LDTS 2022/2023

Rui Maranhão

Today

- Test Automation
- Verification and Validation
  - Human Error, Fault, Error, Failure
  - Verification vs Validation

Test automation

During builds we run tests, so we need a tool that help us to automate testing

This will be discussed further in later classes

through the user interface

writing code in the main

DISADVANTAGES?

How can we test a system?

- Difficult to manage
  - how to set the state of the system before the test?
  - how to reset the effects of the test in the state after test execution?

- Expensive to repeat
  - need to interact again through the user interface
  - rewrite the main method for each new test case

So, we need a tool

- that stores the test cases

- that sets and cleans the context of tests

JUnit

Spock

Testing Frameworks

JUnit/Spock

The tester
focus on testing

The framework does the plumbing

- for each test
  - setup context
  - run the test
  - collect results
  - clean the context
- present results

```
@Before
public void setUp() throws Exception {
    bank = new Bank(“International”);
}

@Test
public void deposit() {
    Account euAccount = new Account(“João”, “EUR”);
    bank.addAccount(euAccount);

    euAccount.deposit(new Money(12, “EUR”));

    assertTrue(euAccount.getBalance().getValue() == 12);
    assertEquals(“EUR”, euAccount.getBalance().getCurrency());
}
```

setup setup

run

```
@Test(expected = IncorrectCurrencyException.class)
public void depositInADifferentCurrency() {
    Account swAccount = new Account(“Jean”, “CHF”);
    bank.addAccount(swAccount);

    swAccount.deposit(new Money(12, “EUR”));
}

@After
public void tearDown() {
    for (Account account : bank.getAccounts()) {
        account.remove();
    }
    bank.remove();
}
```

run

clean clean

```
def setup() throws Exception {
    bank = new Bank(“International”)
}

def ‘making a deposit’() {
    given: ‘an euros account'
    def euAccount = new Account(“João”, “EUR”)
    bank.addAccount(euAccount)
    when: ‘a deposit of 12 euros'
    euAccount.deposit(new Money(12, “EUR”))
    then: ‘the balance is incremented’
    euAccount.getBalance().getValue() == 12
    euAccount.getBalance().getCurrency() == “EUR”
}
```

setup setup

run

```
def ‘deposit different account’() {
    given: ‘a swiss francs account'
    def swAccount = new Account(“Jean”, “CHF”)
    bank.addAccount(swAccount)
    when: ‘a deposit of 12 euros'
    swAccount.deposit(new Money(12, “EUR”))
    then: ‘an exception is thrown'
    thrown(IncorrectCurrencyException)
}

def cleanup() {
    for (def account : bank.getAccounts()) {
        account.remove()
    }
    bank.remove()
}
```

run

clean clean

In the cases that an exception occurs we may want to verify that the state didn't change

```
@Test(expected = IncorrectCurrencyException.class)
public void deposit() {
    Account swAccount = new Account(“Jean”, “CHF”);
    bank.addAccount(swAccount);

    swAccount.deposit(new Money(12, “EUR”));
}
```

VS

```
@Test()
public void deposit() {
    Account swAccount = new Account(“Jean”, “CHF”);
    bank.addAccount(swAccount);

    try {
      swAccount.deposit(new Money(12, “EUR”));
      fail();
    } catch(IncorrectCurrencyException e) {
       assertEquals(0, swAccount.getBalance());
    }
}
```

