

EMS30 Week 2 Les 2: Unittesten

Leerdoelen week 2 les 2. Je leert hoe je:

- C-code systematisch kan testen met behulp van het eenvoudige test-framework **Catch2**;
- tests automatisch na elke commit kan laten uitvoeren, door gebruik te maken van Bitbucket **pipelines**;
- test-**stubs** kunt gebruiken om software die bedoeld is voor een embedded systeem, toch zo veel mogelijk op een pc te kunnen testen.

Waarom testen?

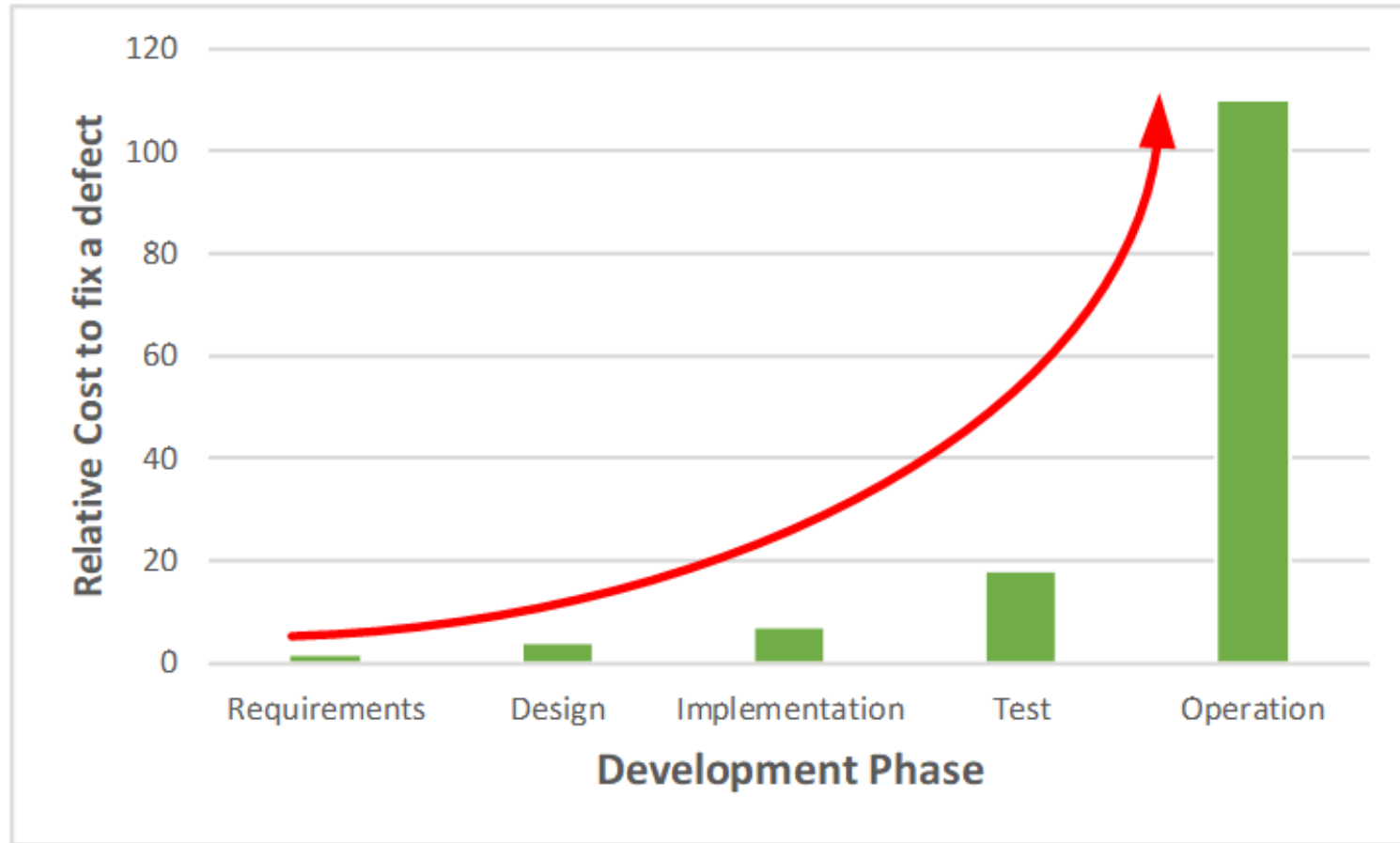
Foutvrije software ontwikkelen is erg moeilijk misschien zelfs onmogelijk.

