



Översikt

- Arbeta med Unit-test.
- Implementera strategi för hantering av undantag

.eerec

Lektion 1: Arbeta med Unit tester

- Unit-tester.
- Tester.
- Tre faser.
- Automatiskt testning.
- Test Driven Development.
- Principer för Test Driven Development.
- Loosely Coupled Application.
- Tightly Coupled Application.
- Skriva unit-test för komponenter i MVC.
- Använda unit-test.

.eerec

Lektion 1: Arbeta med Unit tester (forts.)

- Specificera rätt mål.
- Mocking Framework.

.eeec

Unit-tester

- Lektionen fokuserar på unit-tester, denna typ av tester utförs under utvecklingen.
- Andra typer av tester, t ex tester för integration eller acceptans kan bara utföras när ett antal komponenter finns på plats.
- I MVC webb applikation kan unit-test instansiera klasser för model eller controllers, kalla på dess metoder och händelser.
- Du kan skriva unit-test som skapar ett objekt för produkt, testar att metoden köp returnerar chart objekt.

.eeec

Tester

- *Unit-test*, kontrollerar små aspekter för funktionen. Exempelvis: att viss metod returnerar rätt värde. Genom att arbeta med flera unit-tester kan du försäkra dig om att dessa täcker hela din applikation.
- *Integrationstest*, kontrollerar hur två eller flera komponenter fungerar tillsammans, kan även användas för att testa hela sidan.
- *Acceptanstest*, fokus på funktion eller tekniska krav som måste fungera för att beställaren skall acceptera.

.eeec

Tre faser

- **Arrange:**
 - Testet skapas instans av klass som skall testas, tilldelar även egenskaper som krävs och skapar alla de objekt som behövs.
 - Bara egenskaper och objekt som behövs för testet skapas.
- **Act:**
 - Testet har den funktion som skall kontrolleras.
 - Oftast en enda procedur och sparar resultat i en variabel.

.eerec

Tre faser (forts.)

- **Assert:**
 - Test kontrollerar resultat med förväntat resultat.
 - Matchas resultatet har programkod passerat testet, om inte kommer fel att visas.
- Efter skapandet av test, kommer test att köras automatiskt när du startar med debugging.

.eerec

Unit-test hjälper till att fixa buggar

- Eftersom unit-test testar en liten och specifik del av programkoden, är det lätt att diagnostisera problem om ett test skulle falla.
- Vanligtvis använder vi testet på en enda klass och isolerar denna från de andra klasserna där det är möjligt.
- Om inblandning av annan klass behövs, skall ett minimerat antal klasser skapas i fasen Arrange.
- Att strukturera på detta sätt, ger möjlighet att fixa fel snabbt pga att källor för fel är litet.

.eerec

Unit-test hjälper till att fixa buggar (forts.)

- Unit-test skall kontrollera programkod som du skriver och inte infrastruktur som eventuell programkod är beroende av.
- Exempelvis:
 - Unit-test skall köras utan att ansluta till databas eller webbtjänst eftersom nätverksproblem eller att tjänst är nere kommer att orsaka fel.
 - Genom att arbeta på detta sätt, kan du särskilja på buggar från programkod, vilket kan fixas med omskrivning av programkod, från buggar från infrastruktur, som du kan fixa till genom att ändra hårdvara, omkonfiguration av webbserver eller anslutningssträng.

.eerec

Automatiskt testning

- Det är viktigt att förstå, att efter definition av unit-test, kan tester köras som kvickt och lätt under hela livscykeln för projektet.
- Visual Studio kör om testet automatiskt när ett projekt startas med debugging. Testet kan även köras manuellt.
- Detta är viktigt, för ny programkod kan orsaka buggar var som helst i utvecklingsfasen.

.eerec

Automatiskt testing -exempel

- Ett exempelprojekt har tre stycken faser, ett unit-test definierat för fas 1 hjälper dig att observera problem som kan uppstå i fas 2 och fas 3, som annars kanske hade passerat ouppmärksammat.
- Fas 1, enkel controller som innehåller två händelser, Index() och Details(), bägge returnerar views. Ett unit-test skrivs, Test_Index_Action, testet testar om händelsen Index fungerar med en integer parameter. När händelsen anropas med integer, skall view visas det antalet produkter som definieras av integer. Testet kommer att passeras.

