Testautomatisering

BDD, RSpec

Idag

- FM:
  - Snabbutvärdering, lab
  - BDD
  - RSpec

Kort om labben

- Lab2 - Snabbutvärdering
  - 1. Hur många timmar har du lagt?
  - 2. Hur många ytterligare timmar kommer du lägga?
  - 3. Svårighet: För **Lätt** / **Lagom** / För **Svår**
  - 4. Lärdommar - “Jag ser... “:
    - Ingen praktisk nytta (**-**)
    - Viss praktisk nytta (**0**)
    - Stor praktisk nytta (**+**)

TDD

- TDD
  - Test Driven Development
  - I korthet: vi utvecklar testerna innan implementationen de skall testa

TDD

- TDD
  - Red -> Green -> Refactor -> repetera…
  - **Red:** Skriv ett test som ger fail
  - **Green** Skriv tillräckligt utav implementationen för att få pass
  - **Refactor:** Är vi nöjd med implementation och test? Om inte – gör förändringar med de test vi skrivit som stöd.
  - … skriv nästa test

TDD

- TDD
  - Vad får vi?
    - För det mesta: En robustare implementation
      - (Mer modulär, etc.)
    - (Vissa) Regressionstester på köpet!
      - (Vi kan behöva ytterliggare regressionstester - men vi bör ha en hyfsad plattform att utgå ifrån här).
    - Mindre skräp
      - (Vi skriver endast tillräckligt med implementation för att testet skall passera)
  - M.m.

- TDD – Varning
  - Vi bör ha någon uppfattning om de övergripande strukturen för vårt projekt innan vi börjar med TDD.

TDD

- TDD – Problem
  - Vi riskerar att missa helheten?
  - Vad är affärsnyttan?

BDD

- BDD
  - Behaviour Driven Develpment
  - I korthet: Vi låter det önskade beteendet styra designen.
  - BDD uppstod som en reaktion på de problem som uppstod då man utvecklade enligt TDD

BDD

- BDD vs TDD
  - BDD bygger på TDD
  - TDD fortfarande mer vanligt
  - BDD utforskar vad ett objekt gör, medans TDD utforskar vad ett objekt är.
  - Om detta är otydligt, var inte orolig: Vi kommer prata BDD imorrn och på fredag också.

RSpec

- RSpec
  - RSpec är ett testramverk för Ruby (motsvarande Test::Unit som vi kikade på förra veckan)
  - RSpec kan användas för att skriva enhetstester
  - RSpec kan användas för TDD
  - Vi kommer använda det för BDD

- Rspec - Installation
  - gem install rspec
  - (eller för xubuntu VM: sudo gem install rspec)

RSpec

RSpec

- Rspec
  - Subject code: koden vi beskriver
  - Expectation: Hur koden vi beskriver förväntas bete sig
  - Example: Ett körbart exempel på hur koden vi beskriver kan användas
  - Example Group: En grupp av kodeexempel
  - Spec file: En eller fler a Example Groups

- Rspec
  - Beskriv ett nytt konto
  - Det skall innehålla 0 USD

RSpec

- Rspec
  - Beskriv ett nytt objekt av typen Account
  - Beskrivning: Dess property ”balance” skall innehålla ett objekt av typen Money. Money-objektet skall representera 0 USD.

RSpec

- Rspec - exempel

```
describe ”A new Account” do
  it ”should have a balance of 0” do
    account = Account.new
    account.balance.should = Money.new(0, :USD)
  end
end
```

RSpec

RSpec

- Rspec

  Ett example resulterar i ett test och man kan även ta ut en rapport (en beskrivning av testet som är tänkt att vara läsbar för vilken stakeholder som helst).

- Rspec

```
module Authentication
    describe User, ”with no roles assigned” do
                …
    end
```

RSpec

- Rspec

  Authentication::User with no roles assigned

RSpec

- Rspec

```
describe User do
    describe User, ”with no roles assigned” do
        it ”should not be allowed to view protected
            content” do
    …
```

RSpec

- Rspec

User with no roles assigned

- should not be allowed to view protected content

RSpec

- Rspec - describe
  - Describe-metoden:
    - Tar 1-2 argument
    - Första argumentet är en klass eller en sträng
    - Andra argumentet är optional och bör vara en sträng

- Rspec - exempel
  - Describe ”A User” {…}
    - => A User
  - Describe User {…}
    - => A User

RSpec

- Rspec - exempel
  - describe User ”with no roles assigned” {…}
    - => A User with no roles assigned
  - describe User ”should require password length between 5 and 40” {…}
    - => User should *require* password between 5 and 40

RSpec

- Rspec – it()

```
describe ”A new Account” do
  it ”should have a balance of 0” do
    account = Account.new
    account.balance.should = Money.new
  end
end
```

RSpec

- Rspec – it()

  Argument: En sträng, en optional Hash och ett optional block

RSpec

- Rspec – it()

```
describe Stack do
    before(:each) do
        @stack = Stack.new
        @stack.push :item
    end
```

… forts. nästa sida

RSpec

- Rspec – it()

```
describe ”#pop” do
    it ”should return the top element” do
        @stack.pop.should == :item
    end

    it ”should remove the top element” do
        @stack.pop
        @stack.size.should == 0
    end
end
```

RSpec

- Rspec – it()

  Varför pending block?

  describe ”#pop” do
  it ”should return the top element”
  End

  =>

  Pending:
  #pop should return the top element

RSpec

RSpec

- Rspec – it()
  - Dvs. vi kan skriva in våra förväntade tester utan att behöva implementera dem
  - Bra när vi utvecklar tester för att enklare se helheten.

