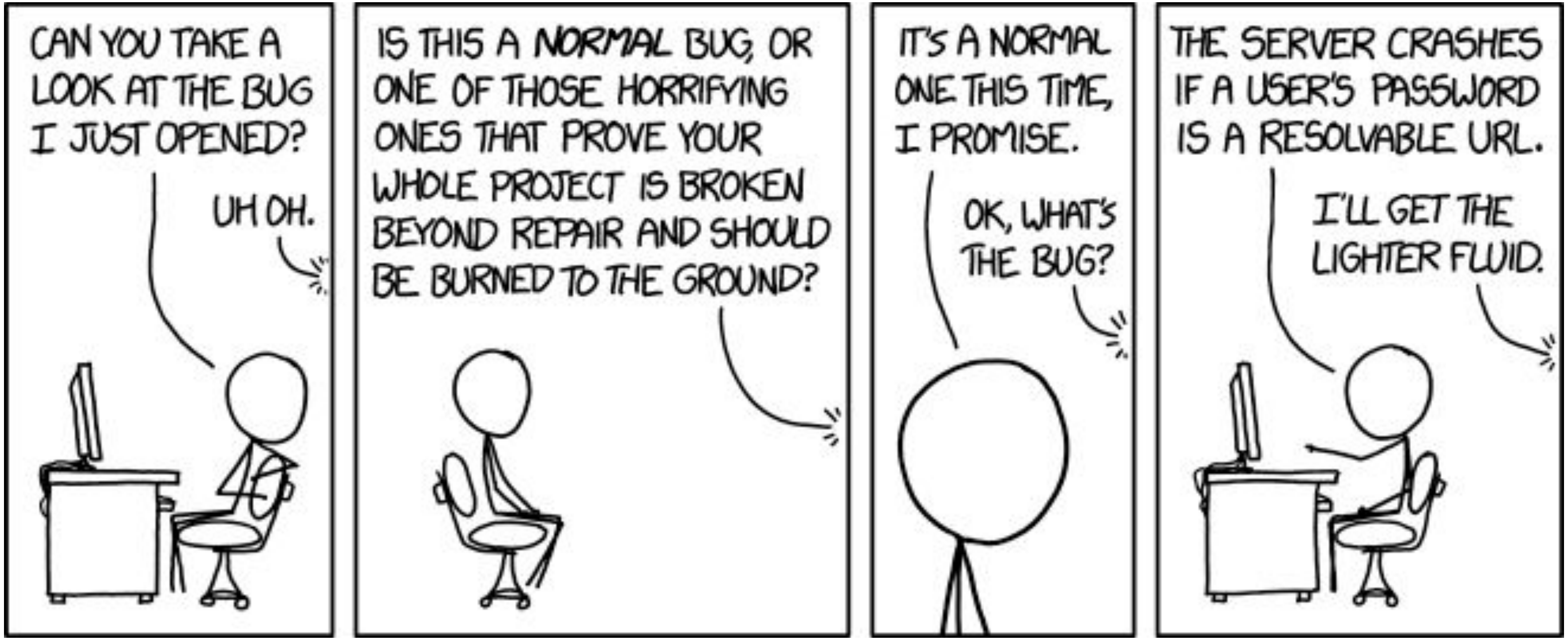
Introduction to

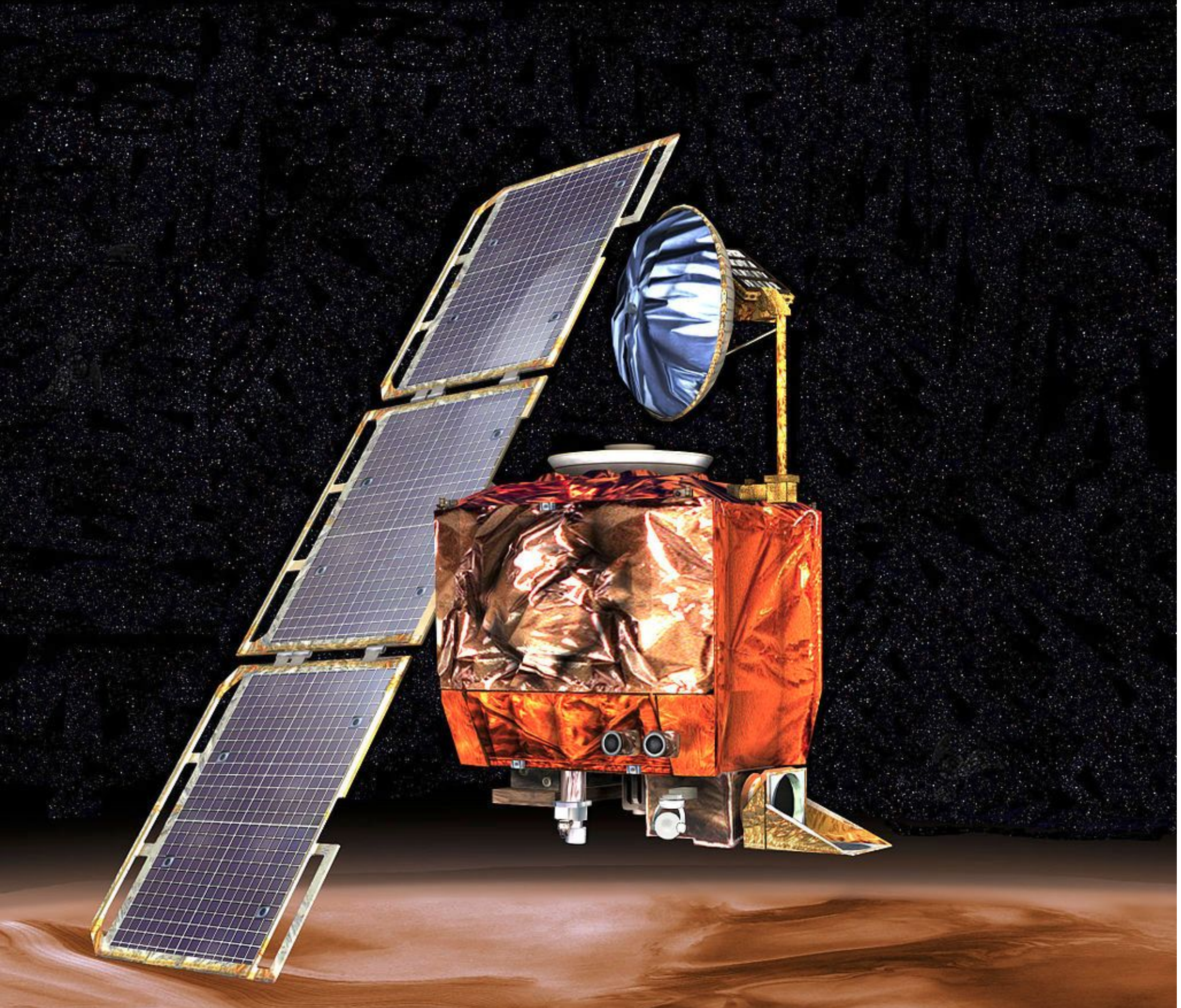
Software Testing

sebinbenjamin


CAN YOU TAKE A LOOK AT THE BUG I JUST OPENED?
UH OH.
IS THIS A *NORMAL* BUG, OR ONE OF THOSE HORRIFYING ONES THAT PROVE YOUR WHOLE PROJECT IS BROKEN BEYOND REPAIR AND SHOULD BE BURNED TO THE GROUND?
IT'S A NORMAL ONE THIS TIME, I PROMISE.
OK, WHAT'S THE BUG?
THE SERVER CRASHES IF A USER'S PASSWORD IS A RESOLVABLE URL.
I'LL GET THE LIGHTER FLUID.

https://xkcd.com/1700/

Airbus A400 crash



Mars Climate Orbiter

“Software testing is the process of running a program with the intention of finding errors”