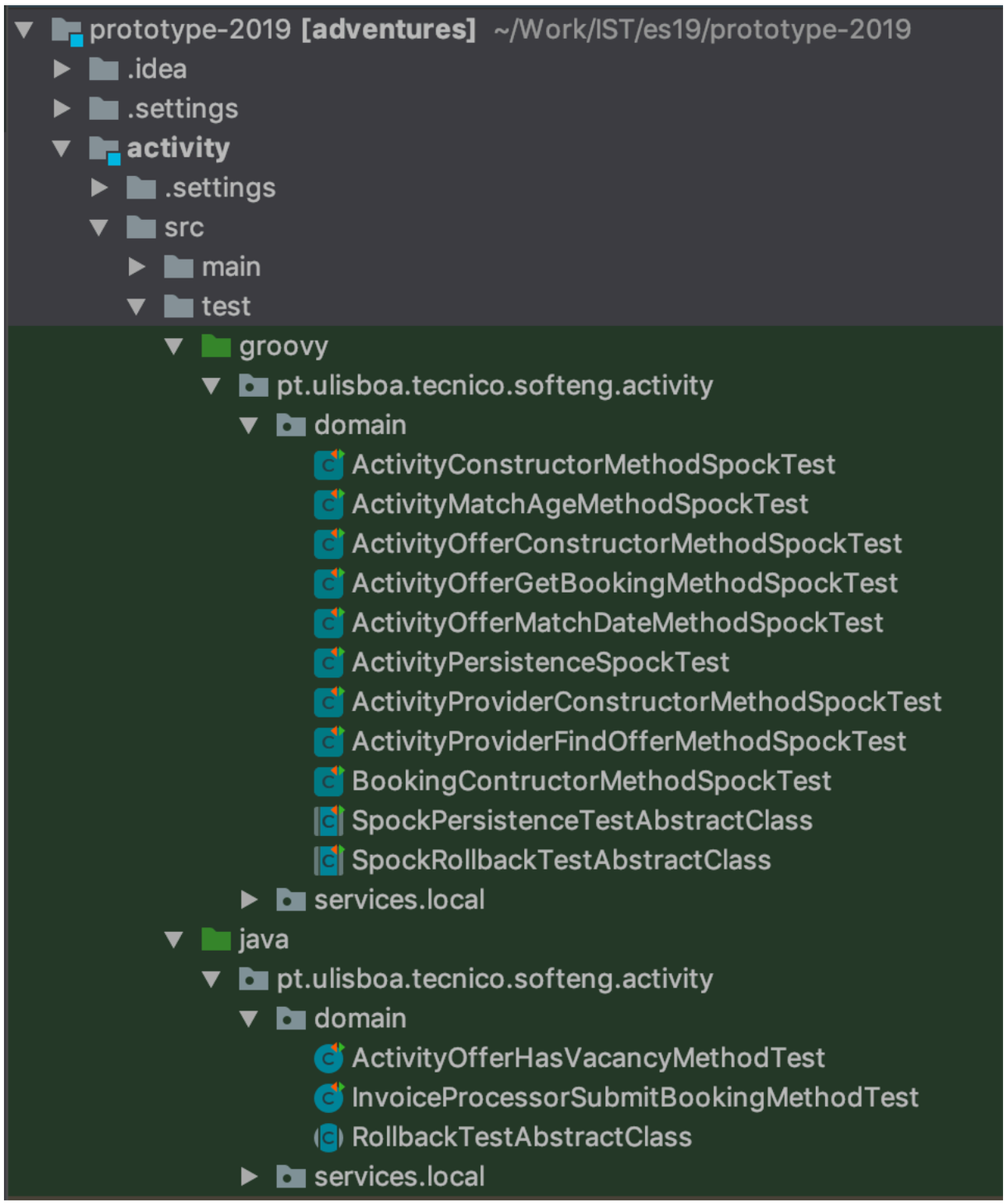
```
def ‘deposit different account’() {
    given: ‘a swiss francs account’
    def swAccount = new Account(“Jean”, “CHF”)
    bank.addAccount(swAccount)
    when: ‘a deposit of 12 euros’
    swAccount.deposit(new Money(12, “EUR”))
    then: ‘an exception is thrown'
    thrown(IncorrectCurrencyException)
}
```

VS

```
def ‘deposit different account’() {
    given: ‘a swiss francs account’
    def swAccount = new Account(“Jean”, “CHF”)
    bank.addAccount(swAccount)
    when: ‘a deposit of 12 euros’
    swAccount.deposit(new Money(12, “EUR”))
    then: ‘an exception is thrown'
    def error = thrown(IncorrectCurrencyException)
    and: ‘the balance is not changed’
    swAccount.getBalance() == 0
}
```

Tests in the project

two different trees that share the same naming structure



Software Testing

To Err is Human

Software has bugs!

Why?

To err is human

Humans write software

Computers
execute software

Human Error

creates

Fault

generates

Error

...

generates more errors

Failure

Identify faults

DISTANCE

Time since it was created

Number of errors before failure

Failure

↓

… identify

↓

Fault

↓ cause

Human Error

↓ infer more

Fault

Identify the cause of bugs

It helps to identify other faults that have the same cause

- Human Error - an human action which produces software faults

- Fault - an omission, a defect, in the software cause of a human error

- Error - an unexpected change in the system behaviour caused by a fault

- Failure - an observable error

Is it a bug

or a feature ?

