TALL SOM FORTELLER

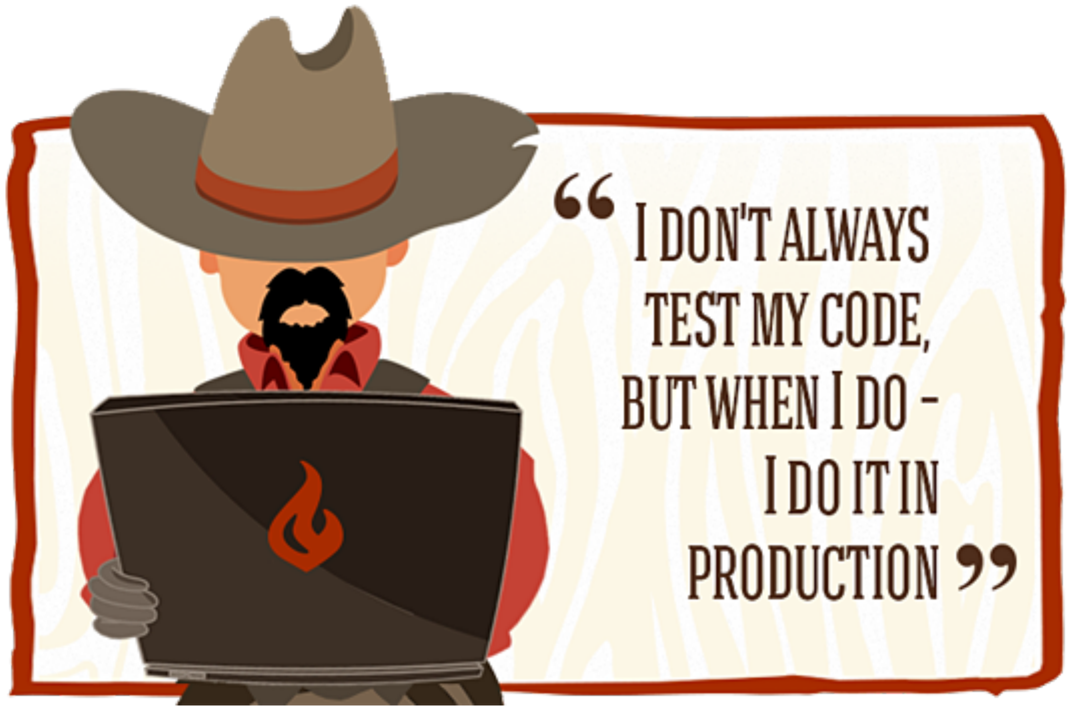
Unit Testing with JUnit 5 and Mockito

Statistisk sentralbyrå
Statistics Norway

Agenda

1. What is Unit Testing?
2. Why we need to do it?
3. How can we do it?
4. Practise with JUnit 5.
5. Mockito.

How developers test their code?


"I DON'T ALWAYS TEST MY CODE, BUT WHEN I DO - I DO IT IN PRODUCTION"

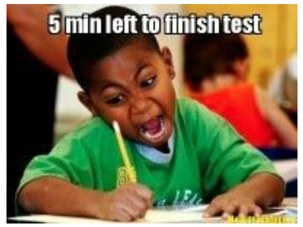
Normal Development Process

Developers are always quiet confident that code will work in production

Testing team takes the challenge to test the code

Then testing team a bug in the release

Deployment gets cancelled due to need for bug fix.