— Glenford J. Myers

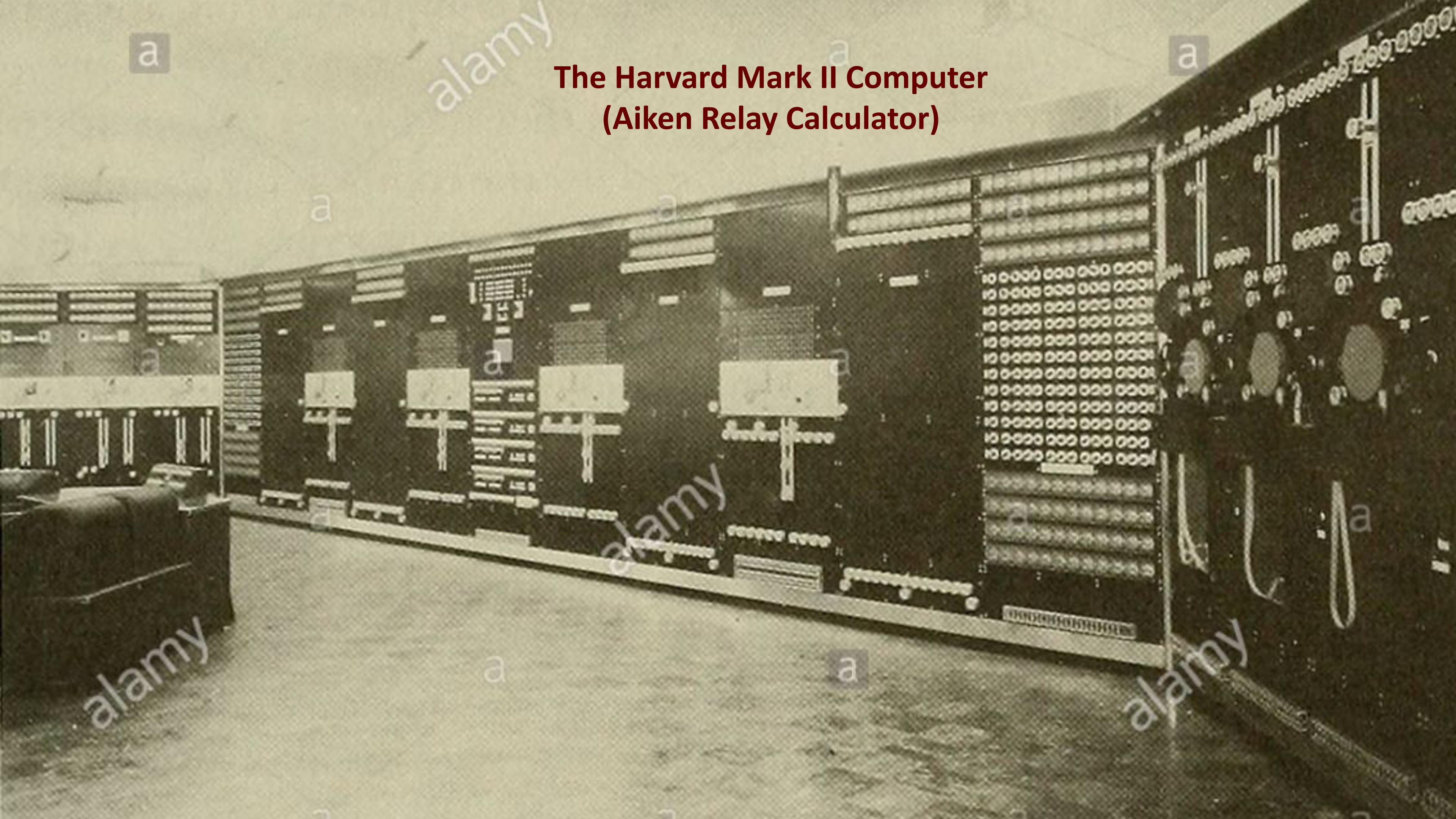
A very brief history of Software Testing

Programmers need to fulfill several distinct goals in order to make users happy.

Increasing ***demand for more frequent software releases*** slow-and-steady development was abandoned, in favor of release-early-and-often approach.

Releasing software that worked on any PC required careful ***configuration testing*** of the many possible environment variables.

