your shoulder", can find errors / edge cases / better way to do things
- Spreads knowledge amongst team members
- Is a bit slower in short run, most studies show better when viewed holistically

Ways To Improve Quality of Code

- **Code reviews / inspections / walkthroughs**
- Other person takes a look at code AFTER code is written
    - What to look for?
        - Better algorithms
        - Magic numbers, hard-to-read code
        - Performance improvements
        - Better design
        - Logic errors
        - Edge cases that are not handled appropriately
        - Tend to find fewer errors than pair programming! Faster in short run, though.

Ways To Improve Quality of Code

- **Multi-modal, multi-person inspection** - code reviews where e.g. one person looks for performance improvements, another for security issues, another for readability issues, etc.
  - Different people find different defects
  - Different strategies find different defects
  - Slower, but one of the best ways to find more defects

Ways To Improve Quality of Code

Collaborative development (developers working together, via pair programming / code reviews / multi-modal reviews / etc) tends to find more defects than testing!

Testing Is Still Necessary Though

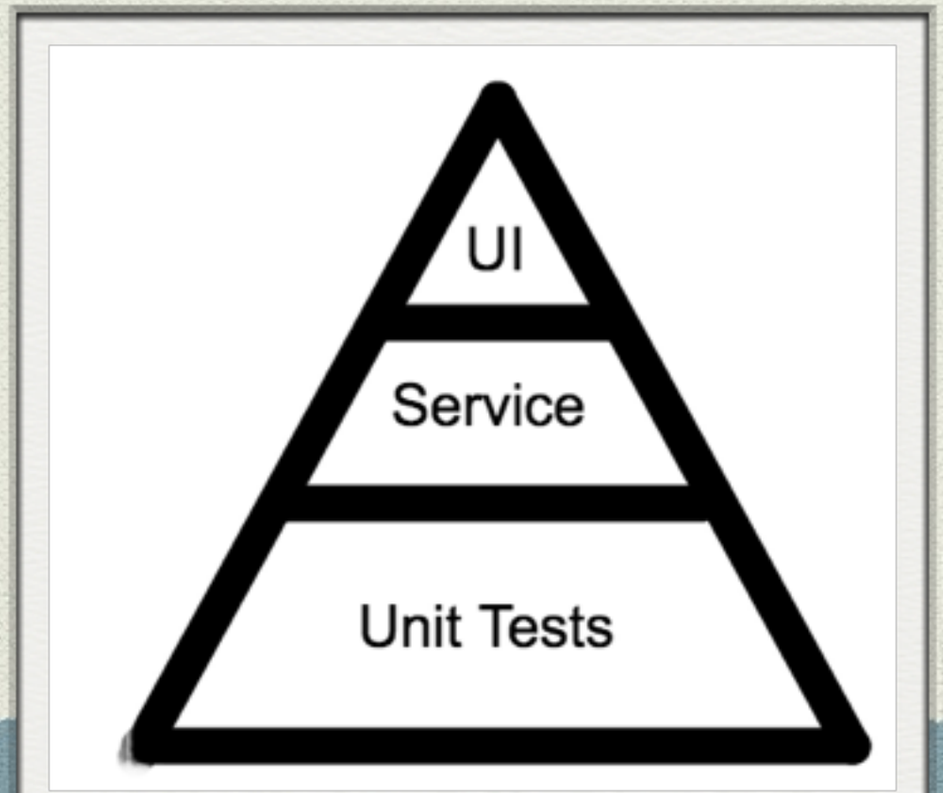
- Testing should be at least somewhat independent (remember pinata gif). An indepedent perspective will find more errors.
- This does not mean that developers should not test! Quality is everyone's responsibility.
- Testing can give you an idea of the quality of the program, so that you know how to allocate resources

Many Kinds of Testing

- **Unit testing** - JUnit, etc. Work on smallest units of code.
- **Component testing** - Execution of a class or package, tested in isolation.
- **Integration testing** - Checking that systems, classes, etc. work together.
- **Regression testing** - Ensuring that tests which previously passed still pass; that is, that the work you did did not introduce any new defects.
- **Systems Testing** - Testing the final system as a whole.
- **Performance Testing** - testing resource usage or speed of system
- **Security Testing** - Ensuring adversaries cannot access your system

Don’t overfocus on one area of testing

- The Testing Pyramid
- 10% UI (Systems) Tests
- 20% Service (Integration) Tests
- 70% Unit Tests



Kinds of Errors

- Off-by-one errors: why boundary values are so important
- Incorrect logic
- Not handling invalid data correctly
- Does not integrate correctly with other parts of the system
- Performance issues
- Security issues
- Null pointer exceptions
- Divide by zero errors
- Ambiguous or incorrectly-interpreted requirements
- Definitional issues

Things to keep in mind when testing…

- Errors aren't always programming-related! Often relate to communications or missing domain knowledge
- Problem is usually the programmer. Sometimes it's the libraries. Not often the compiler, system software, hardware, etc.
- Automation is going to save you so, so much time in the long run
  - ... but automation only checks what it is looking for!
  - Usually a good idea to have a manual test before release
- Error rate in manual testing is comparable to bug rate in the code being tested
  - Also a mind-numbing job.

Testing != Debugging

- Testing focuses on finding defects, not fixing them, or even determining the general area where they lay

Limitations of Testing

- Testing does not prove absence of defects! Ever! There is always the possibility of failures that you did not think of (e.g., threading and concurrency issues, edge cases, meteor strike, etc.)
- Exhaustive testing is (almost) impossible.
- Testing helps you to see software quality; it does not improve it on its own. However, it is an important way to ensure that you are developing quality software.

The General Principle of Software Quality

- In the long run, writing quality software is "free" due to the time saved in development
- In the short run, seems more expensive (Go-Kart / Jet Plane analogy)
- Numerous studies have shown that time/resources saved by producing quality software more than make up for extra time (for larger products)
  - Why? Better code, fewer defects, defects fixed sooner

But this may not always be your objective!

- Paul Graham - sometimes better to put out something fast, iterate and fix later
- When your "customer" is lots of people - e.g. Facebook, Dropbox, etc.
- "If you're not embarrassed by your first version, you waited too long to ship"
- Side note: Graham wrote some very interesting, thought-provoking articles on software engineering earlier in his career (now it's mostly “startup stuff”). See "Why Arc Isn't Especially Object-Oriented" (http://www.paulgraham.com/noop.html), "Java's Cover" (http://www.paulgraham.com/javacover.html), and "Beating the Averages" (http://www.paulgraham.com/avg.html). All are over a decade old, but very relevant.