Spraakmakende fouten:

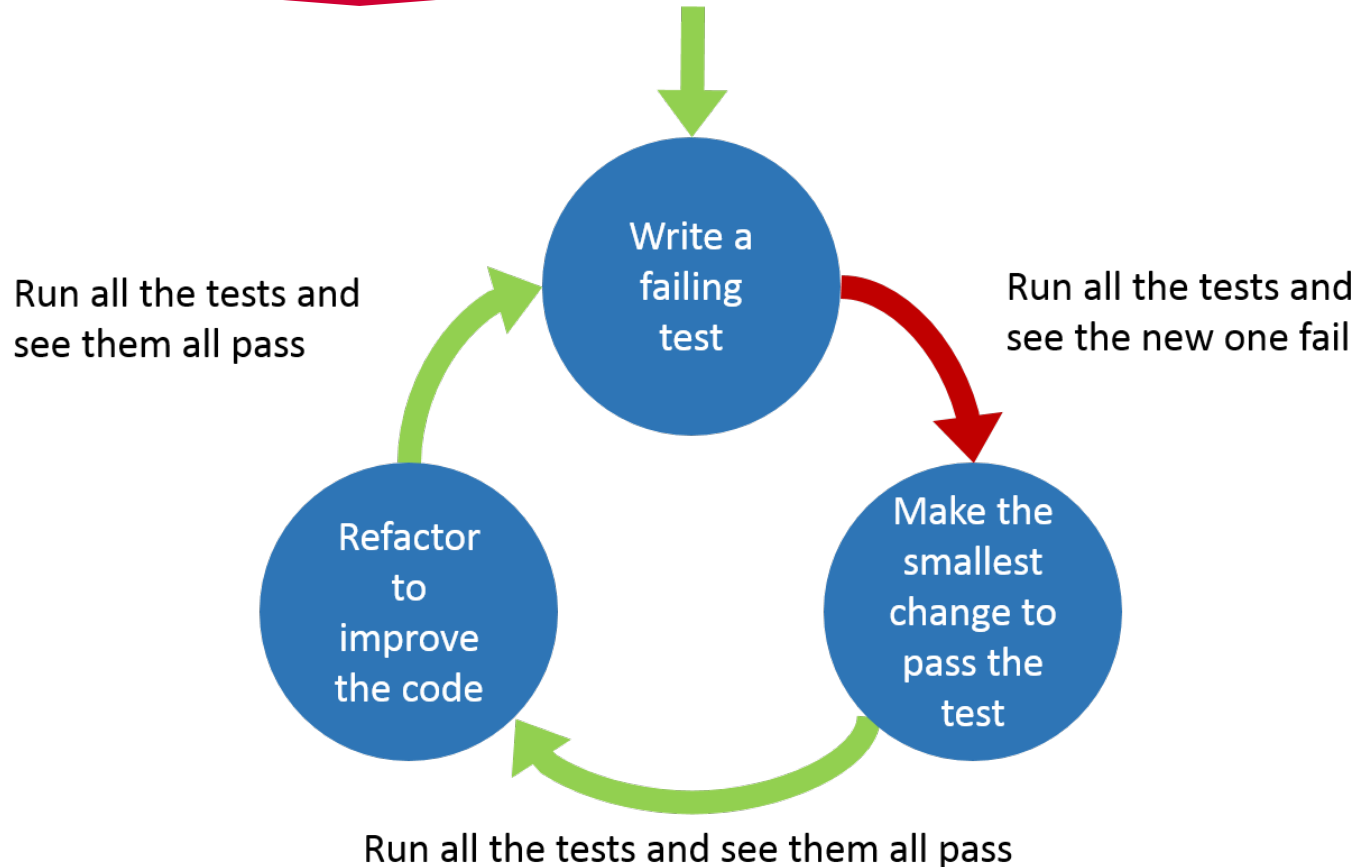
- 1996 – [Ariana 5 raket](#)
- 2013 – [Toyota versneld als op de rem wordt geduwd](#)
- 2018 – [Boeing 737 MAX 8 stort neer](#)
- 2019 – [Metro Hoekse Lijn 2 jaar vertraagd en kost 175 million Euro extra](#)

