mini-lecture

WELCOME TO

TESTING 101

Pavlína Wurzel Gonçalves
University of Zurich
Switzerland

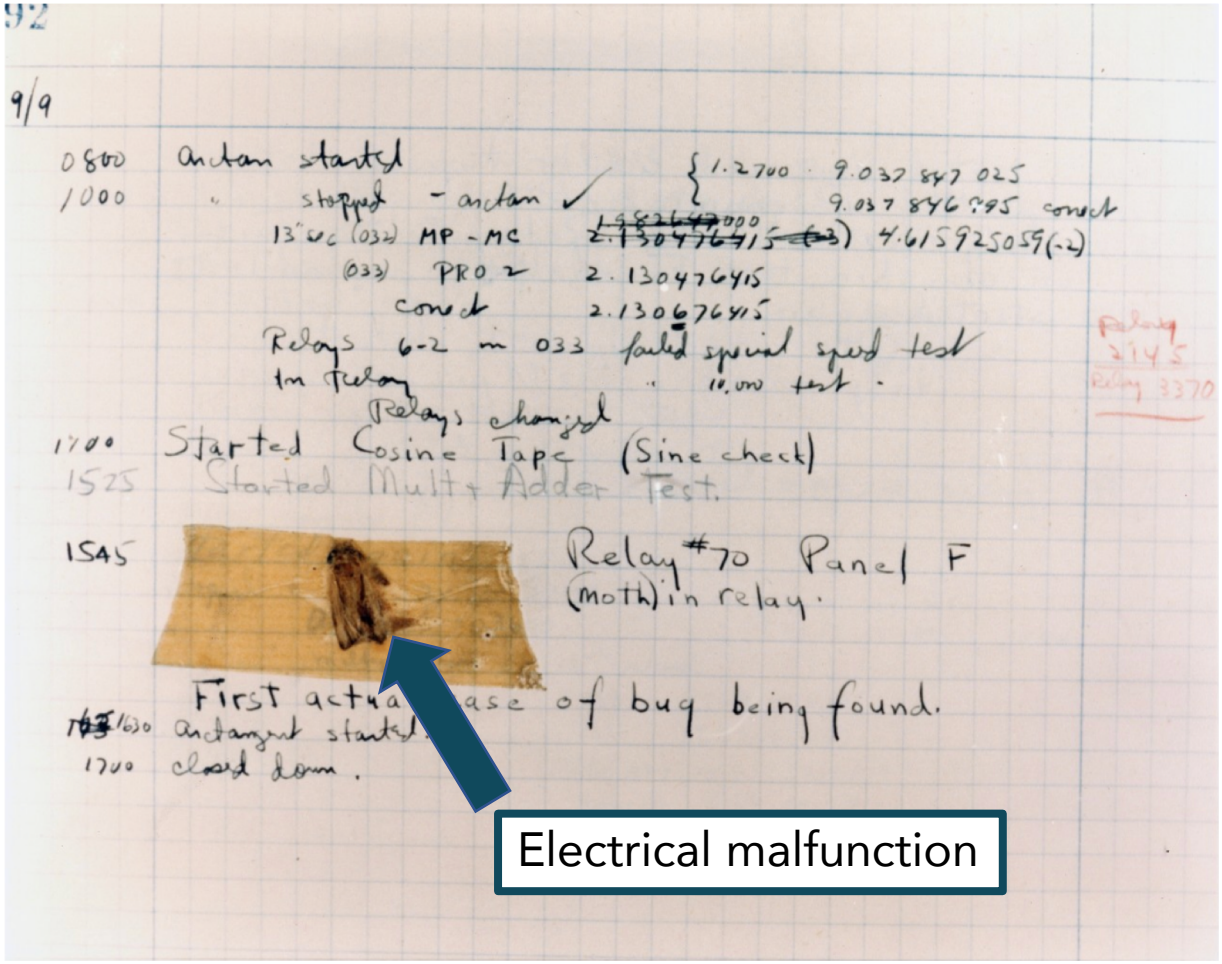
Bugs

flaws or errors in a software program

1947
Mark II Aiken Relay Calculator@
at Harvard University

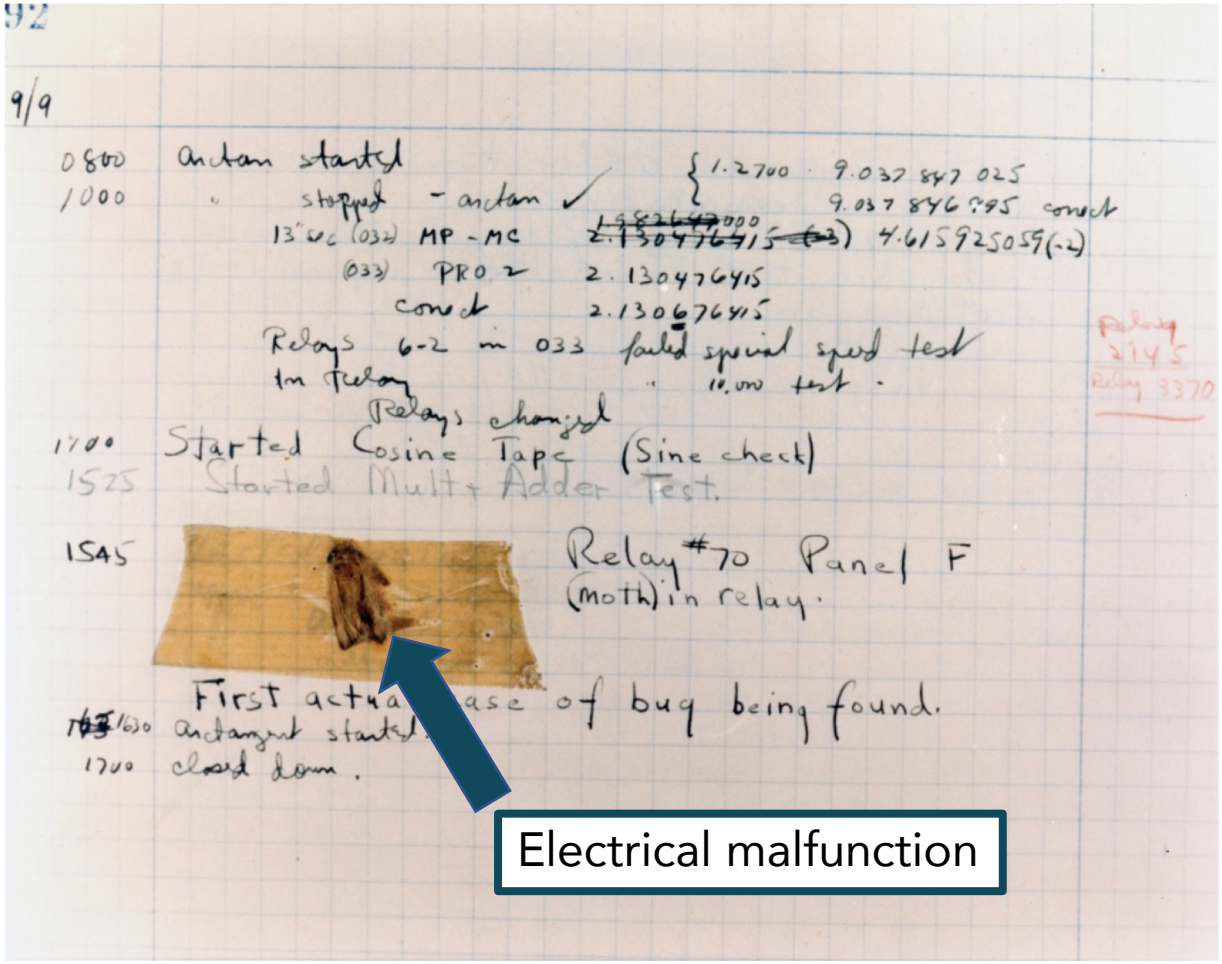
"First actual case of bug being found"

Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia



Bugs

flaws or errors in a software program



Testing
A process of identifying bugs – malfunctions and defects in the software

1950’s – 1960’s

Structured testing a rare sight

- Programmers were, first of all, expected to write programs that had no defects in the code
- NASA, Gerald M. Weinberg - formed the ”first real professional testing group”

The Mariner 1 Spacecraft, 1962

- A hyphen missing in coordinates calculation
- The rocket was destroyed in early flight
- $169 million (nowadays value)

”There is no relationship between the ‘size’ of the error and the problem it causes. Thus, it is difficult to formulate any objective for program testing.”

- Gerald M. Weinberg

1960’s-1970’s

Testing and testing teams becoming a necessity

- Commercialization of software development
- Software becoming more complex
- Huge scaling of companies, shortage of competent personnel
- Large amounts of low quality code
- Challenges for developers and the software engineering process

1980’s-1990’s

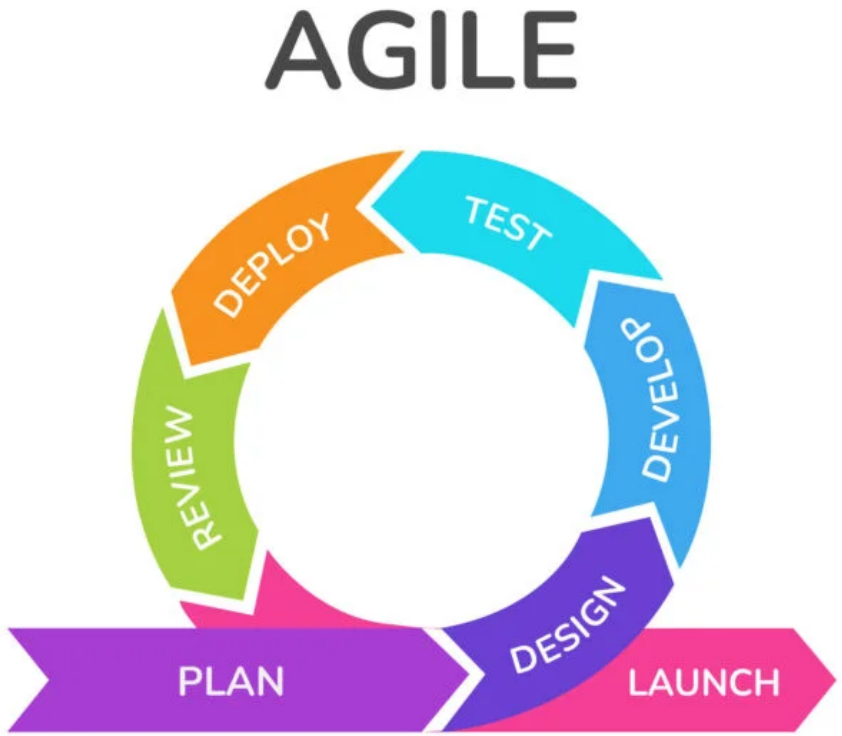
Testing embedded in the software engineering process

