Unit Testing (2)
Test doubles

Elektronica – ICT

Sven Mariën

(sven.marien01@ap.be)

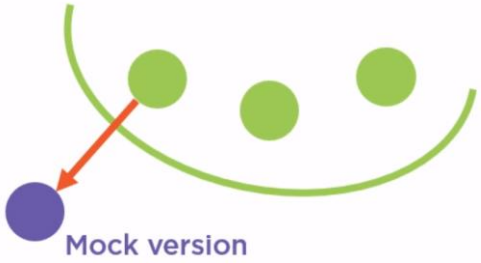
2018-2019

ARTESIS PLANTIJN
HOGESCHOOL ANTWERPEN

Unit testing : dependencies

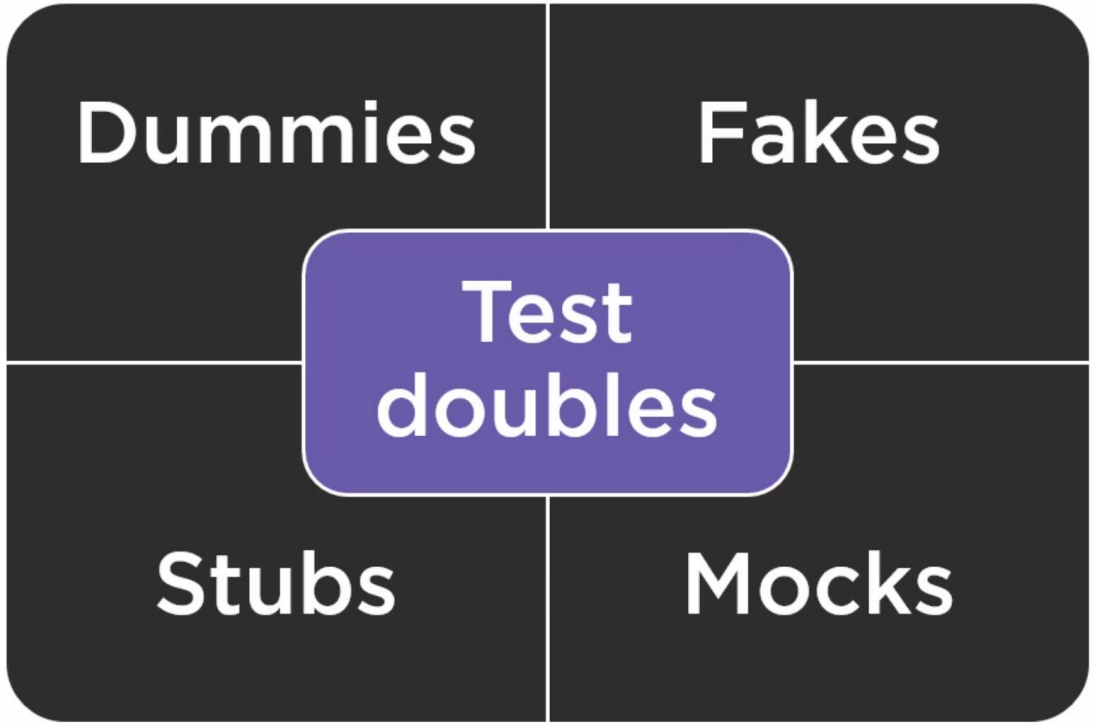
- Hoe kunnen we een unit testen als die nog externe dependencies heeft ?
- Volgens het **Dependency Inversion Principe** (SOLID) worden deze best expliciet gemaakt
- We hebben dan verschillende mogelijkheden:
  - Testen met de echte dependency (unit test maakt een instantie ervan aan)
    - Vb. FileLogger
  - Testen met "**Test doubles**"
    - Dit is een vervanging voor de echte dependency.
    - Je kan deze zelf schrijven of laten aanmaken door een Mocking Framework, bv.
      - Moq
      - FakeItEasy
      - Nsubstitute
      - …

Replacing the actual dependency that would be used at production time, with a test-time-only version that enables easier isolation of the code we want to test.