- Rspec – before

```
describe Stack do
   before(:each) do
      @stack = Stack.new
      @stack.push :item
   end
end
```

Körs innan varje test – likt setup i Test::Unit

RSpec

- Rspec – before
  - Vi kan dock göra mer med before
  - Vi kan använda det för att sätta olika kontext för olika test

RSpec

- Rspec – before
  - describe Stack, ”when empty” do
    - before(:each) do
      - @stack = Stack.new
    - end
  - end
  - describe Stack, ”when almost empty (1 element)” do
    - before(:each) do
      - @stack = Stack.new
      - @stack.push 1
    - end
  - end

RSpec

- Rspec – before
  - describe Stack, ”when full” do
    - before(:each) do
      - @stack = Stack.new
      - {1..10}.each {|n| @stack.push n}
    - end
  - end

- Rspec – before
  - Men vi har också möjligheten att göra ngt innan alla tester
  - describe Stack do
    - before(:all) do
      - @stack = Stack.newend
    - End
  - Vi kan använda både before(:all) och before(:each) för samma klass.

- Rspec – before
  - Before är extremt nyttigt.
  - Det är väldigt sällan man vill ha samma setup för samtliga tester.
  - Ofta har man istället grupper av test, där 3-4 tester använder samma preconditions

RSpec

- Rspec – after
  - after(:each)
  - after(:all)

RSpec

RSpec

- Rspec – hjälpmetoder
- Precis som i Test::Unit kan vi använda oss av hjälpmetoder

- Rspec – hjälpmetoder
- module UserExampleHelpers
  - def create_valid_user
    - return User.new(”user”, ”password”)
  - end
  - def create_invalid_user
    - return User.new(”user”, ”pass”)
  - end
- end

RSpec

- Rspec – hjälpmetoder
- describe User do
  - include UserExampleHelpers
  - it ”does something when it is valid” do
    - @user = create_valid_user
  - end
- end

RSpec

- Rspec – nästlade test

- describe Stack do
  - describe ”when full” do
    - describe ”when it recieves a push” do
      - it ”should raise an error” do
        - # ….should raise_error(StackOverFlowError)
      - end
    - end
  - end
- end

RSpec

- Rspec – Expectations
- should och should_not

RSpec

- Rspec – Expectations

- should och should_not

  - test_user = User.new(”First”, ”Last”)
  - test_user.should == ”First Last”
  - test_user.should_not == ”FirstLast”

- Rspec – Expectations

- should och should_not

  - [1,2,3].should_not == [1,2,3,4]
  - [1,2,3].lenght.should == 3

- Rspec – Expectations

- should och should_not

  - [1,2,3].should_not == [1,2,3,4]
  - [1,2,3].lenght.should == 3

- Rspec – Expectations

- should och should_not

  - [1,2,3].should_not == [1,2,3,4]
  - [1,2,3].lenght.should == 3
  - [1,2,3].should include(3)

- Rspec – Expectations
- match
  - ”Totals: 12”.should match(/Totals: 12/)

RSpec

- Rspec – Expectations

- raise_error

- lambda {User.new}.should raise_error(ArgumentError)

RSpec

- Rspec – Expectations
- be_
- Om objektet har en metod som slutar på ”?”
- Istället för
  - Array.empty?.should == true
- Säger vi:
  - Array.should be_empty

RSpec

- Rspec – Expectations

- be_
- User har en in_role?(role)-metod
  - => user.should be_in_role(”Admin”)

- Rspec – Expectations
- have
- Om objektet har en metod som börjar med has_
- Istället för
  - hash.has_key?(:id).should == true
- Säger vi:
  - hash.should have(:key)

- Rspec – Expectations
- have
- Collections - ibland
- Istället för
  - team.length.should == 9
- Säger vi:
  - team.should have(9).players()
- OBS! här krävs det lite ytterligare – se boken kap 11 för mer info

RSpec

RSpec

- Rspec
  - describe
  - it
  - pending
  - before
  - after
  - Hjälpmetoder
  - Nästlade test

- Class TestSomeClass < Test::Unit::TestCase
- => describe SomeClass

- def test_something
- => it ”should do something descriptive”

- def setup
- => before(:each)

- def teardown
- => after(:each)

- assert_equal(4,array,length,”…”)
- array.length.should == 4

RSpec vs Test::Unit

- Rspec
  - Saker vi inte tagit upp
  - Shared Examples
  - Lambda { user.Create!(:role => admin)}.should change{User.admins.count}

Rspec

RSpec

- Rspec
  - Kodexempel

RSpec

- BDD
  - Hur passar det här in i BDD?
  - Ännu har vi inte tagit upp särskillt mycket nytt
  - RSpec gör ungefär samma saker som Test::Unit
  - Vi behöver ytterligare en pusselbit

Unit Test

- BDD
  - Cucumber
  - Detta är ett verktyg för utveckling enligt BDD
  - Exempel:
  - Vi kommer kika på detta imorrn

- BDD
  - Exempel
  - Feature: Division
    - In order to avoid silly mistakes
    - Cashiers must be able to calculate a fraction
    - Scenario: Regular numbers
      - * I have entered 3 into the calculator
      - * I have entered 2 into the calculator
      - * I press divide
      - * the result should be 1.5 on the screen

Cucumber

- Källor
  - The RSpec book
    - Främst Rspec-kapitlet (Kap 10-11 i min utgåva)

RSpec

Nästa gång

- Cucumber
- Lab 3

- Innan dess:
  - Installera rspec gem (gem install rspec)
  - Gå in på http://rspec.info
  - Kör igenom deras bowling_spec - exempel
  - Säkerställ att allting verkar fungera
  - require ”bowling” => require ”./bowling.rb”

- ...

Fin