- Structured and automated testing on the rise
- Testing became the tool to measure software quality

2000’s

Testing done all the time

- Agile Software development, Extreme Programming
- Delivering software in short cycles
- Software engineering more flexible, iterative and collaborative
- Testing not done only after writing the code



Software Testing Nowadays

- **Testing automation**
  - Testing part of every step in the software development process
- **Tooling and Frameworks**
  - Tools enabling planning and integration with other activities in software development
- **DevOps and Continuous Integration/Deployment**
  - Short iterations and frequent deployment require seamless transitions between development, testing, and production stages
- **Enhanced User Experience Testing**
  - Emphasis on ensuring systems are easy to use and adopt by users

Software defects are still here…

Published 12:58 PM SELČ, April 3, 2022

Share

THE HAGUE, Netherlands (AP) — Trains operated by the national rail network were halted across the Netherlands Sunday by what the operator called a technical problem.

Erik Kroeze, a spokesman for railway operator NS, said the problem was in a planning software system. He said there were no indications it was caused by a cyberattack.

NS said trains would be halted until 5 p.m. while it sought to fix the problem.

"We are working hard on recovery, but unfortunately it is not yet possible to say how long this situation will last," NS said in a statement on its website.

… with threats in how software is (mis)used

Cyber warfare and cyber defense

The WhisperGate Campaign (Ukraine, 2022)

What is software testing

executing a piece of software to see if it behaves as expected

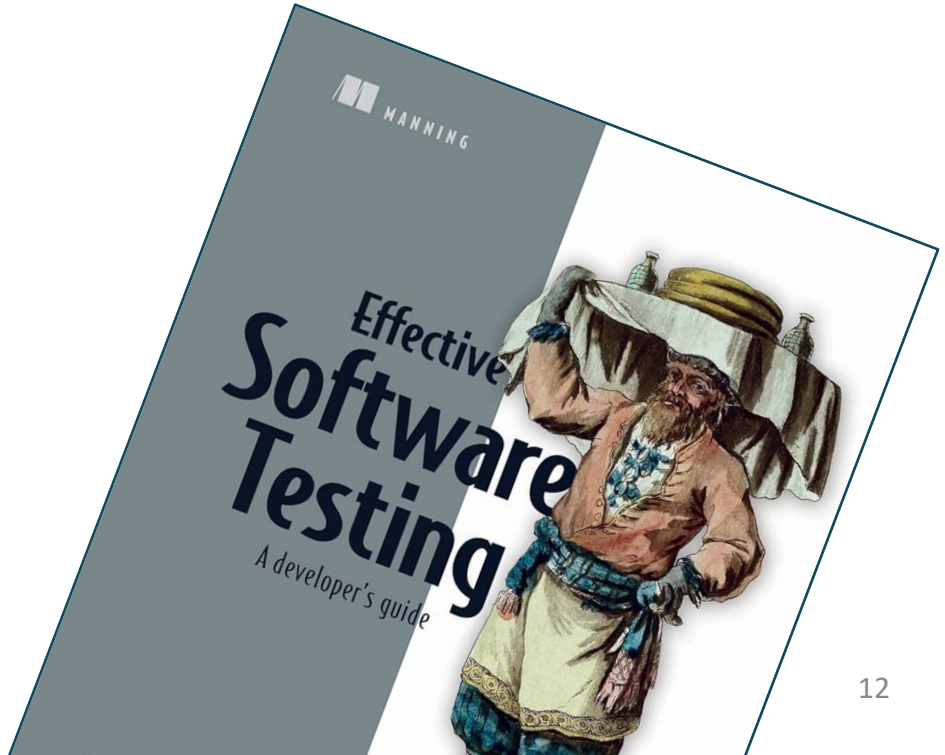
What is software testing

Validation

Are we building the right thing?

Verification

Are we building it in the right way?

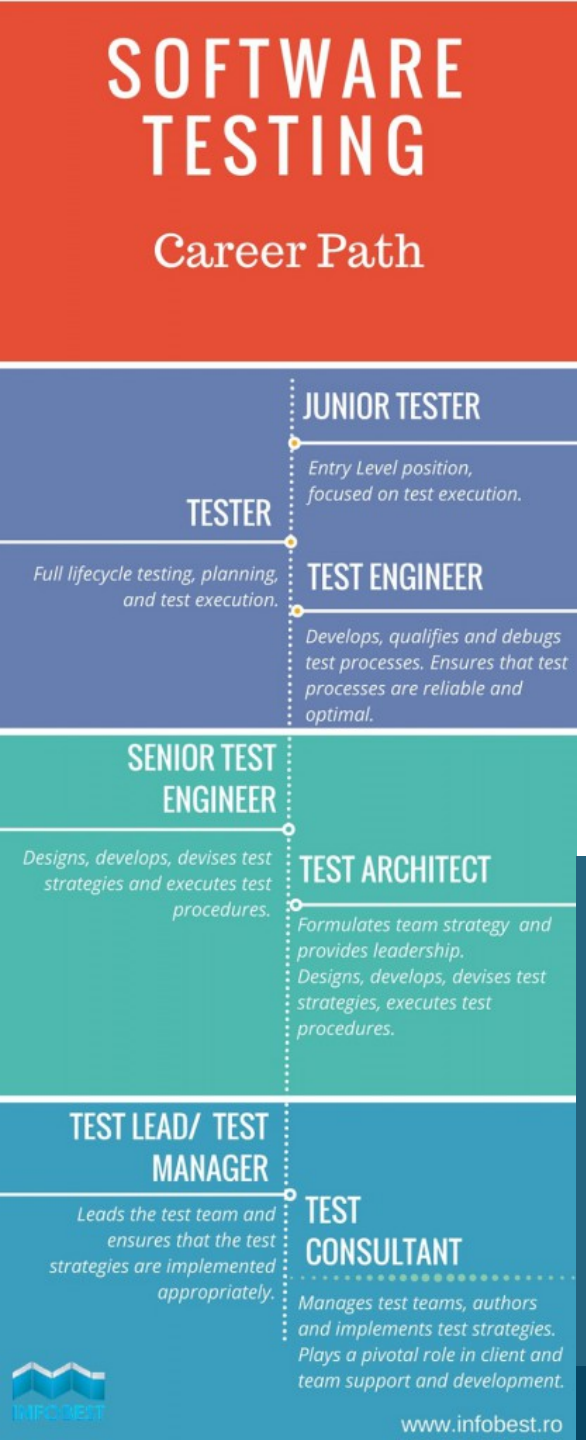


Aniche, M. (2022). *Effective Software Testing: A developer's guide*. Simon and Schuster.

But…

- The majority of developers does not test
- Developers rarely run their tests in the IDE

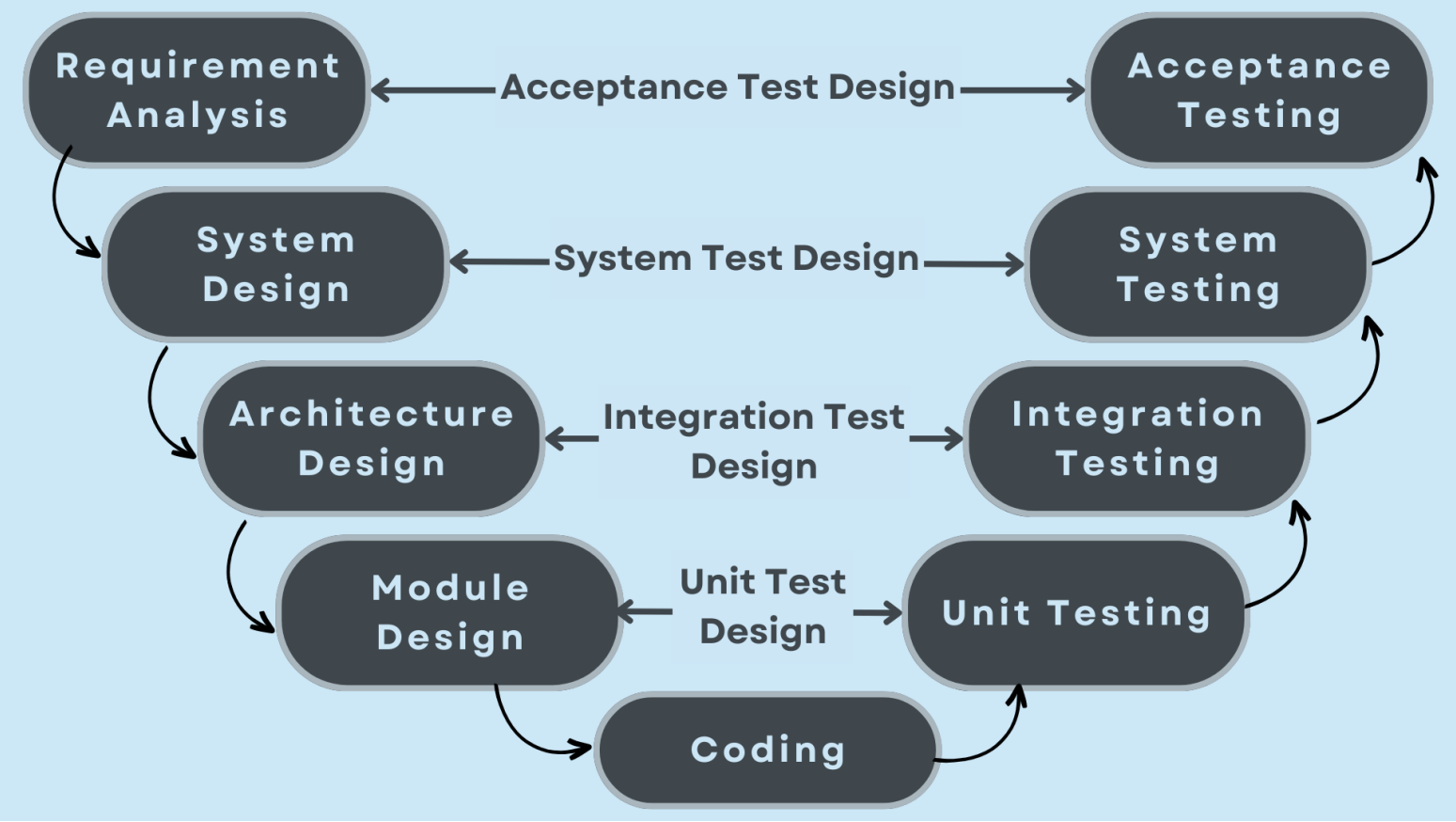
Beller, M., Gousios, G., Panichella, A., & Zaidman, A. (2015, August). When, how, and why developers (do not) test in their IDEs. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 179-190).