“Test doubles”

- Dummy
  - Is een leeg object zonder gedrag of state.
- Fake
  - Heeft een gedrag zoals het echte object maar kan niet in productie worden ingezet (bv. Een InMemoryDatabase)
- Stub:
  - Heeft geen gedrag maar geeft wel een bepaalde ingestelde waarde terug bij het aanroepen van een property of method. Net voldoende om de test te doen slagen.
- Mock
  - Mocks kunnen worden ingezet om na te gaan of bepaalde properties en/of methods wel degelijk worden aangeroepen.



Voordelen van het gebruik van “Test Doubles”

- Testen hebben een kortere uitvoeringstijd, bv.
  - Als de dependency een traag algoritme bevat
  - Als de dependency externe resources moet aanspreken: DB, webservice,…
- Ontwikkeling kan in parallel gebeuren:
  - Mogelijk is de “echte” dependency nog niet klaar
  - Wordt ontwikkeld door een ander team
  - Wordt ontwikkeld door een externe partij
- Testen worden betrouwbaarder
  - We beperken ons immers tot de “unit”
- Testen met een dependency die willekeurige output geeft
  - Bv. Een sensor die temperatuur,… meet kunnen anders moeilijk of niet getest worden

“Moq” framework

- Installeren in je project adhv. NuGet package
- Deze kan je zowel als **Stubs en/of** als **Mocks** gebruiken.
- Maak een nieuwe “Test double” aan:

```
var personFromMock = new Mock<IPerson>();
var personToMock = new Mock<IPerson>();
```

- Gebruik het object waar nodig:

```
var paymenthandler = new PaymentHandler(personFromMock.Object, personToMock.Object);
```

- Standaard gedrag van “Moq” objecten:
  - Properties en Methods geven default waarde terug:
    - **0** voor int, double, …
    - **false** voor een boolean
    - **null** voor een string , objectverwijzing
  - Properties houden hun waarde niet bij als ze worden ingesteld (enkel “getters”)

```
public interface IPerson
{
    2 references
    string Name { get; set; }

    2 references
    string FirstName { get; set; }

    3 references
    int Age { get; set; }

    8 references
    IBankAccount Account { get; set; }

    1 reference
    bool IsStudent();
}
```

Stubs

- Om een bepaalde “property” een vaste waarde laten teruggeven:

```
personToMock.Setup(x => x.Age).Returns(21);
personToMock.Setup(x => x.FirstName).Returns("Jonan");
```

- Om een “method” een waarde te laten teruggeven

```
personToMock.Setup(x => x.IsStudent()).Returns(true);
```

- Dit kan tevens worden gestuurd aan de hand van verwachte parameter(s):

```
var accountToMock = new Mock<IBankAccount>();
accountToMock.Setup(x => x.CanWithDrawal(20)).Returns(true);
accountToMock.Setup(x => x.CanWithDrawal(50)).Returns(false);
```

- Of ongeacht welke waarde als parameter wordt doorgegeven:
  - Gebruik het **It** object

```
accountToMock.Setup(x => x.CanWithDrawal(It.IsAny<double>())).Returns(true);
```

  - Ook Ranges, e.d. zijn allemaal mogelijk

```
accountToMock.Setup(x => x.CanWithDrawal(It.IsInRange(10,50, Range.Inclusive))).Returns(true);
```

```
0 references
public class Person : IPerson
{
    1 reference
    public string Name { get; set; }
    2 references
    public string FirstName { get; set; }
    3 references
    public int Age { get; set; }
    14 references
    public IBankAccount Account { get; set; }
    2 references
    public bool IsStudent()
    {
        return Age >= 17;
    }
```

```
5 references
public interface IBankAccount
{
    1 reference
    string IBAN { get; }

    7 references
    double Balance { get; set; }

    2 references
    bool CanGoNegative { get; }

    4 references
    bool Deposit(double amount);

    4 references
    bool WithDrawal(double amount);

    6 references
    bool CanWithDrawal(double amount);
    4 references
    bool CanDeposit(double amount);
}
```

Stubs (2)