Nog veel meer voorbeelden: <https://www5.in.tum.de/~huckle/bugse.html>

Waarom unit's testen



Test Driven Development (TDD)

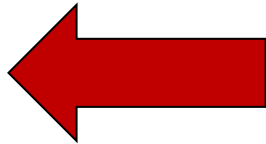


Bron plaatje: <https://hiddeninplainsight.co.uk/post/embedded-tdd/>

https://en.wikipedia.org/wiki/Test-driven_development

Er zijn veel test frameworks beschikbaar die het schrijven en uitvoeren van unittests vereenvoudigen:

- AceUnit
- CUnit
- Unity
- Embunit
- CMock
- Catch2



<https://github.com/catchorg/Catch2/blob/devel/docs/Readme.md>

Zie: https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks

Voorbeeld test module breuk

EMBEDDED SYSTEMS

test_breuk.cpp

```
#define CATCH_CONFIG_MAIN
#include <catch2/catch.hpp>
// Include C code in C++ bestand
extern "C"
{
    #include "breuk.h"
}

TEST_CASE("rekenen met breuken", "[breuken]")
{
    // Init voor meerdere sections
    Breuk a = {1, 3}, b = {2, 4};

    // Als init niet werkt heeft verder testen geen zin
    REQUIRE(a.teller == 1);
    REQUIRE(a.noemer == 3);
    REQUIRE(b.teller == 2);
    REQUIRE(b.noemer == 4);

    SECTION("optellen"){
        // Add
        Breuk c = add(a, b);
        // Check resultaat
        CHECK(c.teller == 5);
        CHECK(c.noemer == 6);
    }
    // ...
}
```

Voorbeeld test module breuk

EMBEDDED SYSTEMS

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18)
project(breuk_test)

include(CTest)
find_package(Catch2 REQUIRED)
include(Catch)

# test_breuk moet een C++ programma zijn
add_executable(test_breuk test_breuk.cpp breuk.c)
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -std=c18 -Wall -Wextra -Wpedantic -g3 -O0")
target_link_libraries(test_breuk PRIVATE Catch2::Catch2)

catch_discover_tests(test_breuk)
```

Zie: https://bitbucket.org/HR_ELEKTRO/ems30/raw/master/Programmas/breuk-test.zip

Voorbeeld test module breuk

test_breuk is a Catch v2.13.9 host application.

Run with -? for options

rekenen met breuken

vermenigvuldigen

/home/ems/Voorbeelden/breuk-test/test_breuk.cpp:29
.....

/home/ems/Voorbeelden/breuk-test/test_breuk.cpp:33: FAILED:

CHECK(c.teller == 1)

with expansion:

2 == 1

/home/ems/Voorbeelden/breuk-test/test_breuk.cpp:34: FAILED:

CHECK(c.noemer == 6)

with expansion:

12 == 6

=====
test cases: 1 | 0 passed | 1 failed

assertions: 20 | 17 passed | 3 failed

Zie: [live demo](#)

Continuous Integration (CI)

Unittests kunnen automatisch worden uitgevoerd bij een commit m.b.v. bitbucket pipelines.

The screenshot displays the Bitbucket Pipelines interface. On the left, a sidebar lists navigation options: Source, Commits, Branches, Pull requests, Pipelines (highlighted), Deployments, Issues, Jira issues, Security, Wiki, Downloads, and Repository settings. The main content area features a purple banner with a message: "We're offering 500 build minutes per month for your current plan. See upgrade options". Below this is a large blue icon representing a container with code symbols. The text "Build, test and deploy with Pipelines" is followed by a description: "Easy to set up Integrated CI/CD for Bitbucket Cloud that helps you automate your code from test to production in the Cloud or using your own infrastructure." A button "Create your first pipeline" is present, with a downward arrow pointing to a terminal window. The terminal window shows the execution of a pipeline named "Pipeline #42" with the following steps and durations:

Step	Duration
> Build Setup	5s
> export	20s
> yarn install	5s
> yarn test	10s
> Deployment step	2m

Continuous Integration (CI)

bitbucket-pipelines.yml