Did we build the right product?

Did we build the product right ?

back to basics...

Software is language

How can we write the right product ?

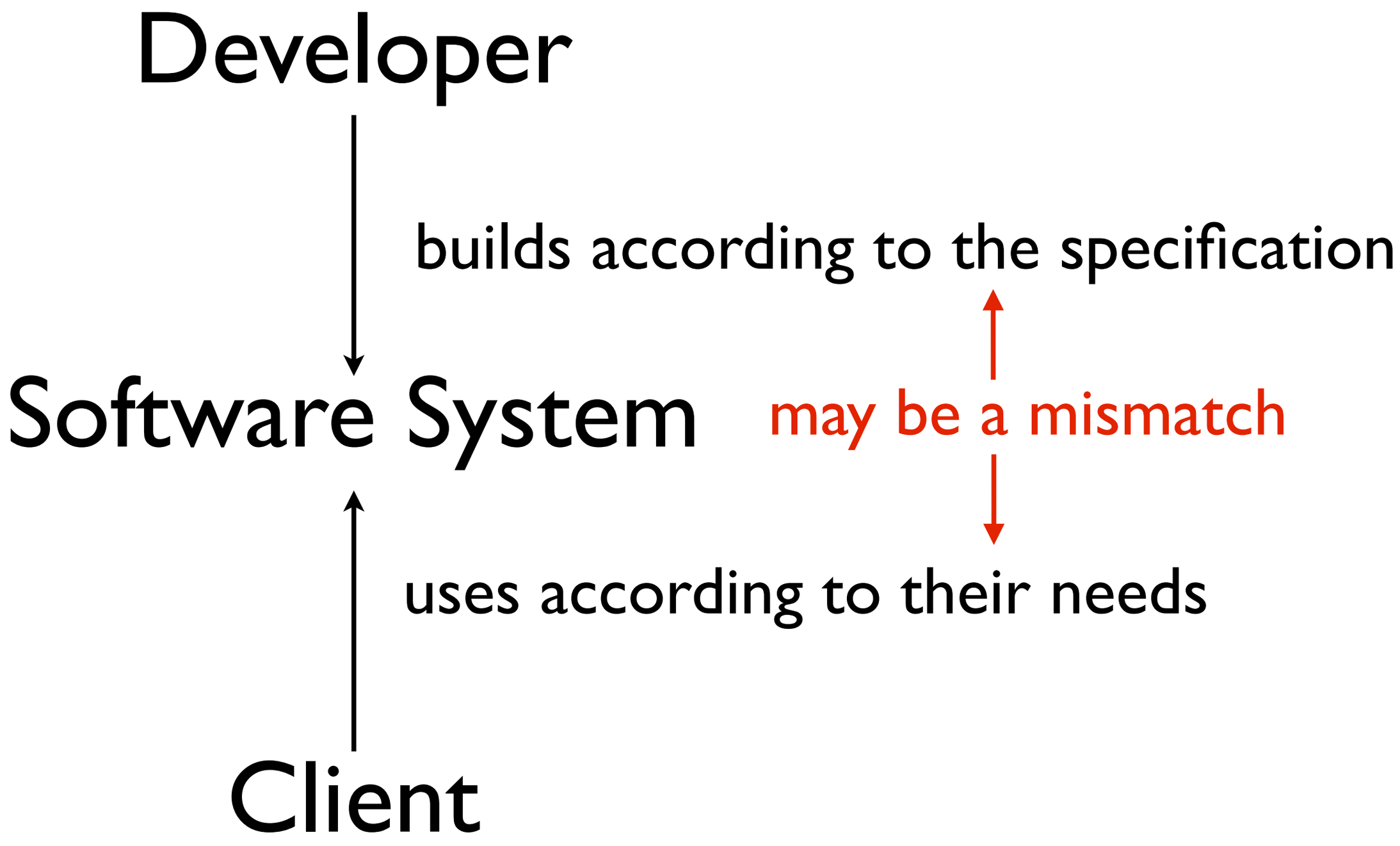
problem specification

vs

client needs

Verification - we did it right, according to the specification

Validation - we did it according to the client needs


Developer
builds according to the specification
Software System
may be a mismatch
uses according to their needs
Client

Software System

tests according
to the specification

tests according
to their needs

Developer
(tester)

Client

There is a mismatch between the problem specification and the client needs

language transformations

How can we solve this mismatch?