- Ook relaties kunnen door het framework automatisch worden ge'mocked'.

```
personToMock.Setup(x => x.Account.Balance).Returns(45); // maakt een Account Mock en zet de return waarde voor Balance
personToMock.Setup(x => x.Account.Balance);             // Maakt enkel een account object, Balance geeft default (+à) terug
```

```
0 references
public class Person : IPerson
{
    1 reference
    public string Name { get; set; }
    2 references
    public string FirstName { get; set; }
    3 references
    public int Age { get; set; }
    14 references
    public IBankAccount Account { get; set; }
    2 references
    public bool IsStudent()
    {
        return Age >= 17;
    }
```

```
5 references
public interface IBankAccount
{
    1 reference
    string IBAN { get; }

    7 references
    double Balance { get; set; }

    2 references
    bool CanGoNegative { get; }

    4 references
    bool Deposit(double amount);

    4 references
    bool WithDrawal(double amount);

    6 references
    bool CanWithDrawal(double amount);
    4 references
    bool CanDeposit(double amount);
}
```

Stubs (3)

- Stubs geven standaard wel een default of ingestelde waarde terug.
- Maar **ze onthouden standaard niet** welke waarde nadien eventueel wordt ingesteld.
- Om toch het echte get/set “property gedrag” te bekomen gebruik je **SetupProperty**

```
personFromMock.SetupProperty(x => x.Age);
personFromMock.SetupProperty(x => x.Name);
```

- Kan ook ineens voor **alle properties** van een Moq object

```
personFromMock.SetupAllProperties();
```

- Je kan hierbij ook individueel een default waarde instellen:

```
personFromMock.SetupProperty(x => x.Age, 18);
personFromMock.SetupProperty(x => x.Name, "Dirk");
```

```
0 references
public class Person : IPerson
{
    1 reference
    public string Name { get; set; }
    2 references
    public string FirstName { get; set; }
    3 references
    public int Age { get; set; }
    14 references
    public IBankAccount Account { get; set; }
    2 references
    public bool IsStudent()
    {
        return Age >= 17;
    }
}
```

Loose versus **Strict** gedrag

```
var personFromMock = new Mock<IPerson>(MockBehavior.Strict);
var personToMock = new Mock<IPerson>(MockBehavior.Loose);
```

- Loose Behaviour
  - Standaard moet je niet alle methods met Prepare instellen
  - Enkel deze die belangrijk zijn voor het resultaat van je test
  - Dit wordt het ‘Loose behaviour’ genoemd
- Strict behaviour:
  - Wens je toch dat er een exceptie wordt gegooid indien een method wordt aangeroepen die je niet had ingesteld met Prepare
  - Kies dan voor het ‘Strict behaviour’

Loose	Strict
Less lines of setup code	More setup code
Default values	Have to setup each called method
Less brittle tests	More brittle tests
Existing tests continue to work	Existing tests may break

Use strict mocks only when absolutely necessary, prefer loose mocks at all other times.

Mocks: testen van het gedrag of de “interactie”