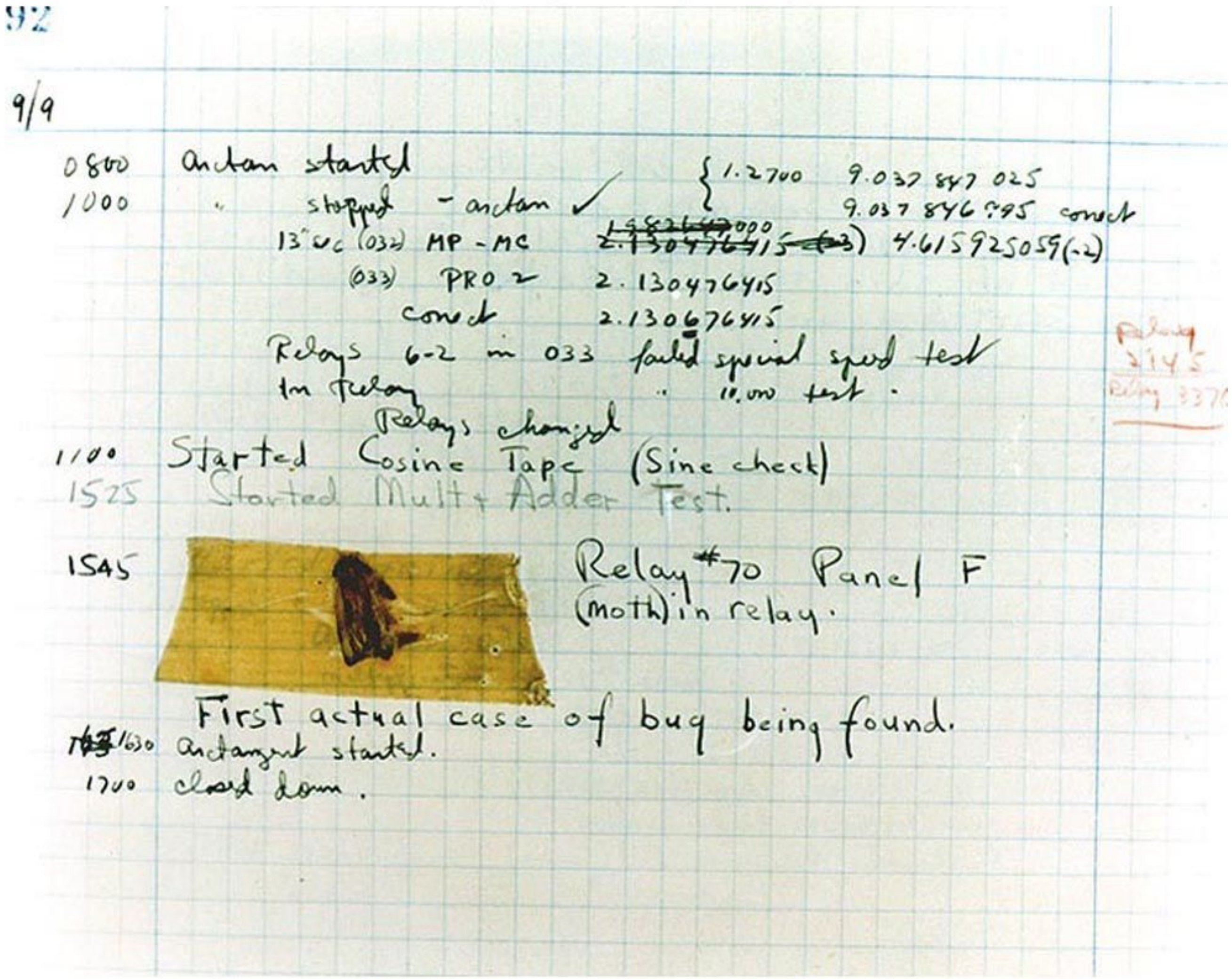
The Harvard Mark II Computer (Aiken Relay Calculator)



Debugging period (1947–1956)

- In 1947, the terms “bug” and “debugging” were coined.
    - Grace Murray, a Harvard University scientist who worked with the Mark II computer, detected that a **moth had got stuck in a relay** causing it not to make contact.
    - He detailed the incident in the work log, pasting the moth with tape as evidence and referring to the moth as the “bug” causing the error, and to the ***action of eliminating the error as “debugging”.***

Photo # NH 96566-KN (Color) First Computer "Bug", 1947



92

9/9

0800 Arctan started

1000 " stopped - arctan ✓ {1.2700 9.037 847 025

9.037 846 995 correct

13" UC (032) MP - MC ~~1.982147000~~ ~~2.130476415~~ (-3) 4.615925059(-2)

(033) PRO 2 2.130476415

correct 2.130676415

Relays 6-2 in 033 failed special speed test

In relay " 10.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)

1525 Started Mult + Adder Test.

1545 Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

~~143~~ 1630 Arctangent started.

