Översikt

- TDD.
- BDD.

.eorec

Lektion 1: Test-Driven Development

- Testa mjukvara.
- Test-Driven Development(TDD).
- Bra test hittar fel.
- Hur programmerare vanligtvis testar.
- Testa först.
- Arbetsflöde, testprincip och värden.
- Assertions.
- Testprincip – Automatisk testing.
- Kod för testing skrivs först
- Regelverk.
- Refactoring

.eorec

Lektion 1: Test-Driven Development (forts.)

- Testinformation.
- Verktyg.

.eavec

Testa mjukvara

"Testing is the process in which a (probably unfinished) program is executed with the goal to find error." [Myers 75?]

"Testing can only show the presence of error, never their absence." [Dijkstra 6?]

Via testning så kan vi inte bevisa att ett program är felfritt eller korrekt.

Kan dock ge oss ett förtroende för det arbete vi har gjort.

.eavec

Test-Driven Development

- Test-Driven Development =TDD, är en viktig del inom agil utveckling.
- Introducerades 2002 av Kent Beck, accepterad och rekommenderad teknik vid programmering, dock med förbehåll.
- TDD förespråkar att efter initial planering systemdesing, kravställning etc, börjar skriva testfall, före implementering.

.eavec

Bra test hittar fel

- Programmerare testar ofta, för att visa att deras program fungerar.
- De vill visa sig själv och andra att de har lyckats.
- För att göra detta så undviker man (medvetet eller omedvetet) problem. Detta leder till:
  - Att testning inte görs ordentligt.
  - Att folk inte litar på testad kod.

Hur programmerare vanligtvis testar

- Få test:
  - Största delen av programkod testas inte.
- Sen testning:
  - Koden blir lätt för svår att testa.
- Defensiva tester:
  - De mest kritiska partierna i programkod, testas inte.
- Osystematisk testning:
  - Gamla tester körs inte om när programkod är rättad.
- Om fel hittas, kan orsaken till detta vara svår att härleda.

Testa först

- Fokus på test:
  - Skriv test innan implementering.
- Övergripande mål:
  - Enkel programkod som fungerar.
- Interativ och inkrimentell process:
  1. Få något lite att fungera först.
  2. Rensa upp i koden (refactoring).
  3. Upprepa.
- Om något går fel, är problemet relaterat till det sista som man har gjort.
- Man kan lite på den kod, som har konstruerats.

Arbetsflöde, testprincip och värden

- TDD definieras av ett arbetsflöde (rytm), test principer och värden.
- Rytmen beskriver fem steg som man snabbt itererar över.
- Testprinciper hjälper oss bestämma vad vi ska göra härnäst i varje steg i arbetsflödet.
- Värdena beskriver den övergripande filosofin som delas mellan XP och Agile.
- Mjukvara definieras av programkod.
- Ta små steg.
- Behåll fokus.

.easec

Arbetsflöde

1. Lägg till ett nytt test:
   a) Välj ett test från testlistan.
   b) Implementera testet.
2. Kör alla tester och se om det nya testet misslyckas.
3. Skriv så mycket programkod som behövs för att testet skall lyckas.
4. Kör alla tester och se om de lyckas.
5. Gör refactoring på programkoden (kontrollera med dina tester att allt fungerar).

.easec

Testprincip – Automatisk testning

- Automatisk testning – seriös testning kan inte göras manuellt:
  - Implementerar alla tester.
  - Håller reda på alla tester.
  - Kör alla tester upprepade gånger.
  - Kontrollerar alla testresultat mot förväntade värden.

.easec

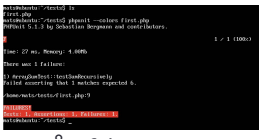
Assertions

- Kända från teoretisk datavetenskap (bevis av korrekthet av program):
  - Pre- och postconditions.
  - Invariants.
- Design-by-contact:
  - Preconditions: villkor (assertions) som måste vara sanna när en metod anropas.
  - Postconditions: villkor som måste vara sanna efter att en metod har avslutas korrekt.