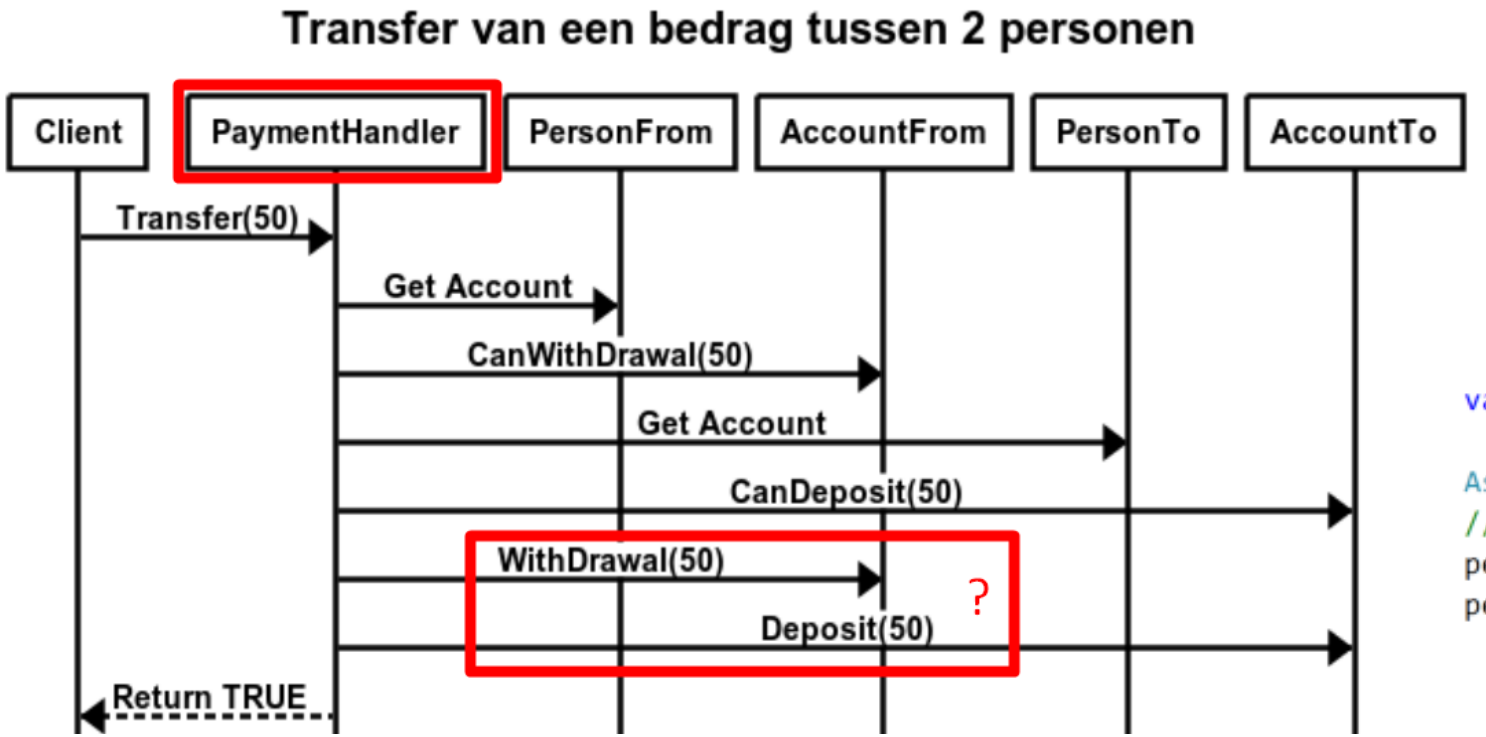
- Soms is het moeilijk of onmogelijk om uit een unittest af te leiden of de test geslaagd is
- Bv. Als de unit weinig of geen resultaat teruggeeft.
- We kunnen dan mbv. Mocks het **gedrag** meer in detail testen.
- Mocks houden bij welke methoden werden aangeroepen tijdens de test.
- Aan het einde van de test kan dan je **verifiëren** of de verwachte methodes daadwerkelijk werden aangeroepen.

```
var personFromMock = new Mock<IPerson>();
var personToMock = new Mock<IPerson>();

var paymenthandler = new PaymentHandler(personFromMock.Object, personToMock.Object);
var result = paymenthandler.Transfer(50);

Assert.IsTrue(result);
//Verify if the correct methods were called on the Mocks
personFromMock.Verify(x => x.Account.WithDrawal(50));
personToMock.Verify(x => x.Account.Deposit(50));
```

```
5 references
public interface IBankAccount
{
    1 reference
    string IBAN { get; }

    7 references
    double Balance { get; set; }

    2 references
    bool CanGoNegative { get; }

    4 references
    bool Deposit(double amount);

    4 references
    bool WithDrawal(double amount);

    6 references
    bool CanWithDrawal(double amount);
    4 references
    bool CanDeposit(double amount);
}
```

Mocks (2)

- Voorbeeld van een interactie tussen objecten
- De Paymenthandler is de Unit in test. Als het resultaat van de Transfer = TRUE weten we nog niet of het geld daadwerkelijk werd overgeschreven. We kunnen dan daarnaast verifiëren of ook de juiste methods werden aangeroepen op de Account Mocks.