Benefits for you

1. Taking a career in Software Testing and Quality Assurance
2. CONFIDENCE ABOUT YOUR CODE for yourself and others
3. Documenting your project's requirements
4. Safe software maintenance and evolution

The V model



Each phase of the software development life cycle there is a corresponding testing phase

Testing adapts to the software development model – linear or iterative

Testing in any case is performed at all levels

Lets build…

an online banking application

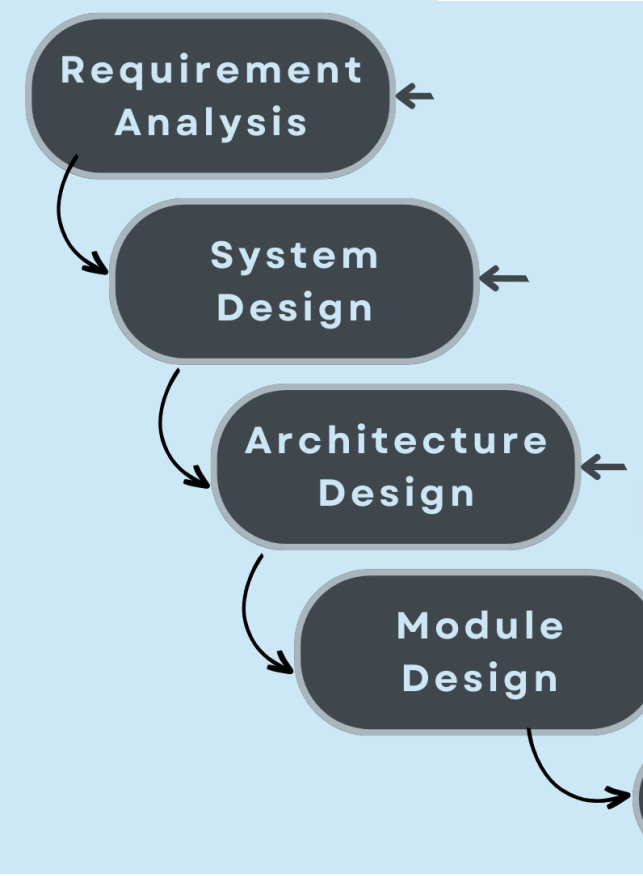
Login on valid credentials