.easec

Kod för testing skrivs först

- Kod för testning skrivs innan kod för applikationen.
- Kod som skrivs för test, läggs i separata filer.



- När programkod börjar utvecklas, testas utveckling mot testkod.
- Om något går fel, får vi oftast tips om vart det har gått fel.

.easec

Regelverk

- I TDD används unit-tests, dessa används för att undersöka en mindre isolerad del i systemet.
- Regelverk:
  - du får inte skriva produktionskod, om det inte finns ett fallerande test som garanterar detta.
  - i ett unit-test, får det inte vara mer programkod än nödvändigt, för att få den att fallera.
  - du får inte skriva mer produktionskod, än vad som är nödvändigt för att få ett test som har fallerat att gå igenom.

.easec

Refactoring

- Refactoring är processen att förändra källkoden utan att förändra dess externa beteende.
- Mål: förbättrad kodkvalitet (glöm inte bort testerna).
- Tecken på att det behövs:
  - Duplicerad kod.
  - Kod som är svår att förstå.
  - Andra problem.
- IDE:s kan hjälpa till med detta.

.easec

Testinformation

- Vilken information skall man använda i sina tester? Välj en liten mängd information där varje värde representerar en konceptuell aspekt eller speciell beräkning.
- Vilken information skall man använda?
  - Använd information som gör tester lätt att läsa och förstå. Skillnader i informationen skall vara meningsfull.
  - Om det inte finns konceptuell skillnad mellan 1 och 2, använd 1.

.easec

Verktyg

- OpenSource:
  - PHPUnit.
  - Codeception.
  - SimpleTest.
  - StoryPlayer.
  - Atoum.
  - Selenium.
  - m fl.
- Windows .NET:
  - inbyggda funktioner i Visual Studio.
  - tilläggsprodukter.

.easec

Lektion 2: Behavior-Driven Development

- Behavior-Driven Development (BDD).
- Fördelar med BDD.
- Fokus på.
- Bygga brygga.
- Samma vokabulär.
- GettingTheWordsRight.
- Specifikation för objektet.
- Kommunikation genom exempel.
- Scenarios, given-when-then.
- Scenarios, exempel.
- Färdig kod.

.easec

Behavior-Driven Development (BDD)

- Är en utvecklingsteknik som har sitt ursprung från Test-Driven Development (TDD).
- BDD använder sig av ett enkelt språk, domain-specific scripting language (DSL), för att konvertera satser till exekverbara tester.
- Resultatet kommer närmre kritera för acceptans för en given funktion, testet används för att validera funktionalitet.
- Naturlig utökning av TDD.

.easec

Behavior-Driven Development (BDD) (forts.)

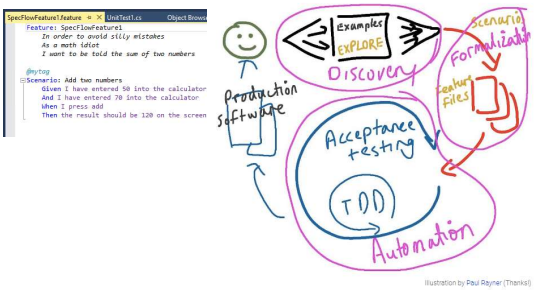


Illustration by Paul Rayner (Thanks!)

.easec

Fördelar med BDD

- Allt utvecklingsarbete kan spåras direkt till mål för utvecklingen.
- Utvecklingen svarar på vad användare behöver. Nöjda kunder = bra affär.
- Effektiv prioritering – kritiska funktioner levereras först.
- Alla involverade, delar samma uppfattning om projektet och kan vara delaktiga i kommunikation kring projektet.
- Delat språk ser till att alla har grundlig insyn i projektets utveckling.

.easec

Fördelar med BDD (forts.)