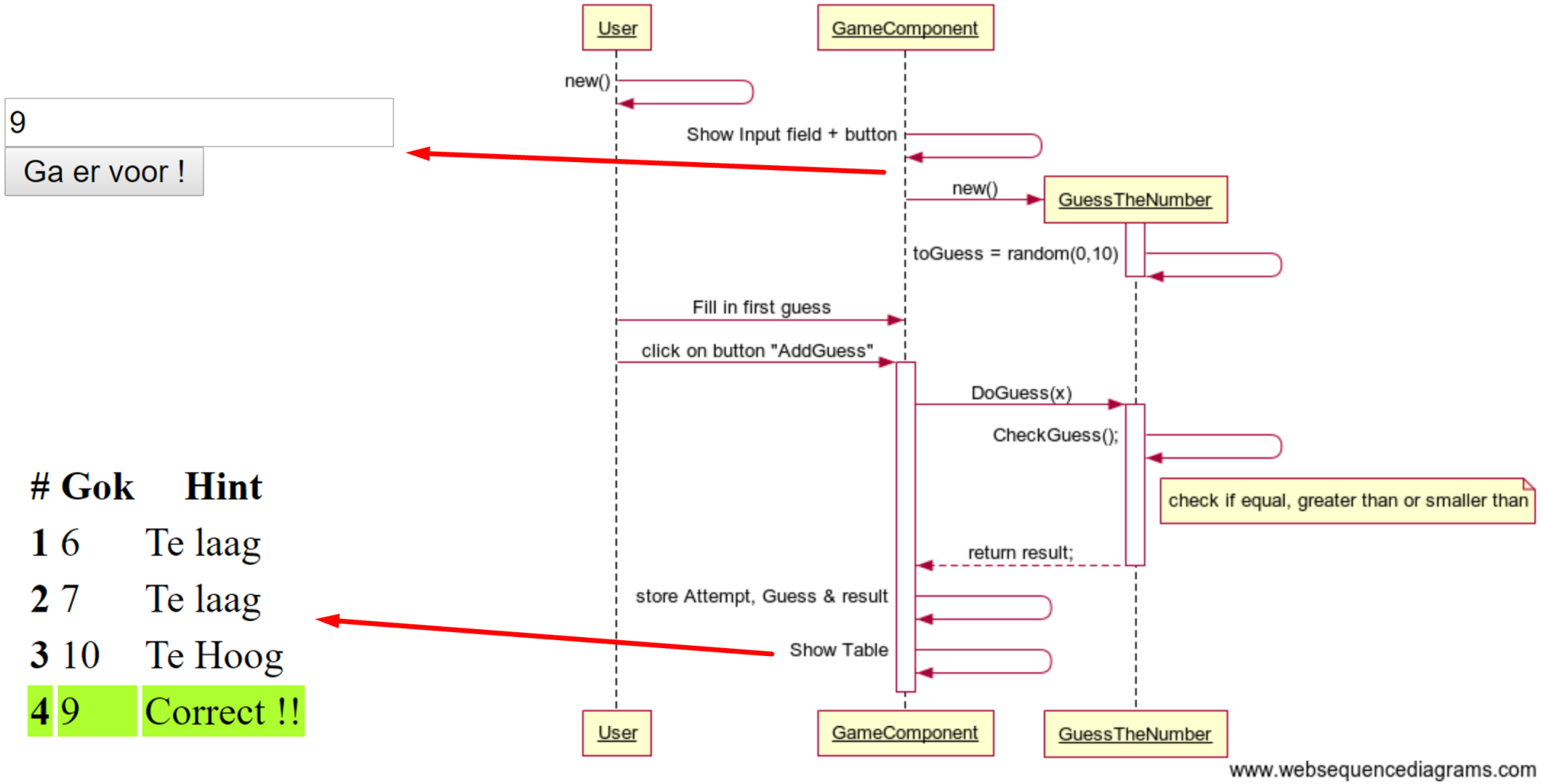



```
var result = paymenthandler.Transfer(50);

Assert.IsTrue(result);
//Verify if the correct methods were called on the Mocks
personFromMock.Verify(x => x.Account.WithDrawal(50));
personToMock.Verify(x => x.Account.Deposit(50));
```