View current balance

Deposit money

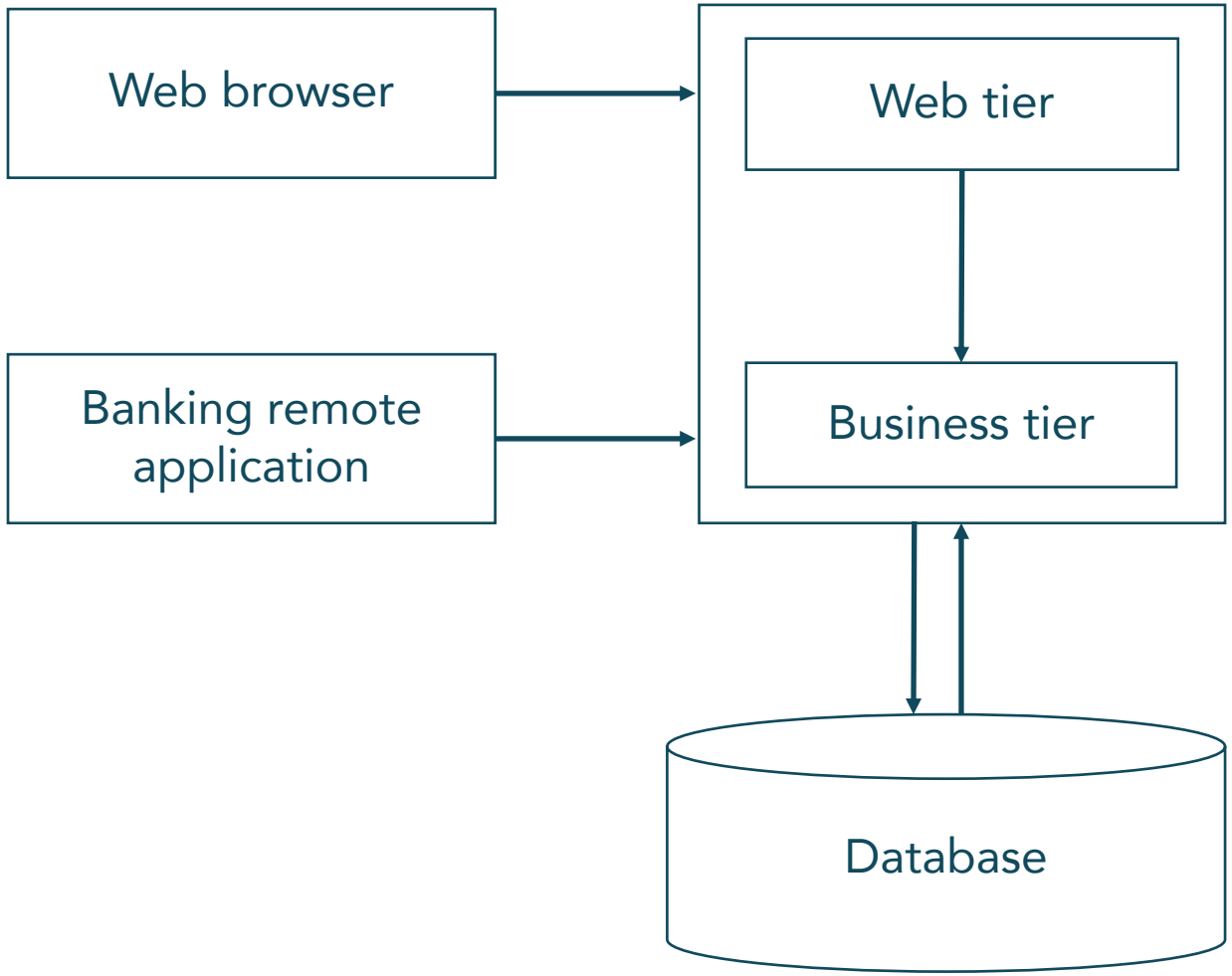
Withdraw money

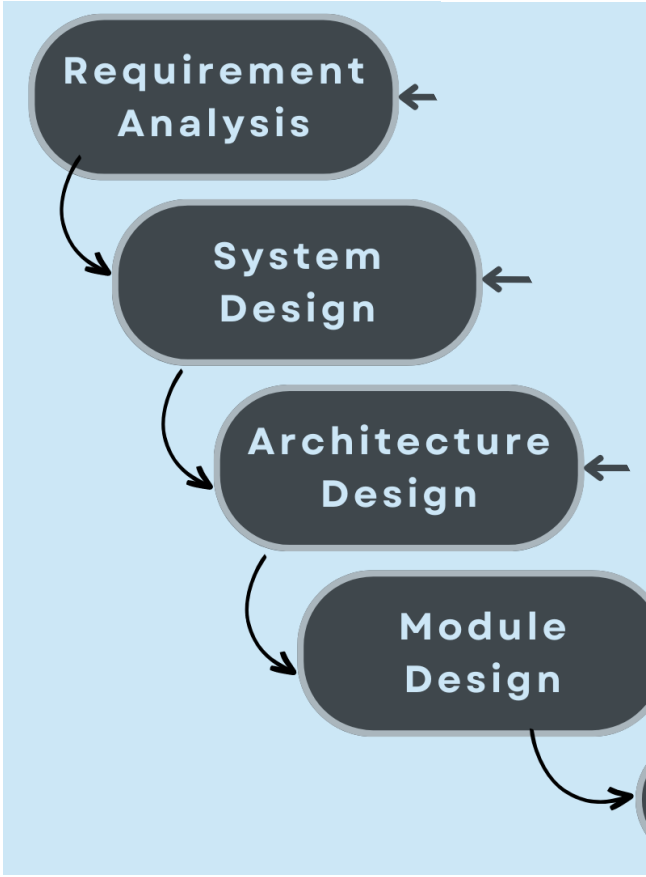
Transfer money



Lets build…

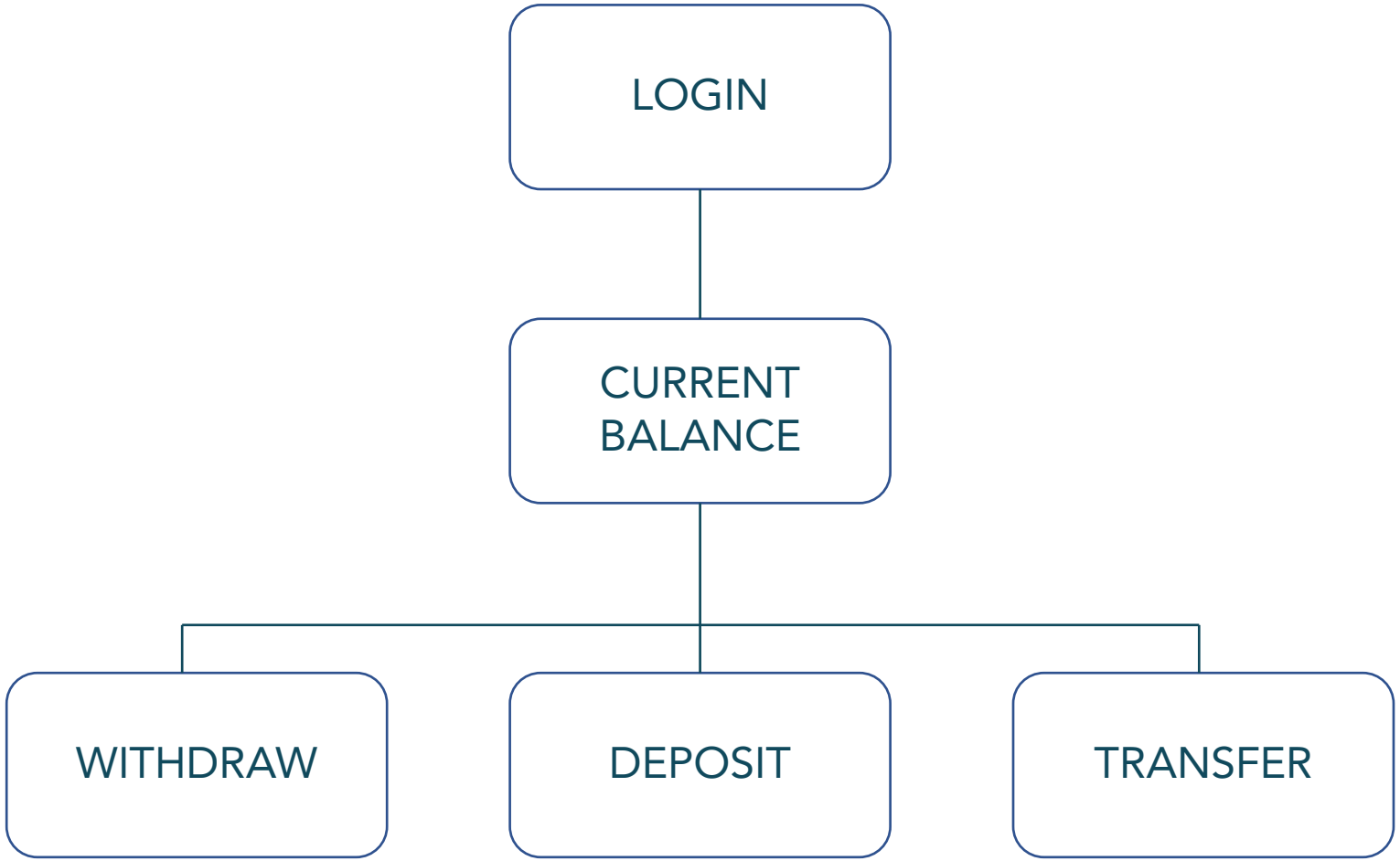
an online banking application

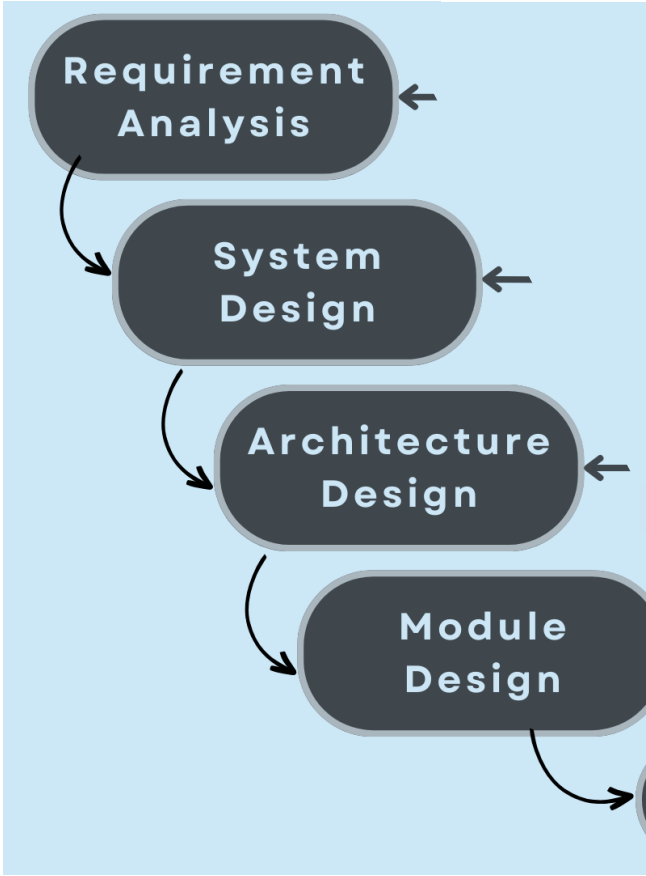




Lets build…

an online banking application

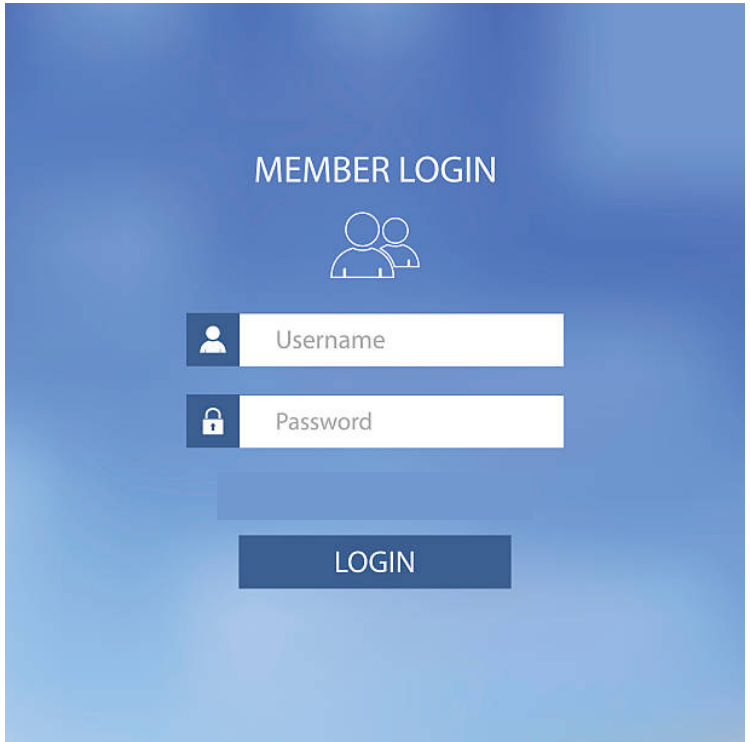


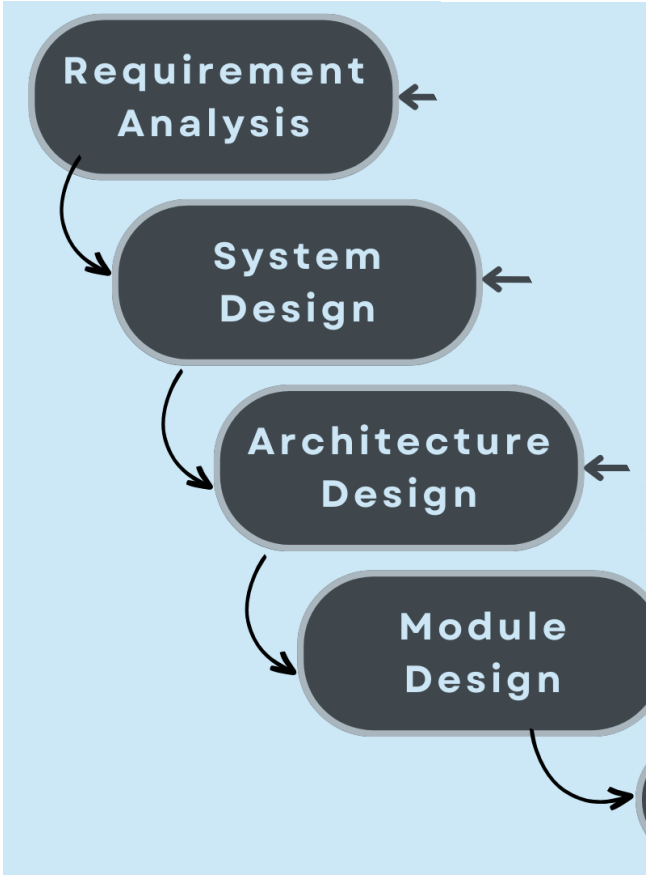


Lets build…

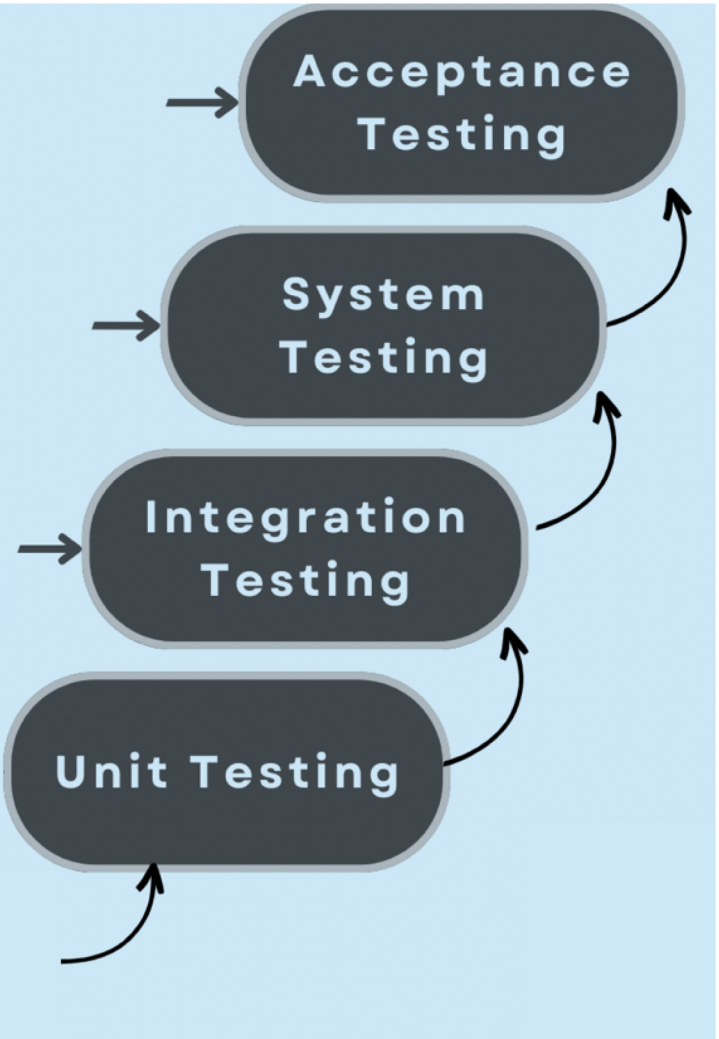
an online banking application

LOGIN





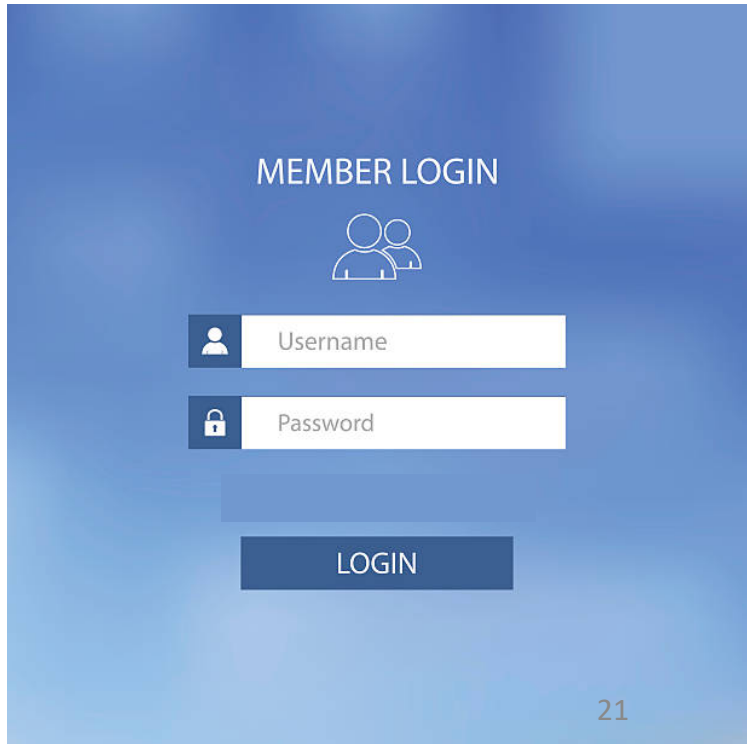
Lets test…