1700 closed down.

Relay 2145

Relay 3370

- Tests were focused on the **hardware** because it was not as developed as today and its reliability was essential for the proper functioning of the software.
- Tests that were performed were only of a corrective nature by taking certain measures in order to make the program work.

Demonstration period (1957–1978)

Charles Baker in 1957 explains the need to develop tests to ensure that the software meets the pre-designed requirements (testing) as well as the program’s functionality (debugging)..

Test development became more important as ***more expensive and complex applications*** were being developed.

The aim was to demonstrate that the program did what had previously been said to be done, using expected and recognisable parameters.

Destruction period (1979–1982)

Pursuing the goal of demonstrating that a program is flawless, one could subconsciously **select test data that has a low probability of causing program failures**, whereas if the goal is to demonstrate that a program is flawed, our test data will have a greater probability of detecting them and we will be more successful in testing and thus in software quality.

The tests will t***ry to demonstrate that a program does not work as it should***, contrary to how it was done until then.

Evaluation period(1983–1987)

A new methodology was proposed that integrates ***analysis, revision and testing activities during the software life cycle*** in order to obtain an evaluation of the product as development progresses.

Emergence of tools for the development of **automated tests**, which notably improved efficiency.

Prevention period (1988 — present)

Redefined the concept of testing as the planning, design, construction, maintenance and execution of **tests and test environments**.

This stage was mainly reflected in the **appearance of the testing phase at** the earliest stage in the development of a product, **the planning stage**

The History of Software Testing

... - 1956

DEBAGGING

testing was often associated to debugging: there was no clear difference between testing and debugging

1957 - 1978

in this period it was shown, that software satisfies the requirements

DEMONSTRATION

1979 - 1982

DESTRUCTION

the goal was to find errors

1983 - 1987

during the software lifecycle a product evaluation is provided and measuring quality

EVALUATION

1988 - ...