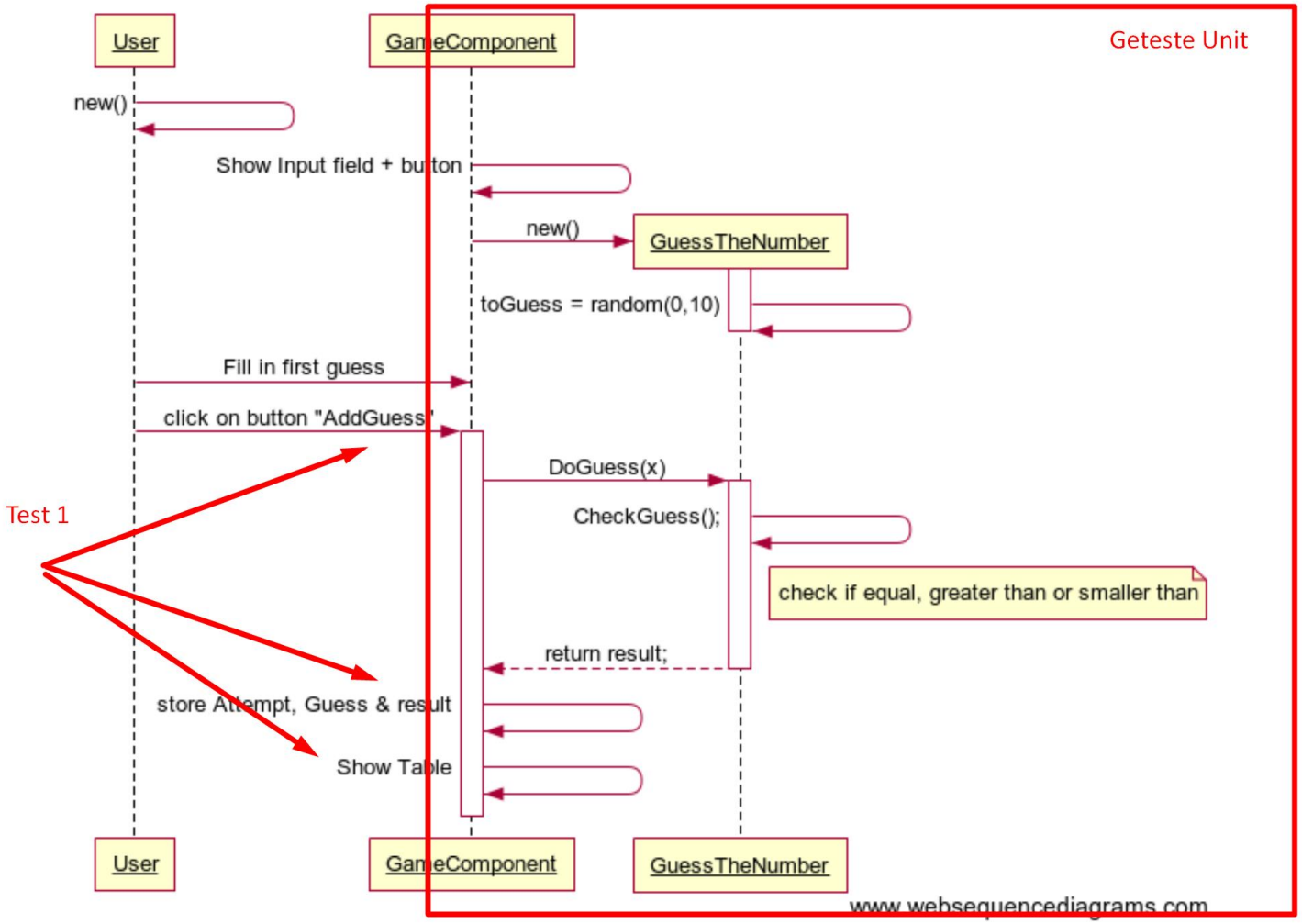
Oefening

- Ontwerp een component (class library) voor aansturing van een automatische zonnewering.
  - Deze component heeft een aantal dependencies (motor, lichtsensor, windsensor) dewelke je uiteraard niet mee in de unit test mag betrekken.
  - Voorzie hiervoor dan ook de nodige “test doubles”, de Motor is een “Mock”, de sensoren zijn “Stubs”.
- Test volgende scenario’s:
  - Als de zonnewering wordt neergelaten (bij wind = 0) dat de motor effectief wordt gestart.
  - De zonnewering mag echter niet worden neergelaten indien de windsensor een windsnelheid aangeeft die hoger is dan 50 km/h. **Verifieer via je test** dat de motor effectief **niet** word gestart.
  - Maak een zonnewering object aan zonder deze te bedienen en zorg voor een lichtinval van > 50000 lux. De zonnewering moet na 1 seconde **autonoom** naar beneden gaan. Test dit scenario.
  - Test het scenario dat de zonnewering autonoom omhoog gaat bij te hoge windsnelheid.