```
# Voorbeeld voor EMS30.
```

```
# Docker image gemaakt door Daniël Versluis: deze draait op de laatste gcc image maar bevat nu  
# ook cmake en catch2.
```

```
image: versd/gcc_cmake_catch2:latest
```

```
pipelines:
```

```
  branches:
```

```
    master:
```

```
      - step:
```

```
        name: Build
```

```
        script:
```

```
        # Deze file moet in het root directory van je bitbucket repository staan.
```

```
        # Indien nodig, kun je met een cd commando naar het juiste directory gaan.
```

```
        # Bijvoorbeeld:
```

```
        - cd Voorbeelden/breuk-test-ci
```

```
        - mkdir build
```

```
        - mkdir test-results
```

```
        - cmake . -B build/
```

```
        - cmake --build build/ --config Debug --target all
```

```
        - ./build/test_breuk -sr junit > test-results/raportage.xml
```

Zie: <https://confluence.atlassian.com/bitbucket/configure-bitbucket-pipelines-yml-792298910.html>

Zie: https://bitbucket.org/HR_ELEKTRO/ems30/raw/master/Programmas/breuk-test-ci.zip

Continuous Integration (CI)

```
Build Tests 🔍 🔄 🗑️ ℹ️
```

```
Build setup 3s >
```

```
cd Voorbeelden/breuk-test-ci <1s >
```

```
mkdir build <1s >
```

```
mkdir test-results <1s >
```

```
cmake . -B build/ 1s >
```

```
cmake --build build/ --config Debug --target all 10s >
```

```
./build/test_breuk -sr junit > test-results/raportage.x... >
```

```
Build teardown <1s >
```

Continuous Integration (CI)

```
Build Tests ⓘ  
  
3 / 20 tests failed  
  
global.rekenen met breuken/vermenigvuldigen test_breuk <1s ▾  
  
1   c.teller == 1  
2  
3  
4   FAILED:  
5     CHECK( c.teller == 1 )  
6   with expansion:  
7     2 == 1  
8   at /opt/atlassian/pipelines/agent/build/Voorbeelden/breuk-test-ci/test_breuk.cpp:33  
9  
10
```

Zie: live demo

Memory tests en test coverage

LCOV - code coverage report

Current view: [top level](#)

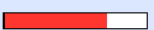
Test: [main.info](#)

Date: 2023-04-20 13:56:55

Hit Total Coverage

Lines: 23 32 71.9 %

Functions: 4 6 66.7 %

Directory	Line Coverage	Functions
breuk-test-coverage	 71.9 % 23 / 32	66.7 % 4 / 6

```
48 Breuk mul(Breuk b1, Breuk b2) 2x [100.0%]
49 {
50     Breuk product;
51     product.teller = b1.teller * b2.teller; 2x
52     product.noemer = b1.noemer * b2.noemer; 2x
53     return normaliseer(product); 2x
54 }
55
56 Breuk divide(Breuk b1, Breuk b2)
57 {
58     Breuk quotient;
59     quotient.teller = b1.teller * b2.noemer;
60     quotient.noemer = b1.noemer * b2.teller;
61     return quotient;
62 }
```

Aan de slag!

Aan de slag met [Opdrachten Week 2 Les 2.pdf](#)