an online banking application



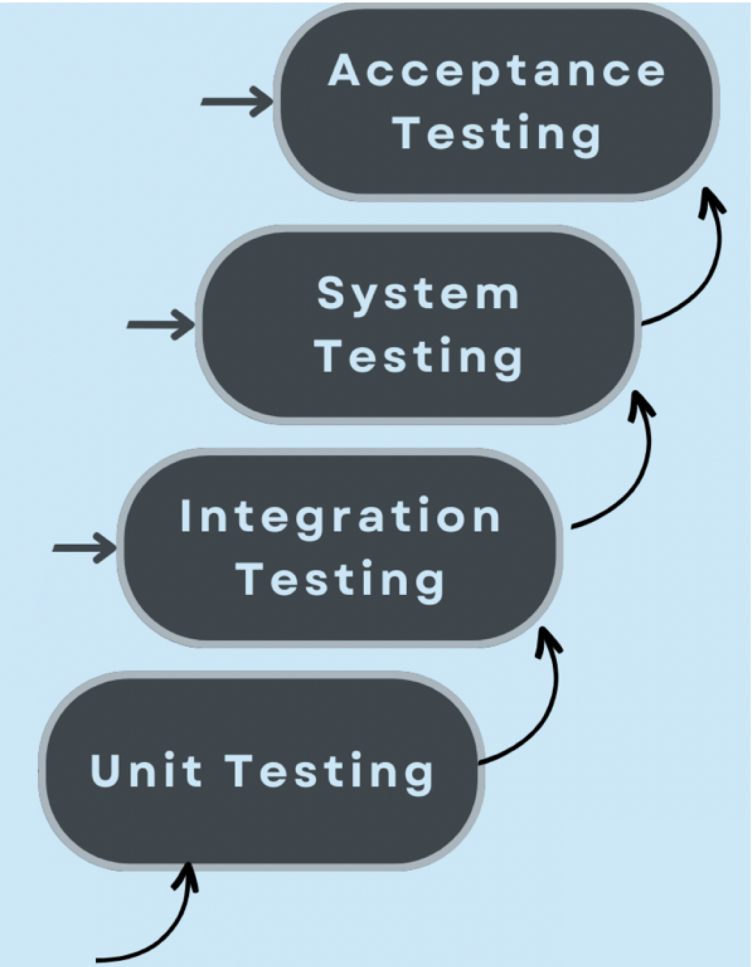
Unit testing
A technique where **individual components** or units of a software application are tested in isolation

- Enter valid username and password
- Enter invalid user name and password
- Click Login without entering credentials

LOGIN

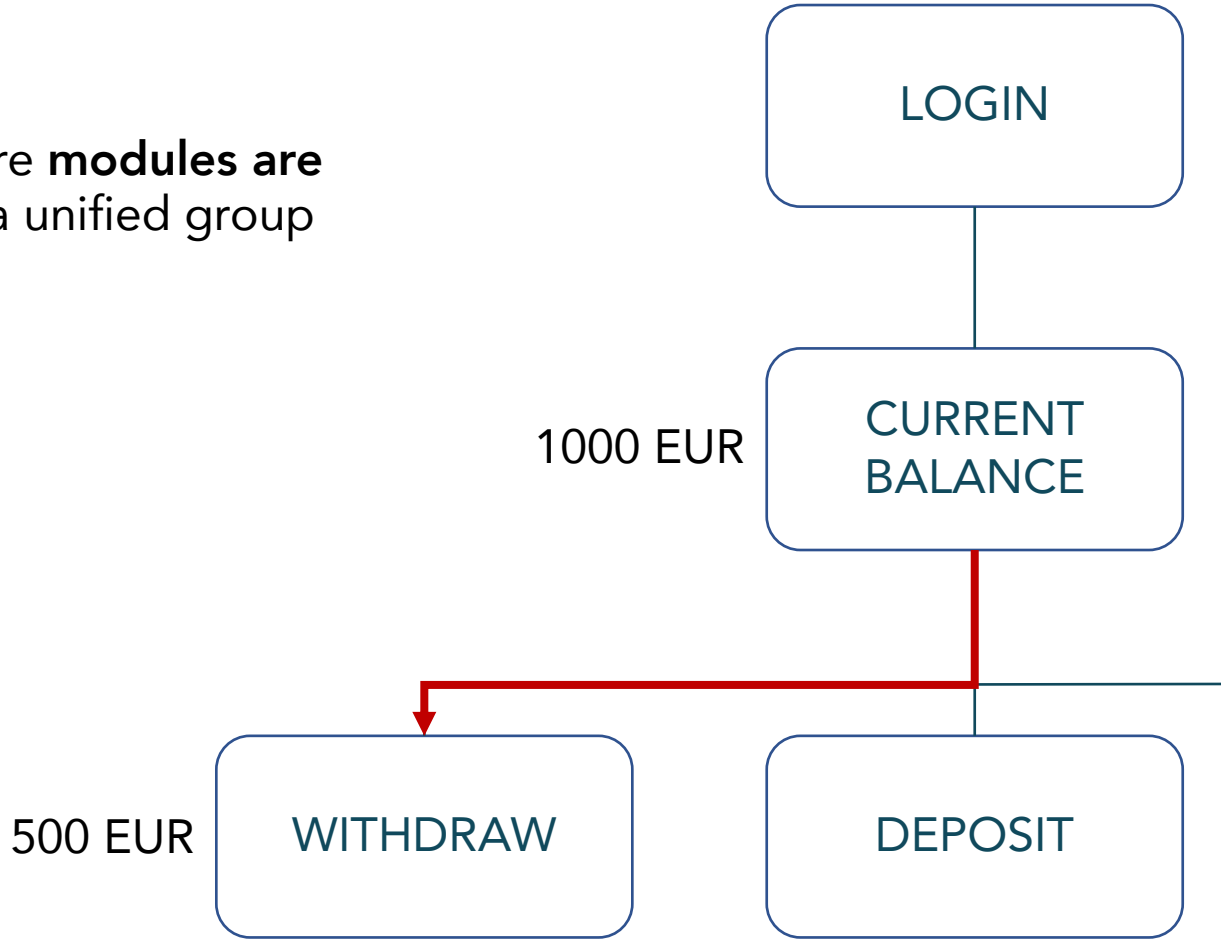


Lets test…
an online banking application

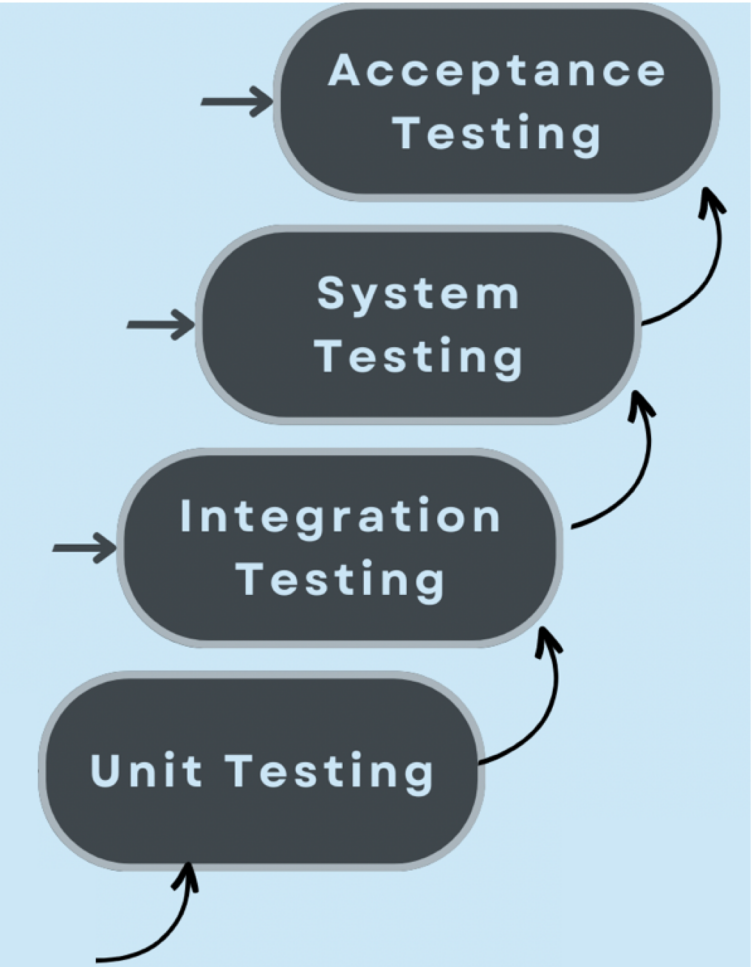


Integration testing

A technique where software **modules are integrated** and tested as a unified group



