Unit Testing

Elektronica – ICT

Sven Mariën

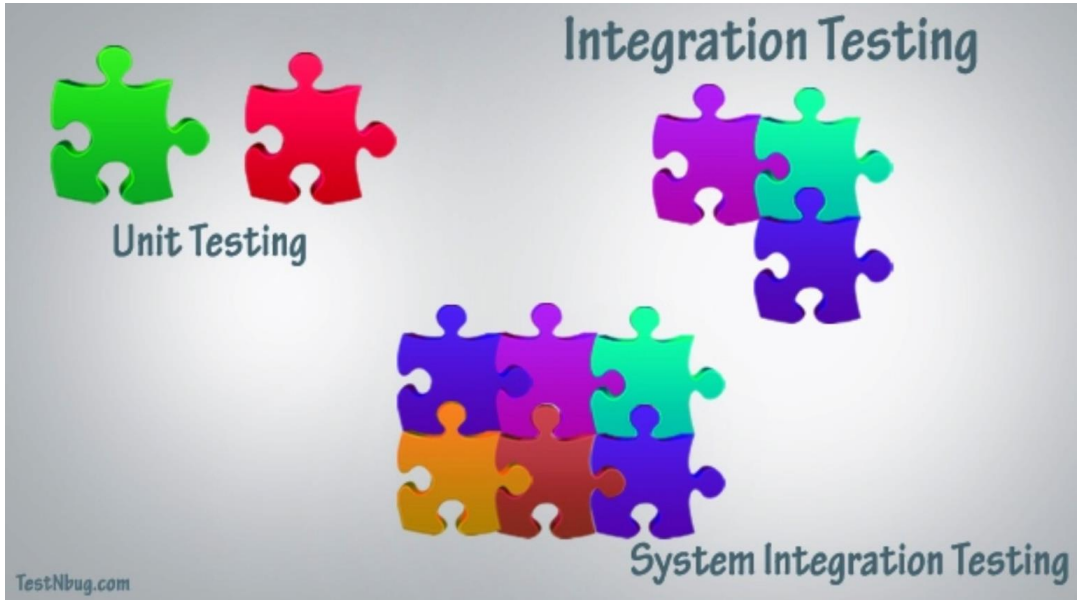
(sven.marien01@ap.be)

2018-2019

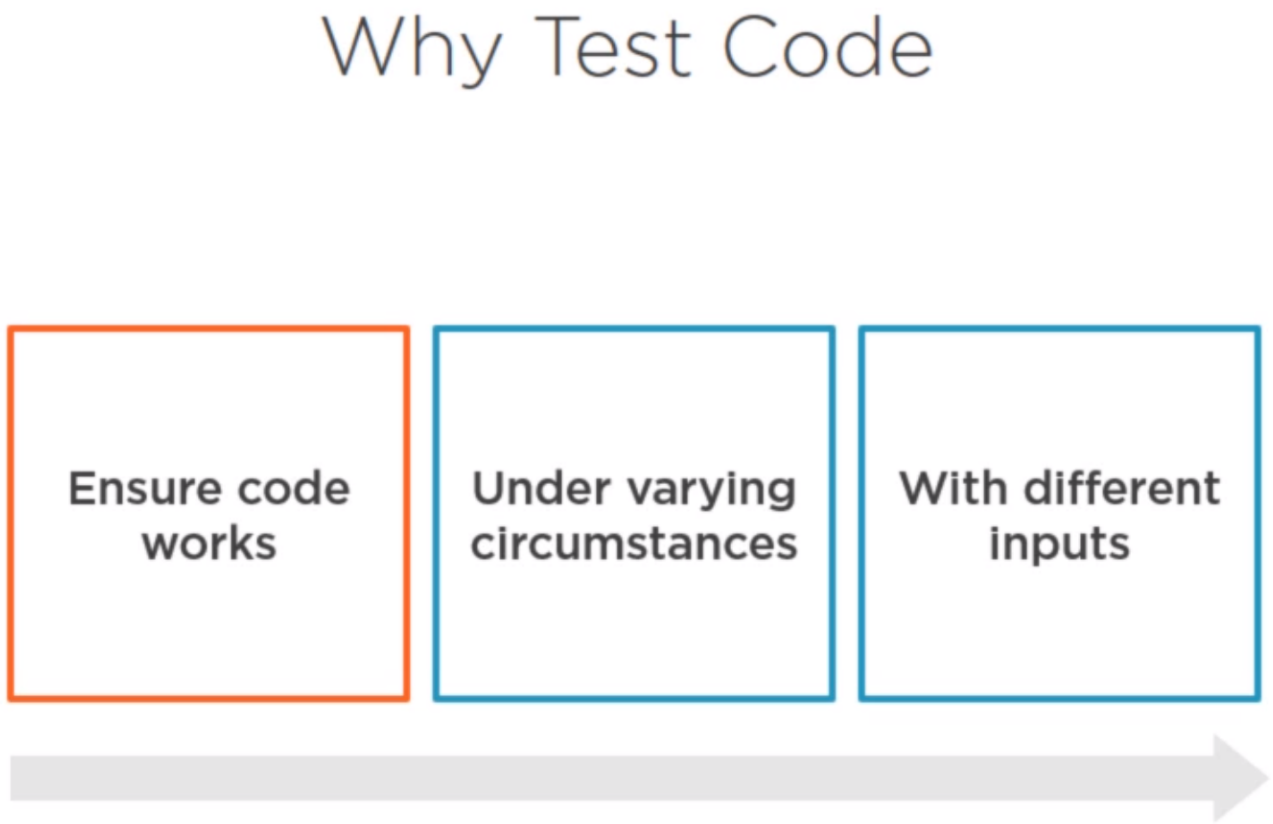
ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