PREVENTION

to prevent faults in requirements, design, and implementation

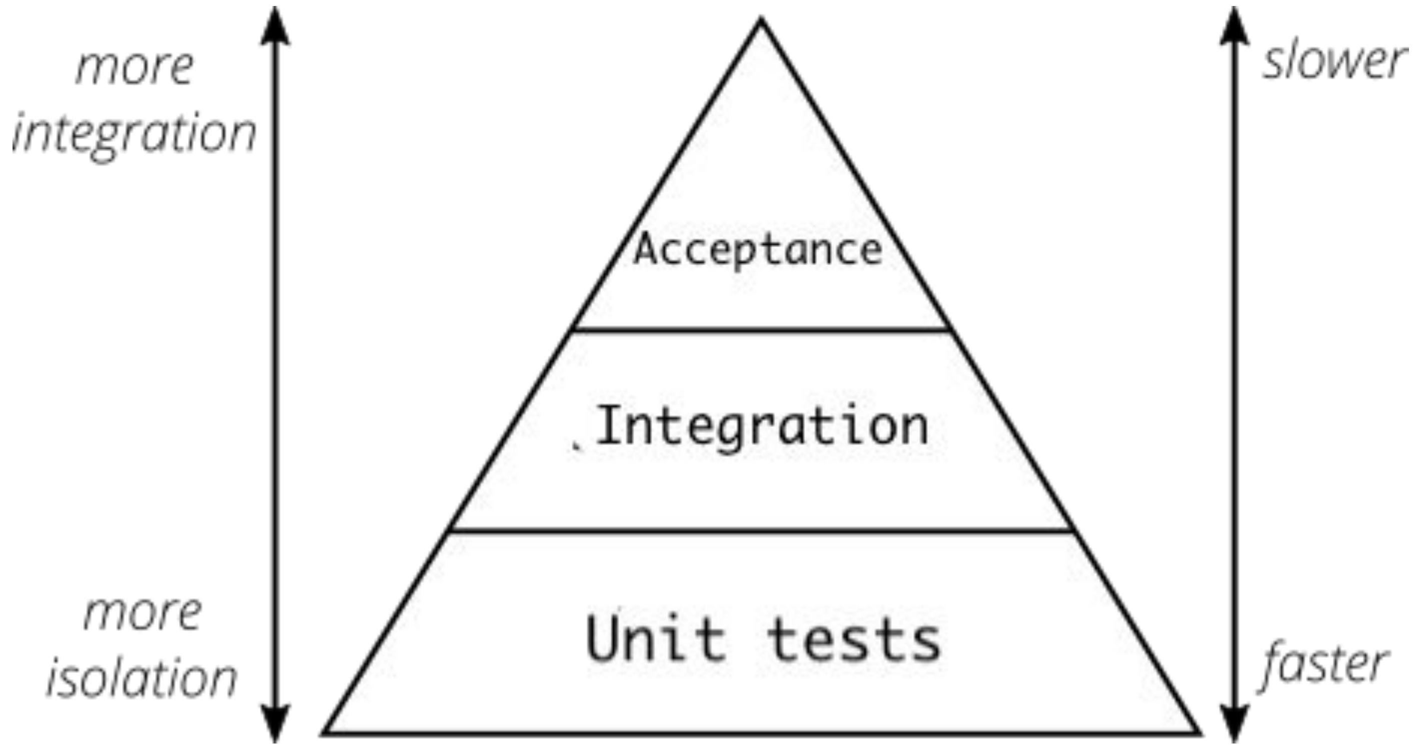
The need for testing

1. To ensure there is no difference between the real and the expectated
2. To make sure your product is scalable, no matter how many people are using.
3. To eliminate as much as possible, the chance of bugs.
4. To offer a product that can perfectly work on different browsers and tech devices
5. To deliver the best product that we can.

https://crossbrowsertesting.com/blog/test-automation/history-of-web-browsers/

Common types of Software Testing

1. Unit Test
2. Integration Test
3. Acceptance Tests/End to End tests
4. Functional Testing
5. Smoke Test
6. Regression Test


more integration
more isolation
Acceptance
Integration
Unit tests
slower
faster

Unit testing

They are dedicated to testing a unit. The nature of these tests is that the module being tested must be simple, generally **small and atomic, not dependent on other modules**.

These tests are **performed by the developers themselves** for knowledge and agility.

Basic principles for the development of unit tests known by its acronym **FIRST**

- **F**ast
- **I**ndependent/Isolated
- **R**epeatable
- **S**elf-validating
- **T**imely

Test-Driven Development (TDD)

A development practice that consists of **first developing the unit tests** based on the logic that the program will have, and then **writing the code that makes the test work properly.**

The **purpose** of this practice is to **have a clear structure before starting with the beginning,** a more robust and efficient code and to reduce the iterations between testers and developers.

Code coverage

Code coverage, or Coverage, is a measure that helps us to know and identify **which parts of the code have been tested and which have not**, obtaining as a result a percentage of the code that is being tested through our unit tests.

This technique is really useful to **identify unnecessary code, impossible conditions and cyclomatic complexity**, in example to obtain a quantitative measurement of the logical complexity of a program.

```
function checkIfEven(userInput) {
  if (userInput % 2 === 0) {
    return true;
  } else {
    return true;
  }
}

console.log(checkIfEven(3) === false ? 'Sucess' : 'Failed');
console.log(checkIfEven(4) === true ? 'Sucess' : 'Failed');
console.log(checkIfEven(2) === true ? 'Sucess' : 'Failed');
```

Integration Testing

The integration tests determine whether the various **independently developed components function correctly** when they are connected to each other.

Combination of different components can affect the individual functionality of each.

Any of Black Box Testing, White Box Testing and Gray Box Testing methods can be used

Aproaches

- Big Bang
- Top Down
- Bottom Up
- Sandwich/ Hybrid

End-to-end testing

A technique that tests the entire software product from beginning to end to ensure the ***application flow behaves as expected***.

The main purpose E2E testing is to **test from the end user's experience** by simulating the **real user scenario** and validating the system under test and its components for integration and data integrity.

Necessary due to the complex nature of modern software systems.

Functional vs non-functional requirements

- Functional: Which tests the real-world, **business application of a software** solution.
    a. For example, a ride-sharing app like Uber must be able to **connect end users to drivers** when all conditions are met, at the bare minimum.
- Non-functional: Which tests the **remaining requirements** of the software (for example performance, security, data storage, etc.)
    a. With the ride-sharing example, this type of testing will ensure that the app is **fast and efficient** when performing its most essential functions, like connecting end users to drivers in this case.

Thank you