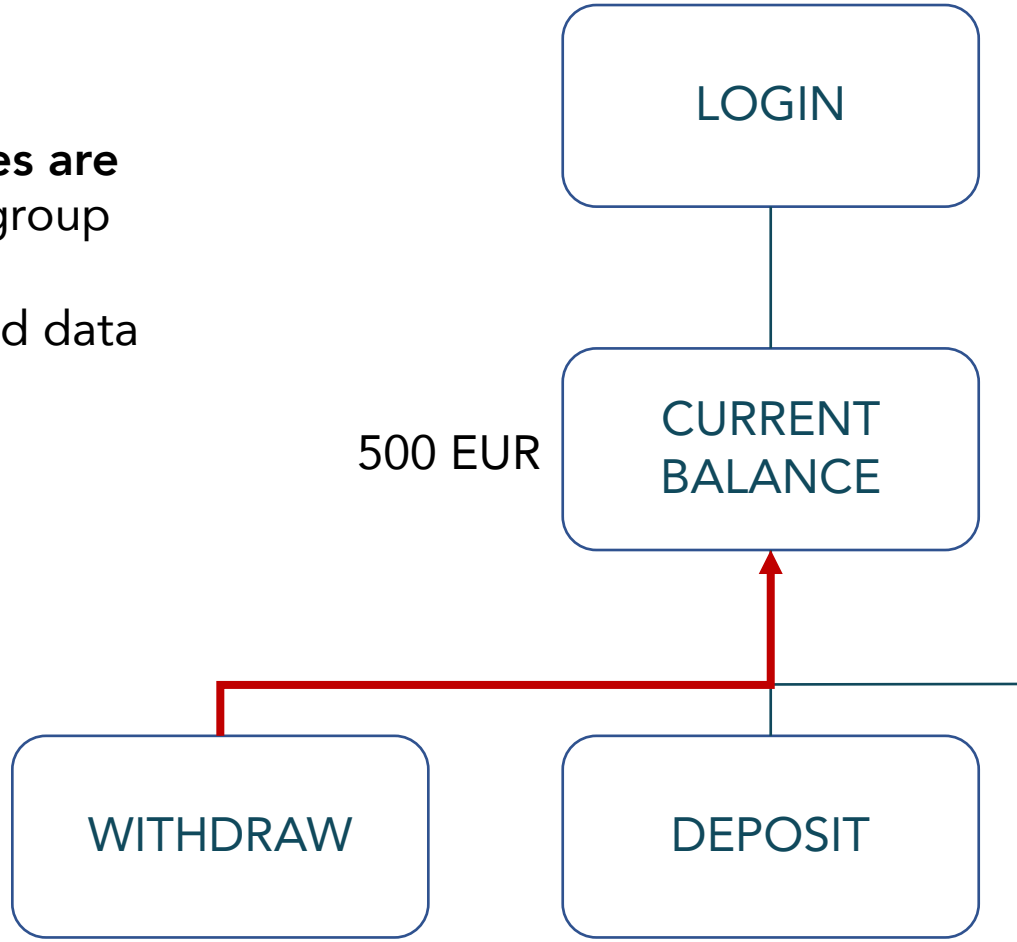
Lets test…
an online banking application



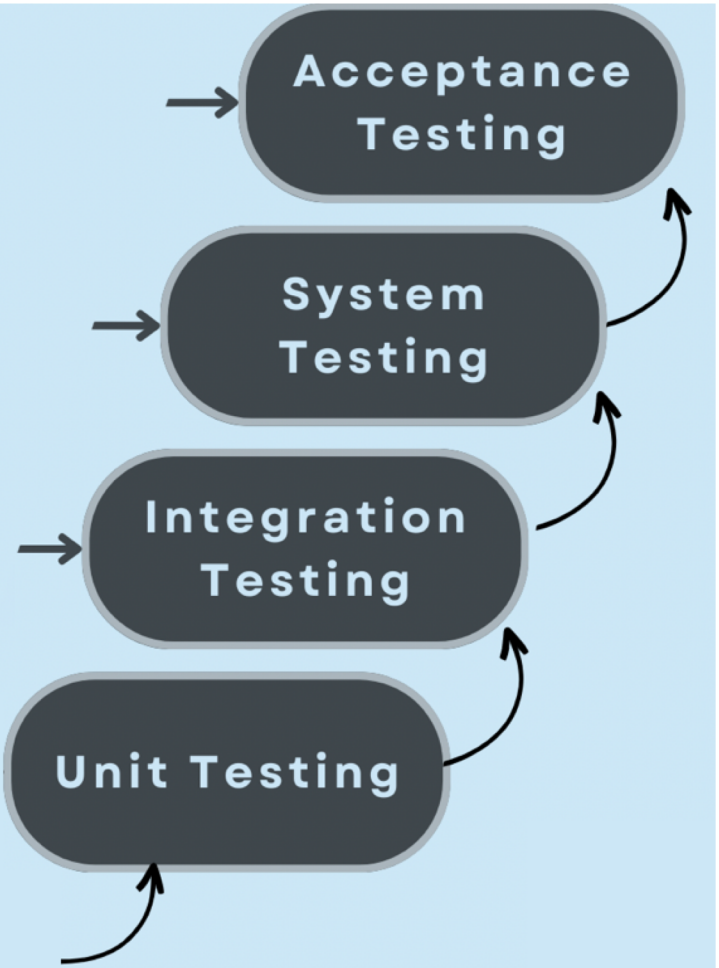
Integration testing

A technique where software **modules are integrated** and tested as a unified group

Checks for expected interactions and data transfer

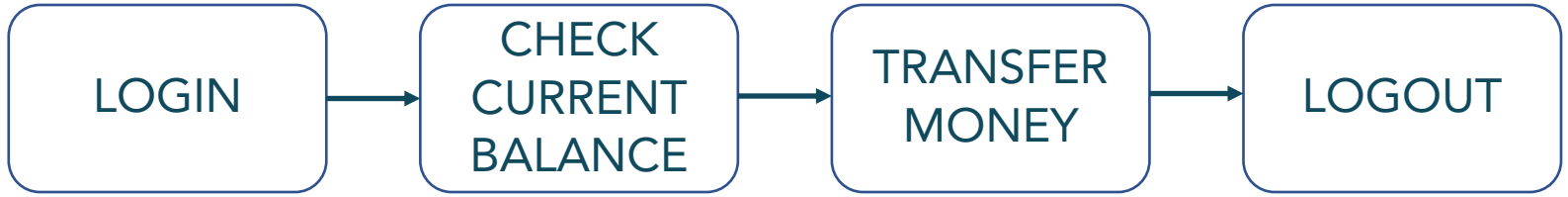


Lets test…
an online banking application

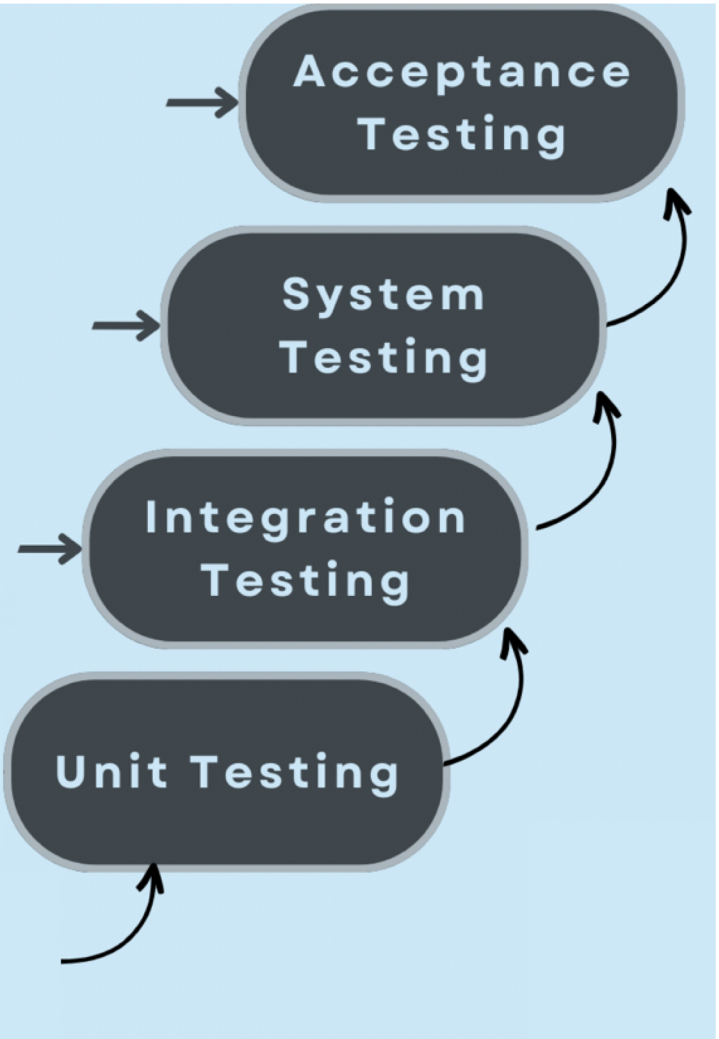


System (end-to-end) testing

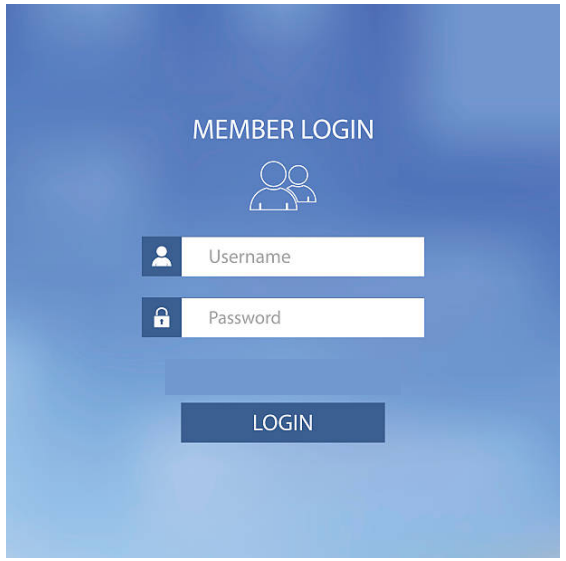
A technique to assess the ***complete functionality*** and performance of a fully integrated software system