After bug fix, due to release pressure don't perform the regression testing

Testers always gets less time due to delay in development process

After some manually testing change gets deployed in production

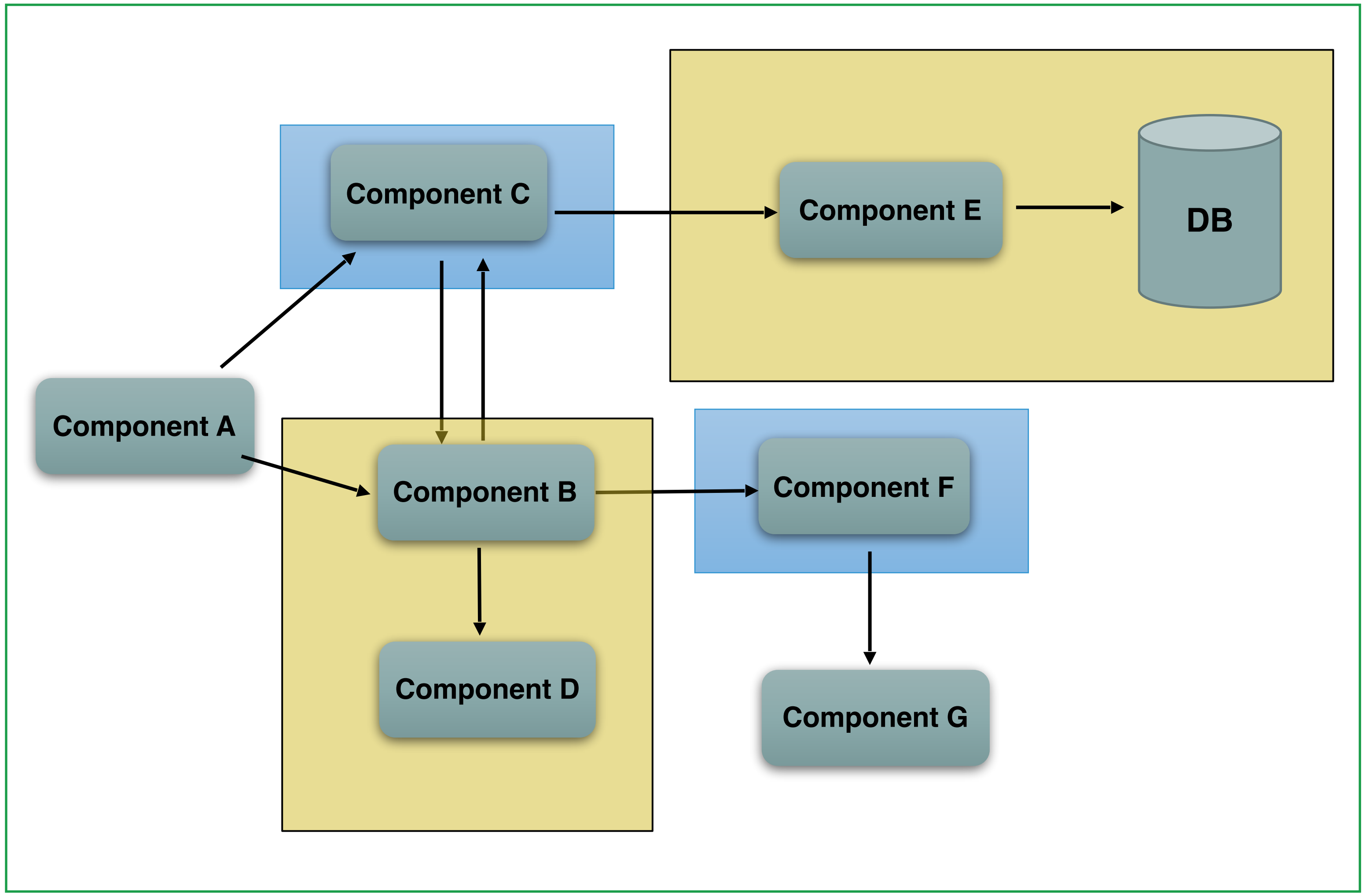
Developers get relaxed

Statistisk sentralbyrå
Statistics Norway

What is Unit Testing ?

- Software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.[1]

Statistisk sentralbyrå
Statistics Norway



🟦 Unit test 🟨 Integration test ⬜ Acceptance test

Statistisk sentralbyrå
Statistics Norway

What is a Unit Test?

- Unit test is a specification, it defines the behavior.
- It is a method marked with annotation (**@Test** for JUnit)
- The test usually calls a public method of the class during the test and checks if the result is as expected.
- All dependencies to the class during the test have been replaced by test doubles (stubs, mocks etc.)

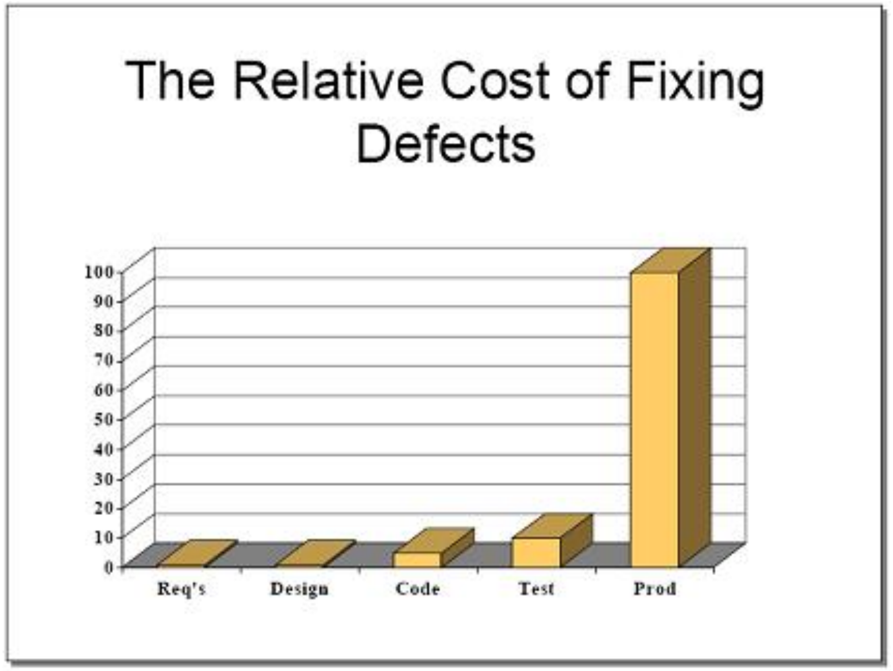
What is a Unit Test?