Wat is Unit Testing ?

- Testen van je code op het laagste niveau
- Testen van de kleinst testbare delen van de applicatie
- Een “unit” kan zijn:
  - Functie
  - Klasse, of een deel van een klasse
- Naast unit testing is er ook nog
  - Integration testing
    - Testen van een aantal klassen tezamen
  - System Testing
    - Testen van de volledige applicatie
  - Acceptance testing
  - Smoke testing
  - Regression testing
  - ,..



Waarom unit testing ?

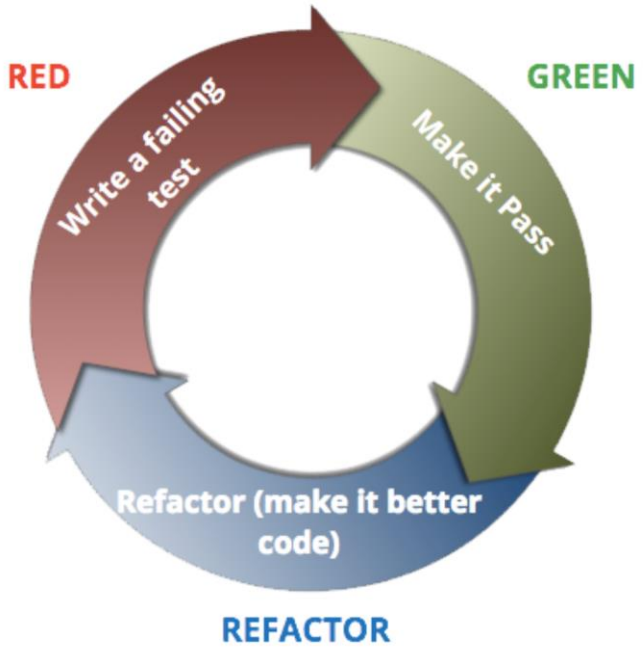


Waarom unit testing ?

- Fouten worden snel gevonden
  - We testen kleine stukken code afzonderlijk vooraleer ze samen worden getest
- Geeft betere code
  - Door het maken van unit tests ga je extra nadenken over de code en alle randvoorwaarden en negatieve paden.
- Geeft extra documentatie
  - De unit testen geven aan hoe een component moeten worden gebruikt (net zoals je zou gaan zoeken op het internet naar code voorbeelden over een bepaalde component.)
- Geeft je een extra zekerheid wanneer bestaande code moet worden aangepast
  - Oplossen van een bug
  - Toevoegen van nieuwe functionaliteit
  - Refactoring van een stuk bestaande code of een component
- ..

Wanneer unit testing ?

- Klassiek zou men denken dat unit tests geschreven worden **nadat** een unit werd ontwikkeld om de goede werking ervan te bevestigen.
- Dit is eigenlijk te laat, door de tests te schrijven **tijdens de ontwikkeling** van de unit wordt je verplicht om testbare code te schrijven en reeds na te denken over de verschillende scenario's die je gaat moeten testen
- Er is ook een tendens richting "Test Driven Development" (TDD). Volgens deze manier van werken worden de test scenario's **eerst** geschreven vooraleer de eigenlijke component wordt geschreven. Dit gebeurt volgens het principe van **Red-Green-Refactor**
- Bij het oplossen van "bugs", wordt er klassiek ook een bijhorende unit test geschreven die het betreffende scenario dat de bug veroorzaakte ook gaat testen.