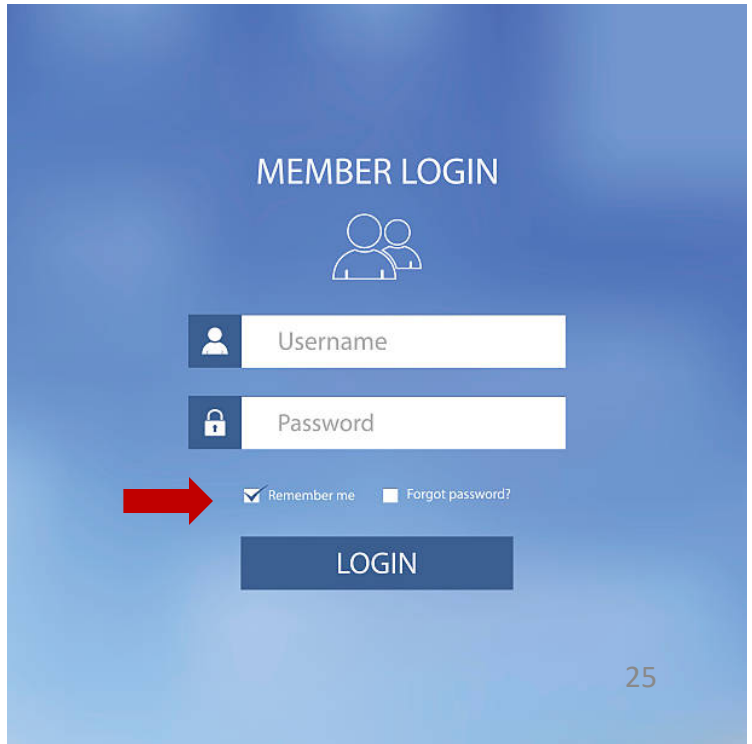


Lets test…
an online banking application

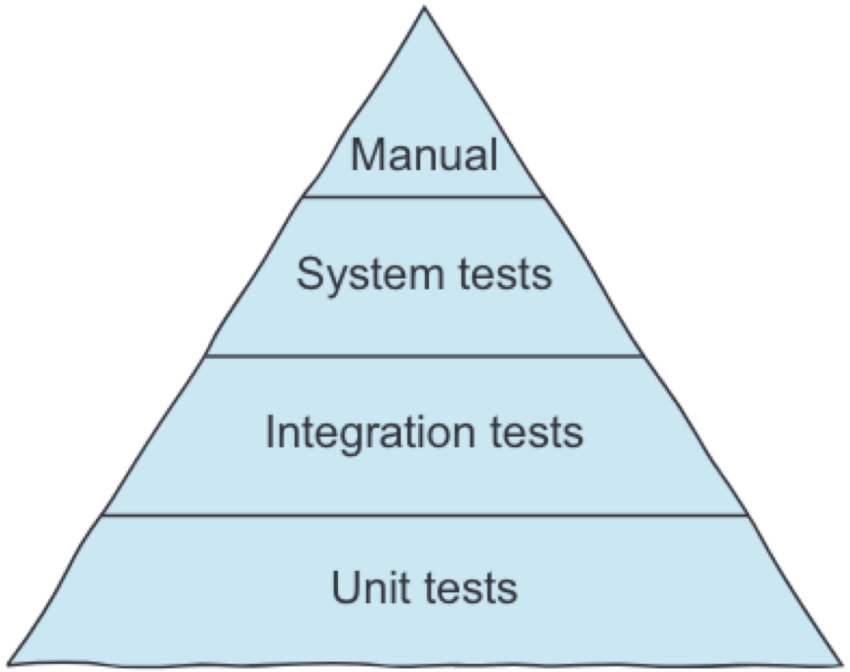


Acceptance testing
A process that ensures the software meets user needs and business requirements.





Levels of Testing



The Testing Pyramid

Unit tests with jUnit

- Can be used to test
  - an entire module or object
  - parts of an object (a method)
  - integration of modules and objects
- Each **test case** is embedded into a **test method**
- **Test classes** contain one or more test methods

Test case

A specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to achieve a particular software testing objective

Unit tests with jUnit

C.java

```
1  /** Contains only static methods. */
2  public class C {
3  
4      /** Return the min of b, c, and d. */
5      public static int min(int b, int c, int d) {
6          if (b <= c && b <= d) return b;
7          // min is either c or d
8          if (c <= d) return c;
9          return d;
10     }
11 
12     /** Return the max of b, c, and d. */
13     public static int max(int b, int c, int d) {
```

CTester.java

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3  
4  public class CTester {
5  
6      @Test
7      public void testMin() {
8          assertEquals(3, C.min(3, 5, 4));
9          assertEquals(2, C.min(3, 2, 5));
10         assertEquals(1, C.min(3, 2, 1));
11     }
12 
13 }
```

Unit tests with jUnit

What if

- we enter negative values?
- we enter positive and negative values?
- all inputs are the same?
- …

CTester.java

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class CTester {
5
6      @Test
7      public void testMin() {
8          assertEquals(3, C.min(3, 5, 4));
9          assertEquals(2, C.min(3, 2, 5));
10         assertEquals(1, C.min(3, 2, 1));
11     }
12
13 }
```

Unit tests with jUnit

assertEquals([message], expected, actual)

assertNull([message], object)

assertNotNull([message], object)

assertSame([message], expected, actual)

assertNotSame([message], expected, actual)

fail([message])