- Which methods are to be unit tested will be assessed on the basis of the complexity / usefulness.
- Each unit test should be independent of other unit tests.

Why Unit Testing ?

```
Assert.AreEqual(
     GetTimeOfDay(),
     "Morning" );
```

1. The cost of fixing errors in production can be dramatically higher than the cost of fixing them in development.

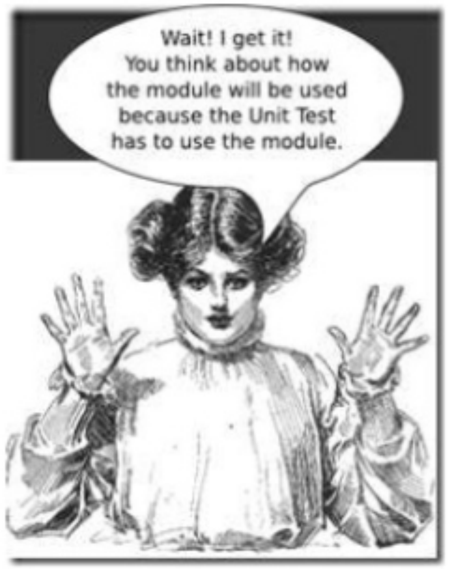


Statistisk sentralbyrå
Statistics Norway

2. Helps to write decoupled, coherent units -> easier to change afterwards.

More focus on how the class can be used by others (API)

- Elements are coupled if a change in one forces a change in the other. “Loosely” coupled features (i.e., those with low coupling) are easier to maintain.
- An element’s **cohesion** is a measure of whether its responsibilities form a meaningful unit.


Wait! I get it! You think about how the module will be used because the Unit Test has to use the module.

Statistisk sentralbyrå
Statistics Norway

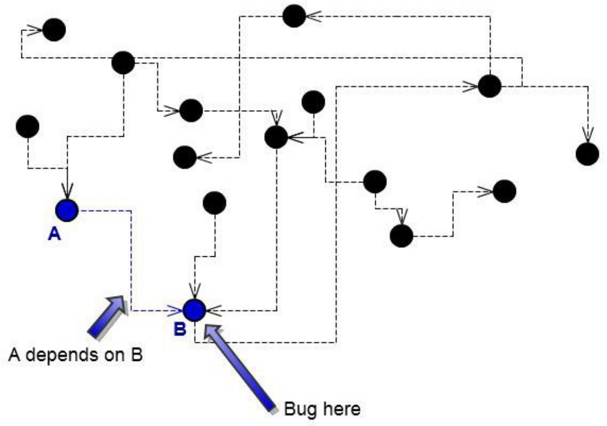
3. Easy and quick check if the class does what it should. Make the coding process more flexible.

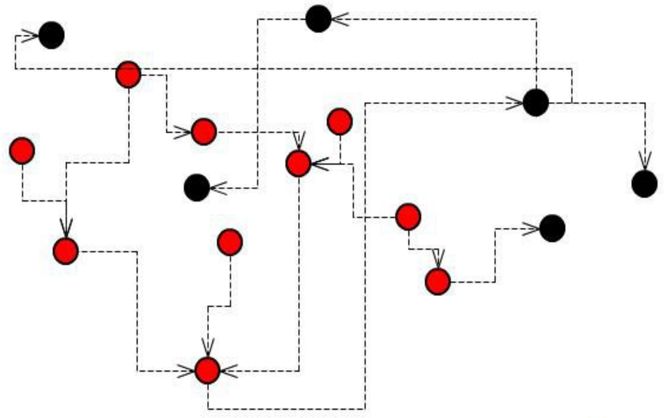
4. Driving unit tests can be easily automated(Jenkins).