Work with the client

there is no problem specification

less transformations

Is it always possible?

if we do not have a client in the team

Have precise specifications

small transformations

So… is it a bug
or a feature ?

What can I test ?

source
code

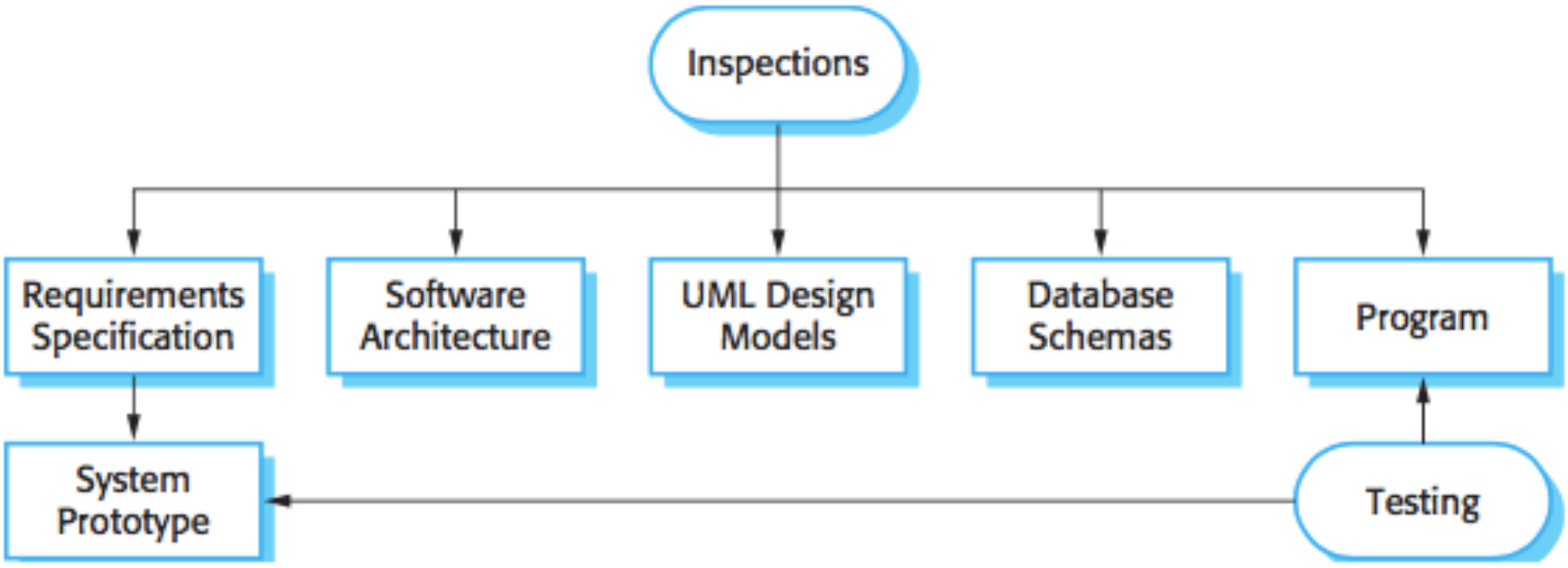
req

models

Tests, reviews, inspections, ...

ents

solution
specifications


Inspections
Requirements Specification
Software Architecture
UML Design Models
Database Schemas
Program
System Prototype
Testing

All software development artefacts!

How do I test ?

As a mathematician

Prove that the software is correct

It is driven by the specification

As an accountant

Inspect the software

It is driven by experience and standards

as an engineer

run the software

“enough” times

When is “enough”, enough ?

Both, static (reviews and inspections) and dynamic (tests) verification and validation is important!

When should we apply each technique?

Reviews and Inspections

can be applied to all descriptions

humans are good at reading descriptions

Tests

runtime behaviour of the system

computers are good at executing descriptions