CTester.java

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class CTester {
5
6      @Test
7      public void testMin() {
8          assertEquals(3, C.min(3, 5, 4));
9          assertEquals(2, C.min(3, 2, 5));
10         assertEquals(1, C.min(3, 2, 1));
11     }
12
13 }
```

Annotations

- Great for test automation

- @Test
  - Annotates a test method

- @Before
- @After
  - Runs before/after every test

- @BeforeClass
- @AfterClass
  - Runs before/after all tests in the class (e.g. to connect or disconnect a data base)

Test-Driven Development (TDD)

"a technique for building software that **guides** software development by **writing tests**"

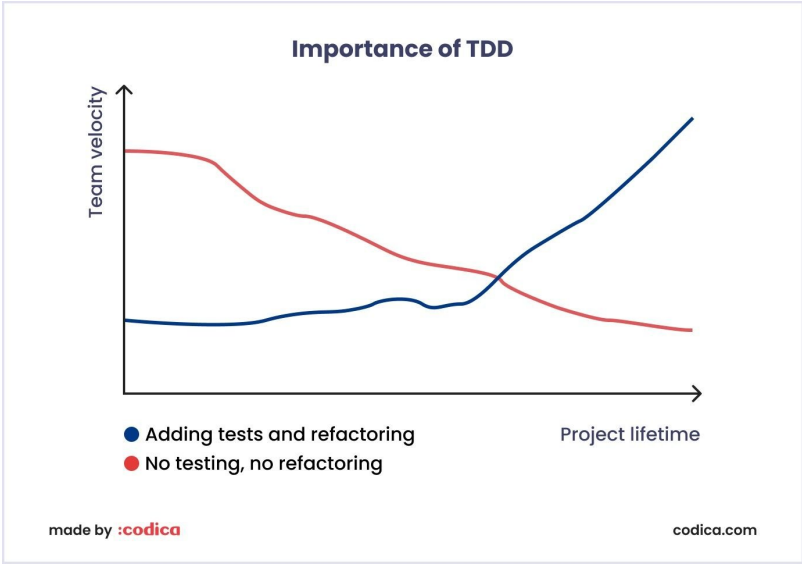
- Martin Fowler

- appropriate to use with unit tests

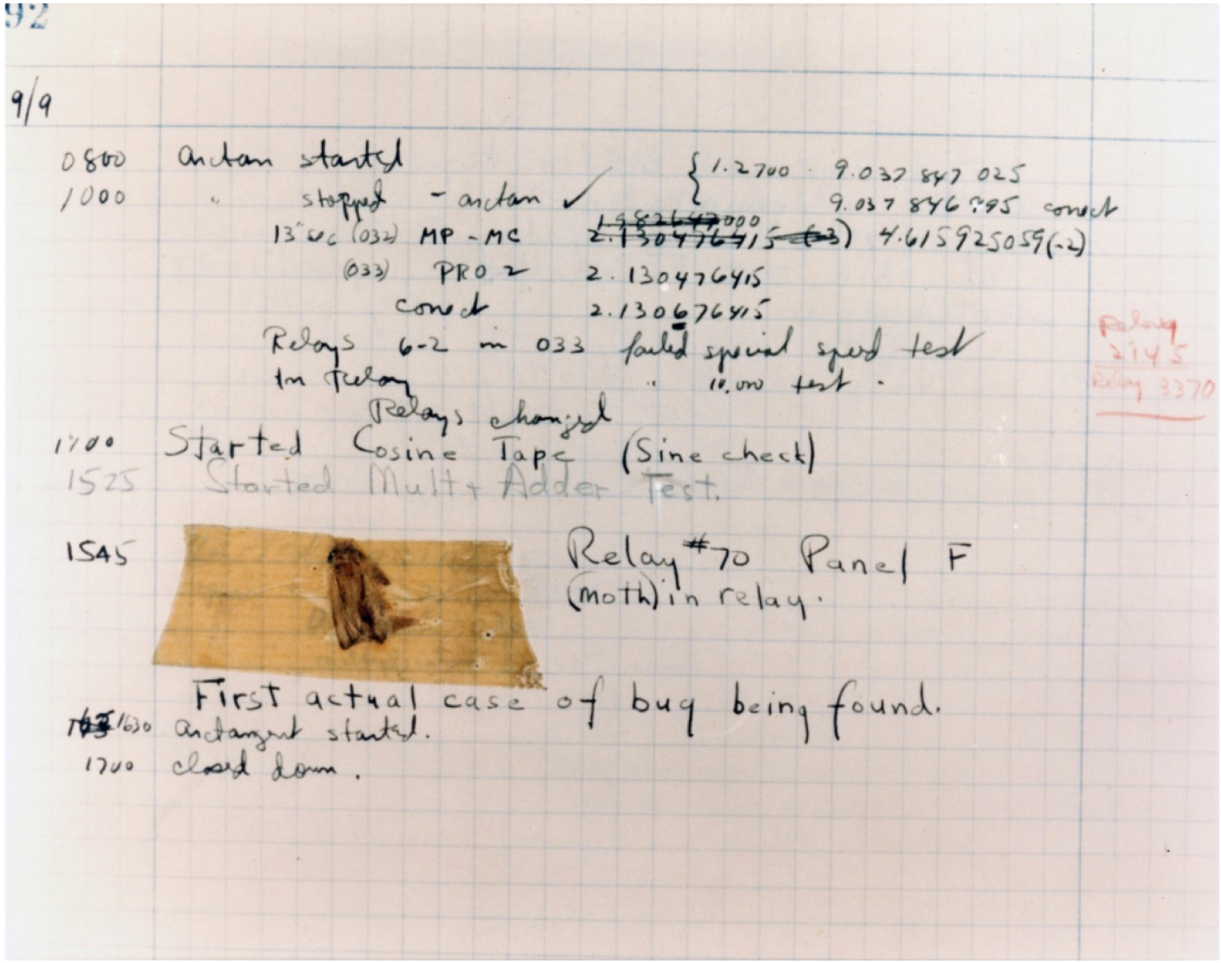
1. Add a test
   - representing a requirement or a goal functionality
2. Run all tests and see the new one fail
   - Support the need for the requirement and test case
3. Write code to make the test pass
   - Implement the new functionality
4. Run all tests and see them all succeed
   - Confirm the new functionality fulfills the expressed expectation
5. Refactor to improve the design
   - Improve the code and clean up
6. Repeat

Test-Driven Development (TDD)

- Helps to clarify requirements
- Aids to prevent introduction of bugs
- Improves test code coverage
- Facilitates usage of non-redundant, relevant test cases



We tested…


9/9
0800 Arctan started
1000 " stopped - arctan ✓
1.2700 9.037 847 025
9.037 846 795 correct
13" UC (032) MP - MC
(033) PRO 2 2.13047 6415
correct 2.13067 6415
Relays 6-2 in 033 failed special speed test
Relays changed
1700 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.
1700 closed down.

Testing process is in place to identify as many bugs as possible

- Bugs are:
  - Reported and documented
  - Fixed
  - Prevented from re-appearing in the future

Regression tests

A test ensuring that a fixed issue does not reappear in the future

Testing

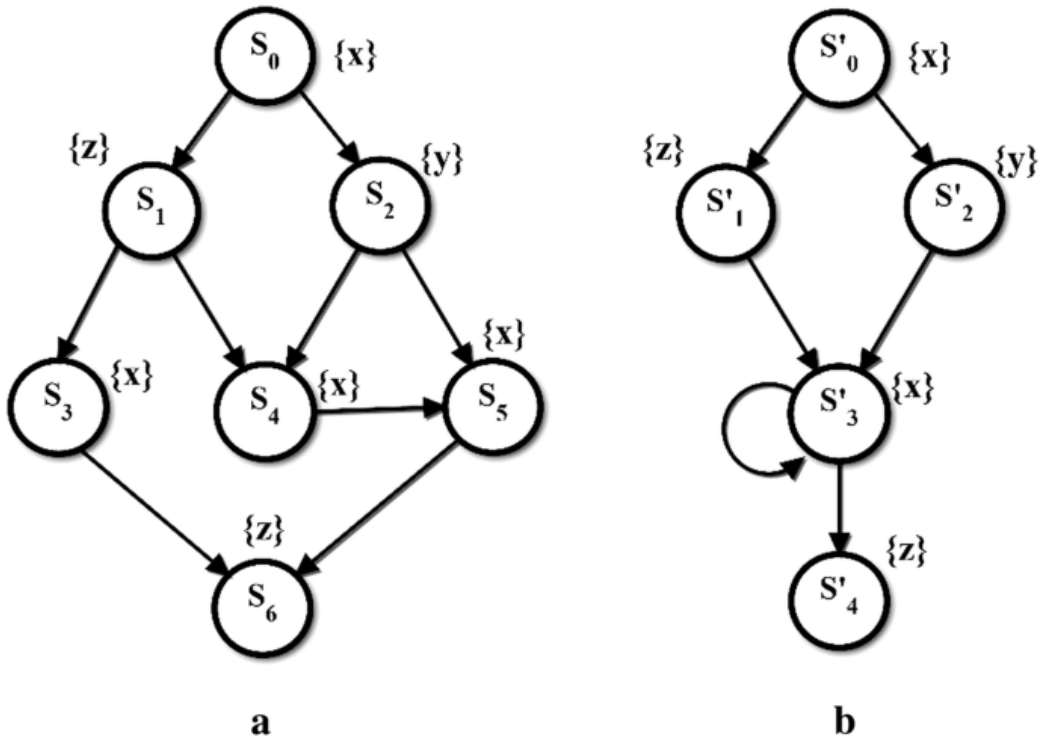
- Executes finite set of test cases on program
- Observes compliance/violation of specifications
- Focuses on test-case generation

Model Checking

- Formally describe possible program states in a mathematical model
- Abstracts standards for input or output formats into formal notation
- Prove compliance/violation of specification

Beyer, D., & Lemberger, T. (2017). Software Verification: Testing vs. Model Checking: A Comparative Evaluation of the State of the Art. In Hardware and Software: Verification and Testing: 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings 13 (pp. 99-114). Springer International Publishing.

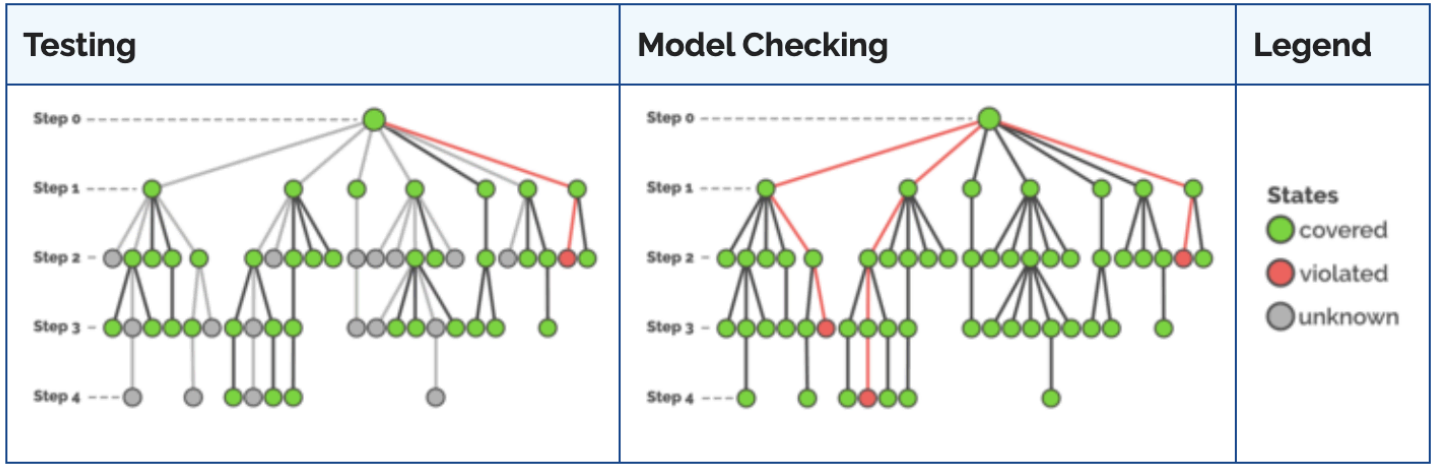
Model Checking



- Formally describes possible program states in a mathematical model
- Abstracts standards for input or output formats into formal notation
- Prove compliance/violation of specification

Beyer, D., & Lemberger, T. (2017). Software Verification: Testing vs. Model Checking: A Comparative Evaluation of the State of the Art. In Hardware and Software: Verification and Testing: 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings 13 (pp. 99-114). Springer International Publishing.

Model-Checking



- is complete
- can find more bugs
- in less time
- requires less adjustments to inputs

Beyer, D., & Lemberger, T. (2017). Software Verification: Testing vs. Model Checking: A Comparative Evaluation of the State of the Art. In Hardware and Software: Verification and Testing: 13th International Haifa Verification Conference, HVC 2017, Haifa, Israel, November 13-15, 2017, Proceedings 13 (pp. 99-114). Springer International Publishing.

Take-aways

1. Bugs can be costly and harmful
2. Testing allows to check your expectations against the actual behavior of your system
3. Testing is now integrated in every stage of software development and needs to satisfy the iterative and continuous nature of delivering software
4. Unit tests and Test-Driven Development are tools close to developers that can highly raise the confidence in your code and your development efficiency
5. There are other techniques beyond testing, like model-checking, to support the verification of your system