Wanneer unit tests uitvoeren ?

- Uiteraard tijdens & na ontwikkeling van een nieuwe unit
- Na oplossen van een bug of aanpassen van een unit
- Voor het inchecken (commit) van je code alles tests uitvoeren
- Kan ook geautomatiseerd (vb. testrunner) worden zodat alle testen worden uitgevoerd na een automatische build – in combinatie met een mailing / notification system

- Zorg er daarom voor dat unit tests compact zijn, snel uitvoeren en robuust zijn (net zoals je produktiecode)
- Investeer zeker in unit testing !

Schrijven van een unit test: AAA methode

Assert

- Equality
  - AreEqual, AreNotEqual
- Identity
  - AreSame, AreNotSame, Contains
- Condition
  - IsTrue, IsFalse, IsNull, IsNotNull, IsNan, IsEmpty, IsNotEmpty
- Type
  - IsInstanceOf<T>, IsNotInstanceOf<T>
- Exception
  - Assert.throws

Initialisatie & cleanup : 3 niveau's

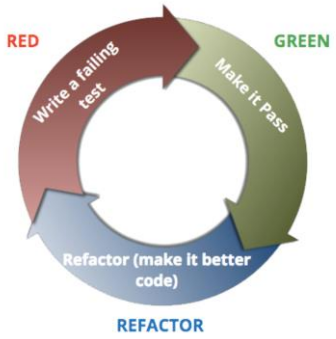
AssemblyInitialize	ClassInitialize	TestInitialize
– Called once for project – Setup resources for all test classes in assembly	– Called once for a class – Setup resources for all tests within a class	– Called once for each test method – Set or reset resources needed for each test

AssemblyCleanup	ClassCleanup	TestCleanup
– Called once after all tests in assembly have run	– Called once after all tests in class have run	– Called once after each test method has run

Unit tests: oefening FileHelperTest met Init & cleanup

- Breid de FileHelperTest uit zodat we een eigen testbestand
  - aanmaken voor de test
  - Verwijderen na de test
  - Gebruik hiervoor de **initialisatie** & **cleanup** attributen

Unit tests : oef TDD



- Opdracht:
  - maak een klasse **StringConverter**
  - Geef een methode: **Capitalize** (string tekst), die alle woorden een omvormd naar woorden startende met telkens een hoodletter (Zoals Deze Zin Dus)
- Werk volgens de TDD methode
  - Maak eerst een testklasse (**StringConverterTest**)
  - geef deze1 test : **TestLowerCaseWord** waarbij een woord met kleine letters wordt getest
  - Deze test werkt niet omdat de StringConverter zelf nog niet bestaat (RED)
  - Maak de stringconverter aan en implementeer de **Capitalize** method zodat de test lukt (GREEN)
  - Verbeter de code indien nodig (REFACTOR)
  - Maak een 2e test **TestUpperCaseWord** waarbij een woord met hoofdletters wordt getest (RED)
  - Pas de StringConverter aan zodat ook deze test werkt (GREEN)
  - Pas aan indien nodig (REFACTOR)
  - Ga verder met 2 woorden, zinnen, spatie voor het eerste woord, na het laatste, lege string,..
  - Breid uit met een 2e parameter (enum CapitalizeMethod) die aangeeft of bij een zin enkel het eerste woord zo niet alle woorden moeten worden omgevormd naar een hoofdletter
  - Check op meerdere zinnen, enz…

Unit tests: oefening RRnr

- Maak een klasse rijksregisternummer
- https://nl.wikipedia.org/wiki/Rijksregisternummer
- Geef deze 2 constructors
  - **Rijksregisternummer** (int jaar, int maand, int dag, Gender geslacht, int dagnr)
  - **Rijksregisternummer** (string rrnr)
- Nadien kan je aan de klasse ls properties het **rijksregisternummer** opragen en ook de inviduele componenten (jaar, maand, dag, geslacht, dagNr en controlegetal)
- Het rrnr wordt steeds als 1 string van **enkel** 11 cijfers gegeven
- **Houd in eerste instantie geen rekening met het jaar >= 2000, maw. enkel met personen die geboren zijn in 19xx**
- Maak vervolgens (of tegelijkertijd) een testklasse hiervoor die zoveel mogelijk scenario's aftest.
- Pas nu de klasse aan zodat je ook rekening houd met geboortes na 1999. De aanwezige testen geven je nu een houvast opdat de oude logica blijft werken zoals voordien.