[This document was copy-pasted from some slides… it may be stale]

**What is Quality Assurance?**

**User Satisfaction**  = **Compliant Product** +   
**Good Quality** +   
**Delivery w/in Budget  
& Schedule**

The equation shown here comes from Robert Glass, as quoted in Pressman's Software Engineering Textbook. Glass' point is that "quality is important, but if the user isn't satisfied, nothing else really matters." So we will focus here on quality, but with the understanding that quality cannot come at the expense of neglecting the requirements, missing deadlines, or exceeding your budget. But as Ousterhout would point out in his book "A Philosophy of Software Design", quality code (which he defines most notably as simple code) is often the key to meeting these other requirements!

Simplicity

Readability

Maintainability

Flexibility / Modularity

Minimization of Duplication

Optimizes Trade-Offs

The three books here are the texts that I am most familiar with, although to be honest I have not read any of them in their entirety. I think this list of 6 elements would be a nice consensus of what the authors agree to be principles that contribute to quality software engineering.

**Testing Quotes**

Quotes found in Pressman book

**Code Like NASA**

YouTube Video

**Testing Strategy**

There are two important terms when it comes to testing: verification and validation

*Verification* determines if the code is functioning correctly

*Validation* determines if the code is fulfilling the requirements

**Quality Assurance is More Than Testing**

In the Agile Methodology, the Quality Assurance stage is often taken as a fancy way of saying "testing". But QA is more than just testing, even if testing is the largest part. One of the other parts of QA can be Code Reviews. This information comes almost directly from the Pressman Software Engineering book. It is a set of guidelines to help guide the code review process.

The quote at the bottom is not to discourage you from holding code reviews, but rather to help you appreciate the guidelines in order to avoid wasting everyone's time.

**Different Types of Testing in Software**

* **Unit Testing** - We've talked about this one the most, it's testing the lowest level modules that basically do something on their own without relying on any other piece of the software project.
* **System Testing** - We've talked about system testing a little bit, it's the testing of the entire system and whether **all** the parts work together as a whole rather than simply working on their own.
* **Integration Testing** - It was left off of this list. Integration testing is somewhere between Unit Testing and System Testing. You can think of it as testing the major functions that rely on the smaller functions that have already been Unit Tested, but still not testing the system as a whole.
* **Functional Testing** - Is testing that verifies each of the requirements are met – closely aligned with the Acceptance Criteria in the Requirements Document. Functional testing is higher up than integration testing but comes before system testing.
* **Acceptance Testing** - Very similar to Functional Testing except that Functional Testing is usually performed by the developers and/or independent test teams whereas Acceptance Testing is performed by the customer. But theoretically, they would use similar tests.
* **Adhoc Testing** - Basically, what you do already to test your code along the way
* **Continuous Testing** - A systematic and repetitive process to test your program at each stage of the SDLC… to do testing regularly rather than at stage 4 (it is "smoke testing" that adds a form of daily testing)
* **Regression Testing** - Testing older parts of the system after any new update
* **Blackbox Testing** - Testing based on the description/API of the code but without any internal knowledge
* **Whitebox Testing** - Testing with full knowledge of the internals of the system
* **Beta Testing** - Put the software into the hands of reals users but with the understanding that the software is not finished and bugs are likely to be found (and must be reported)

**Bugs vs Errors vs Defects**

To support his claim on the increasing cost of correcting an error, Pressman cites a 1981 textbook by Barry Boehm and a 1981 course by the IBM Systems Sciences Institute (an internal organization from IBM). A few people have challenged this claim by saying that the System Sciences Institute never published data to support these claims and therefore, the Bug Cost Curve might be a myth. Laurent Bossavit is one of the most prominent ([GitHub Gist](https://gist.github.com/Morendil/ebfa32d10528af04e2ccb8995e3cb4a7) and [The Register](https://www.theregister.com/2021/07/22/bugs_expense_bs/)) and there's even an academic paper in [Empirical Software Engineering](https://www.researchgate.net/publication/308264787_Are_Delayed_Issues_Harder_to_Resolve_Revisiting_Cost-to-Fix_of_Defects_throughout_the_Lifecycle) supporting these doubters.

However, most major software companies and government agencies stand by Pressman's claims. Google says that its own internal products support the original numbers of 1000x savings in a [presentation at the IEEE International Conference on Software Testing, Verification and Validation (2010)](https://docs.google.com/presentation/d/1kAsvIzKMU47pxAWOz2oyV394N9zvRZ_nDM3ijSAPhtk/htmlpresent).

If you ask me, the doubters are being provocative and drumming up trouble where there isn't any. Of course you can find a few projects with bugs that are an exception to the rule. But the Bug Cost Curve as a general rule is sound and it's what we should expect when planning a software project. The Empirical Software Engineering article writers (Menzies, Nichols, Shull, Layman) and plenty of others suggest that the Agile Methodology itself with its iterative nature and continuous testing can help protect against such extravagant costs, along with modern development and testing tools.

**Common Python Testing Frameworks**

Pytest and PyUnit

**Blackbox Testing**

def is\_prime(n):

Ask students to identify our assumptions, generalizations, typical cases, edge cases, etc?

**Whitebox Testing**

Ask students How does full code access change our approach to these tests?

def is\_leap\_year(year):

**Leap Year Code**

Code example

**How to Test**

The code snippet on the right is from the Zune Music Player, specifically the Zune 30 that came out in 2006. It is supposed to calculate the number of years since 1980. It is based on having the number of days since 1980. The problem is that 2008 was a leap year, so on Dec 31 2008 there were 366 days left in the counter. This meant that the while loop was going to keep going around. Then, since 2008 was a leap year, we went into the first "if" statement. But 366 > 365 so we finished the loop with days unchanged. This resulted in an infinite loop and the Zune never booted. Microsoft's solution was to wait until Jan 1st 2009 when the problem would correct itself!

**Advanced Testing (GUIs, Pictures, Sounds, Movies)**

How would you test a function that manipulates a picture or music?

Here are some other ideas: [softwareengineering.stackexchange.com](https://softwareengineering.stackexchange.com/questions/166517/how-to-unit-test-image-processing-code) and [stackoverflow.com](https://stackoverflow.com/questions/3487774/whats-the-best-way-to-unit-test-code-that-generates-images).

Note that cryptographic hashing might be a good way to quickly detect changes in large data. However, they could not be used for any sort of fuzzy testing.

**Metrics and Standards**

There's a lot more to be said about *quantifying software quality* with metrics and industry standards.

A typical measure of reliability is reliability rate over a specific time frame. So something like 96% correct over 2 week time frame. In other words, if you run the product 100 times for 2 weeks each, there would be 96 correct runs and 4 times when something went wrong.

Some other common measures are Mean Time Between Failure (MTBF) which is the sum of Mean Time to Failure (MTTF) and Mean Time to Repair (MTTR).

As stated in the Geeks for Geeks article: A six sigma method is one in which 99.99966% of all the products to be produced have the same features and are of free from defects.

ISO 9000 is one such industry standard and a company can become certification which shows that they have high standards and processes in place within their software engineering process.

**Summarizing Software Quality by Pressman**

requirements (x2)

standards

implicit (x2)