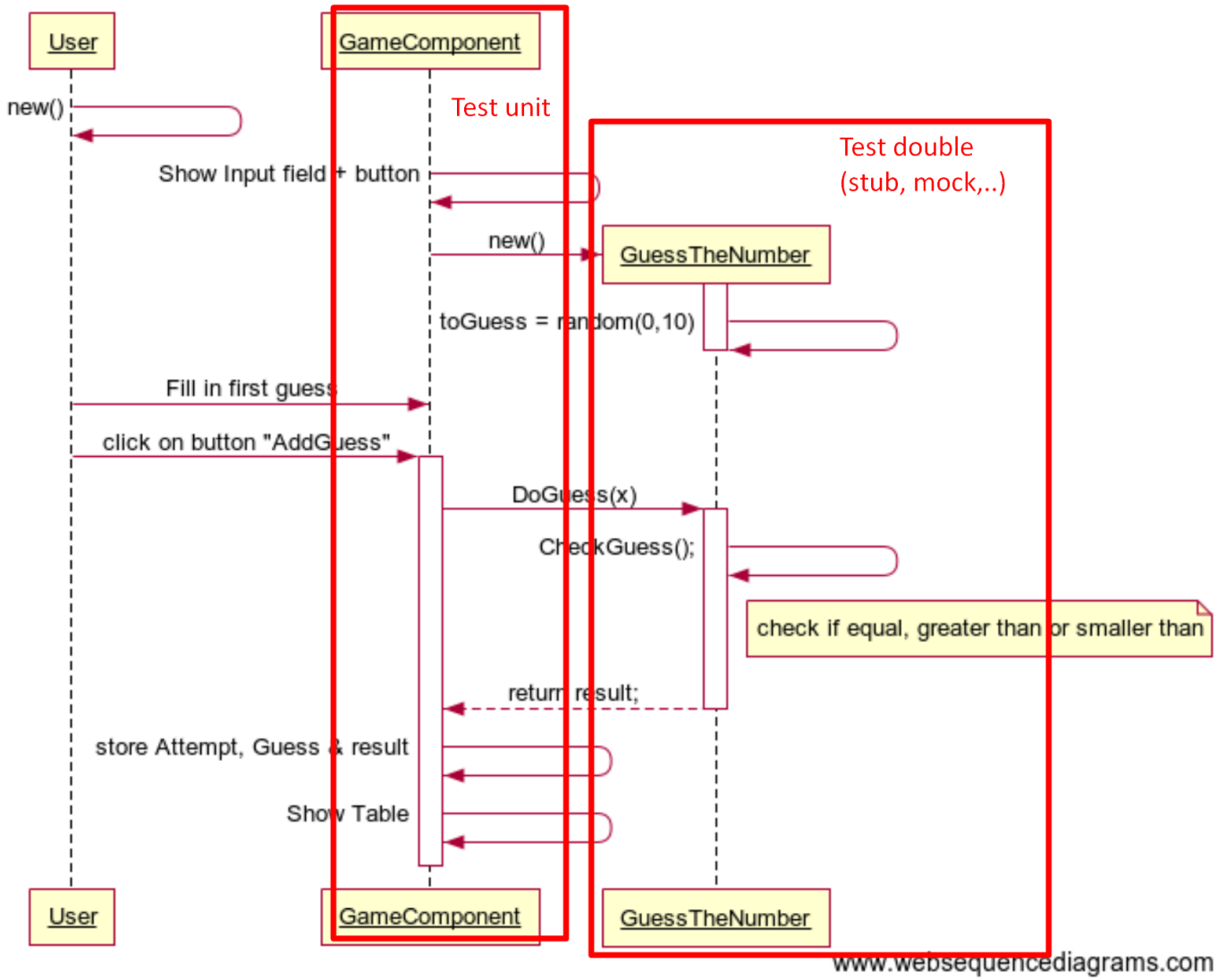
Testing met Jasmine/Karma



Wat hebben we nu getest ?



Gebruiken van een “Test double”



3e optie “SpyOn”:enkel bepaalde methodes beïnvloeden