- Resultatet av mjukvarudesign matchar existerande och stödjer kommande behov för företaget.
- Förbättrar kvalité för programkoden och reducerar risker för projektet.

.easec

Fokus på

- BDD fokuserar på:
  - var skall process startas?
  - vad skall testas och vad skall inte testas?
  - hur mycket skall testas?
  - vad skall testet kallas?
  - varför fallerar testet?
  - ett nära samarbete med beställaren.
- Med TDD finns risk att tappa fokus, utvecklare skriver test utifrån egna förväntningar.
- I TDD kommer programkod skapas för att klara utvecklarens tester.

.easec

Bygga brygga

- BDD försöker bygga en brygga mellan olika synsätt på datorsystem, mellan de som skall använda systemet och de som utvecklar det.
- Utgår från TDD och influerat av Domain Driven Design.
- Fokuserat på att minimera hindren mellan specifikation, design, implementering och acceptans, för ett system.
- Ger möjlighet till inkrementel leverans av ett system och ger möjlighet till utvecklingsteam att snabbt ta till sig agila metoder.

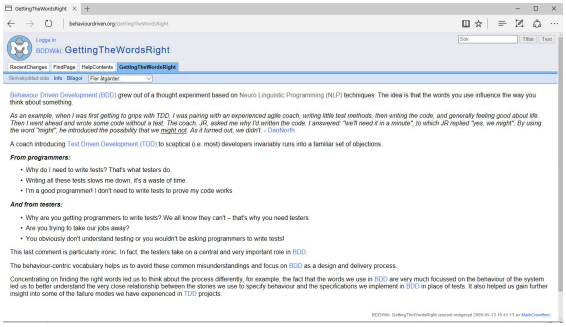
.easec

Samma vokabulär

- BDD bygger på ett väldigt specifikt och litet vokabulär, för att undvika missförstånd.
- Alla inblandade, företaget, utvecklare, testare, analytiker och chefer, använder samma ord.
- Ingen ny teknik, utan tänkt att sammanföra välfungerande tekniker under en gemensam och konsekvent terminologi.

.easec

GettingTheWordsRight



.easec

Specifikation för objektet

- I TDD används unit tests, i BDD kallas dessa inte för unit tests utan för specifikation för objektet.
- Dessa används för att visa hur mindre isolerade delar i systemet skall bete sig, istället för hur de skall testas.

.easec

Kommunikation genom exempel

- Funktion utvecklas från kommunikation mellan utvecklare och företaget, genom att arbeta med exempel.
- Exempel struktureras utifrån specifikt mönster: Context-Action-Outcome och skrivs i ett speciellt format som kallas Gherkin.
- Exempelvis:

```
Feature: Product basket
      In order to buy products
      As a customer
      I need to be able to put
      interesting products into a basket
```

.easec

Kommunikation genom exempel (forts.)

- Genom User Stories och samtal, har du kommit fram till följande:

```
Feature: Product basket
      In order to buy products
      As a customer
      I need to be able to put interesting
      products into a basket

Rules:
      - VAT is 20%
      - Delivery for basket under £10 is £3
      - Delivery for basket over £10 is £2
```

.easec

Scenarios, given-when-then

- Exempel leder fram till beteende, i BDD kallas dessa för scenarios.
- Scenario använder sig av termerna Given-When-Then.
- Syftet för **given**, är att sätta systemet i ett känt läge, innan det kan användas. Flera given, använd **also**.
- **When** används för att beskriva nyckelfunktioner som användaren utför.
- Syftet med **then**, är att observera resultatet.
- Istället för flera when, kan **and** och **but** användas.