6. You use the unit test to confirm and then fix a bug.

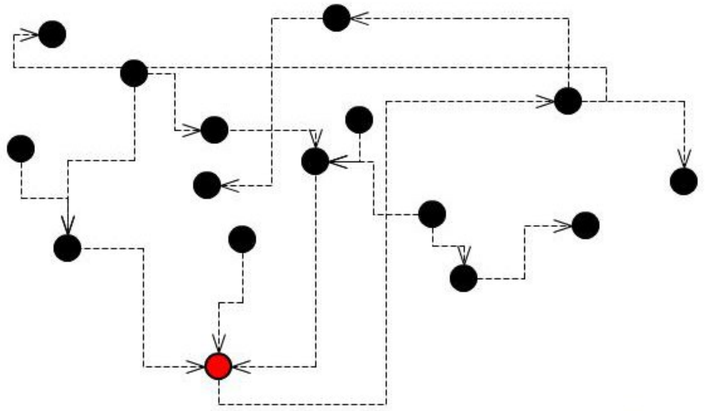
7. If a test fails, only the latest changes made in the code must be debugged.

8. Easy debugging





With Integration testing



With Unit testing

9. Refactoring does not create fear anymore that something breaks.

10. Developers feel safe.

11. Easier to understand what the purpose behind the class was, serves as documentation of the desired behavior.

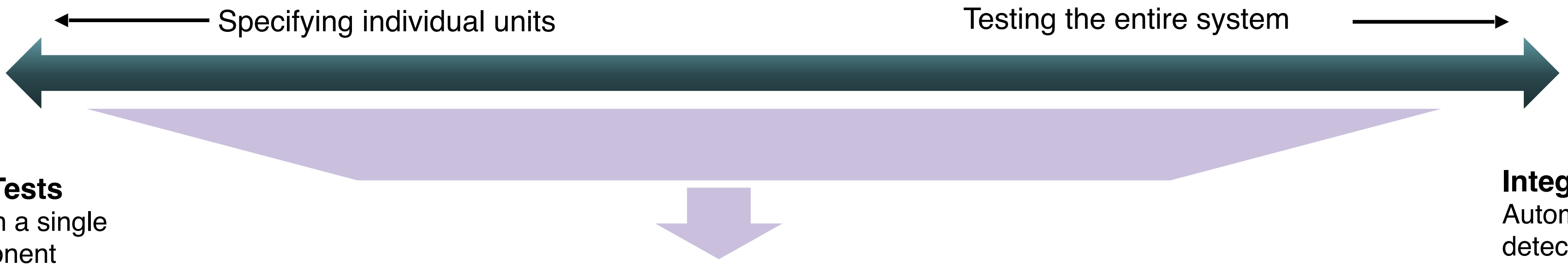
YOU ENSURE READABILITY

“Some” drawbacks with Unit testing

Drawbacks with Unit Testing

- Takes more development to write production code + unit test code.
- The employer must be aware of this.
- Bad tests that run green can give false impression.
- Test the core logic, do not decorate.
- Code coverage tool helps to see which code is covered by unit tests.
- Write the minimum number of tests that specify behavior.
- Will not capture errors at higher levels (integration, system).
- Difficult to test multi-threaded code.

Where we exist now ?



Unit Tests
Design a single component

Integration Tests
Automate entire system to detect regressions

Dirty Hybrids

Unclear goals. High maintenance, don't prove much. Most common in our systems.

How to write a good Unit Test?

✓Make each test independent of every other test.

✓Simple behavior should be specified in one and one test.

✓Don't make unnecessary assertions.

✓Unit tests are a design specification of how a particular behavior should work, not a list of observations of all the code does.

✓Only test one code unit at a time.

✓Your architecture must support test units (ie classes or very small groups of classes) independently, not all linked.

✓Mock out all external services and state.

How to write a good Unit Test

- ✓ Avoid testing only "happy path”.
- ✓ Test limit values and out-of-domain values as well.
- ✓ Avoid Testing Simple Logic (toString ()).
- ✓ Avoid overly complicated tests, hard to tell if the test is correct.
- ✓ Avoid lots of small tests, attempts to test cover all "execution paths”.
- ✓ Avoid mocking types you do not own (third party).
- ✓ Create wrapper around external library / system.

Structure of a Unit Test (AAA)

- Set Up
- Declare expected results.

Arrange

- Execute the desired method during the test.
- Get actual result.

Act

- Verify actual results matches the expected results (**Assert**)

Let’s begin with JUnit

https://github.com/statisticsnorway/workshop-unit-testing

Mockito

https://git-adm.ssb.no/projects/SAN/repos/junit-workshop/browse/src/test/java/solution/mockito

TALL SOM FORTELLER

Takk!

Statistisk sentralbyrå
Statistics Norway