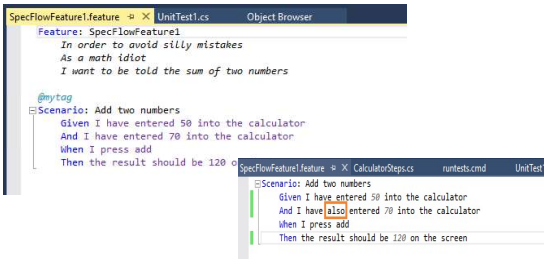
.eosec

Scenarios exempel

```
Feature: Product basket
  In order to buy products
  As a customer
  I need to be able to put interesting products into a basket

  Rules:
  - VAT is 20%
  - Delivery for basket under £10 is £3
  - Delivery for basket over £10 is £2

  Scenario: Buying a single product under £10
    Given there is a "Sith Lord Lightsaber", which costs £5
    When I add the "Sith Lord Lightsaber" to the basket
    Then I should have 1 product in the basket
    And the overall basket price should be £9

  Scenario: Buying a single product over £10
    Given there is a "Sith Lord Lightsaber", which costs £15
    When I add the "Sith Lord Lightsaber" to the basket
    Then I should have 1 product in the basket
    And the overall basket price should be £20
```

.eosec

Scenarios exempel (forts.)



.eosec

Scenarios exempel (forts.)

```
Feature: Score Calculation
  In order to know my performance
  As a player
  I want the system to calculate my total score

Scenario: Gutter game
  Given a new bowling game
  When all of my balls are landing in the gutter
  Then my total score should be 0

Scenario: Single Pin
  Given a new bowling game
  When I've hit exactly 1 pin
  Then my total score should be 1
```

```
[Binding]
public class BowlingSteps
{
    private Game _game;

    [Given(@"a new bowling game")]
    public void GivenANewBowlingGame()
    {
        _game = new Game();
    }

    [When(@"all of my balls are landing in the gutter")]
    public void WhenAllOfMyBallsAreLandingInTheGutter()
    {
        _game.Frames = "00000000000000000000";
    }

    [When(@"I've hit exactly 1 pin")]
    public void When1PinIsHit()
    {
        _game.Frames = "10000000000000000000";
    }

    [Then(@"my total score should be (\d+)")]
    public void ThenMyTotalScoreShouldBe(int score)
    {
        Assert.AreEqual(score, _game.Score);
    }
}
```

.easec

Färdig kod

```
// features/bootstrap/Basket.php

final class Basket implements \Countable
{
    private $shelf;
    private $products;
    private $productsPrice = 0.0;

    public function __construct(Shelf $shelf)
    {
        $this->shelf = $shelf;
    }

    public function addProduct($product)
    {
        $this->products[] = $product;
        $this->productsPrice += $this->shelf->getProductPrice($product);
    }

    public function getTotalPrice()
    {
        return $this->productsPrice
            + ($this->productsPrice * 0.2)
            + ($this->productsPrice > 10 ? 2.0 : 3.0);
    }

    public function count()
    {
        return count($this->products);
    }
}
```

```
namespace UnitTestProject1
{
    [Binding]
    public class CalculatorSteps
    {
        private Calculator calculator = new Calculator();
        private int result { get; set; }
        [Given(@"I have entered (.*) into the calculator")]
        public void GivenIHaveEnteredIntoTheCalculator(int number)
        {
            calculator.FirstNumber = number;
        }

        [Given(@"I have also entered (.*) into the calculator")]
        public void GivenIHaveAlsoEnteredIntoTheCalculator(int number)
        {
            calculator.SecondNumber = number;
        }

        [When(@"I press add")]

        public void WhenIPressAdd()
        {
            result = calculator.Add();
        }

        [Then(@"the result should be (.*) on the screen")]
        public void ThenTheResultShouldBeOnTheScreen(int expectedResult)
        {
            Assert.AreEqual(expectedResult, result);
        }
    }
}
```

.easec

Två inriktningar

- För BBD, finns det två inriktningar:
  - xBehave.
  - xSpec.
- SpecFlow (likt Cucumber) är ett exempel för xBehave. Bygger på acceptanskriterier, men tillhandarhåller ett lätt sätt att bygga unit tests.
- mSpec är ett exempel på xSpec.

.easec

Övning arbeta med BDD

.easec

Repetitionsfrågor

.easec