.eerec

Automatiskt testing –exempel (forts.)

- Fas 2, utvecklare modifierar händelsen så den returnerar partial view, för att följa namnkonvention så ändras namnet för händelse till `_index`. Testet kommer att falla, eftersom namnet har ändrats. Utvecklare modifierar testet så testet kallar på den namnändrade händelsen istället och test kommer att passeras.
- Fas 3, utvecklare modifierar händelsen `Index` så att en LINQ-fråga används, för att uppfylla ett nytt krav. Men utvecklare kommer att skriva fel och händelsen returnerar inga poster. Testet kommer att falla.



Automatiskt testing –exempel (forts.)

- Unit-test hjälper utvecklare med att fokusera på lösningar för att gå igenom testet och på sätt ser till att ny funktionalitet inte orsakar fel i programkod som är skriven tidigare.
- Unit-test kan användas med vilken projektmetodik som helst, för att uppmärksamma buggar och förbättrar programkvalité.



Test Driven Development

- I Test Driven Development (TDD) är unit-test en central funktion.
- TDD är oftast beskriven som en separat utvecklingsmetodik, en del betraktar det som en viktig del i metodiken Extreme Programming.



Test Driven Development (forts.)

- I TDD arbetar utvecklare med korta cyklar för att implementera krav för en funktion:
 - *Skriv test*, utvecklare börjar med att skriva ett test, kräver att utvecklaren förstår kraven, dessa fås från user stories och user cases. Därefter körs testet, eftersom ingen programkod är skriven kommer testet att falla.
 - *Passera testet*, skriver snabbt enkel programkod som passerar testet.
 - *Refactor*, rensar upp i koden, duplicerad kod tas bort, förbättrad läsbar. Kontinuerligt körs testet om.

.eerec

Principer för Test Driven Development

- *Skriv test innan programkod*, test skapas som första steg, innan programkod skrivs. Testet måste falla första gången. Testet är egentligen en specifikation för funktionalitet som du bygger. Genom att skriva testet först, försäkra du dig om att du har förstått kraven.
- *Gå framåt i små steg*, genom att bryta upp en stor applikation i små bitar av funktionalitet ökar produktiviteten. Utvecklare kan lösa funktion snabbt och få förståelse omgivningen de körs i.
- *Skriv bara tillräckligt med programkod för att passera testet*, även om det är frestande att skriva in ytterligare funktionalitet, så skriv bara programkod för att testet skall gå igenom.

.eerec

Principer för Test Driven Development (forts.)

- Utvecklare kan referera till testerna som exempel på hur klasser och metoder skall användas.
- Testerna kan även användas för att demonstrera dessa.

.eerec

Loosely Coupled Application

- Konceptet att skriva applikation så att alla klasser kan fungera oberoende av varandra, kallas för Loosely Coupled Application.
- I MVC har vi tre delar:
 - Model, klasser och logik.
 - View, gränssnitt som baseras på information i Model.
 - Controller, svarar på förfrågningar och väljer View.
- De tre delar i MVC kan förändras individuellt utan att påverka de andra delarna.
- Lätt att skriva unit-test, då det är lätt att ersätta full funktionalitet i en klass, med en förenklad variant.

.eerec

Tightly Coupled Application

- ASP.NET WebForms kallas för Tightly Coupled Application.
- Varje .aspx sida är beroende av underliggande programkod.
- Är svårt att testa.

.eerec

Skriva unit-test för komponenter i MVC

- Modeller är enkla att testa, pga dessa är separata klasser. Dessa klasser kan instansieras och konfigureras under fasen arrange.
- Controllers är en enkel klass som du kan testa, men kan vara komplex om du använder information, speciellt om denna information kommer från en databas.
- Unit-test skall inte blanda in infrastruktur, skall bara testa programkod.
- Eventuell information som behövs, måste finnas i minnet. Detta kallas för test double.

.eerec

Använda unit-test

- Lägg till ett nytt projekt till din befintliga programkod, baserat på mallen Unit Test Project.
- Referenser måste läggas till så programkod får tillgång till klasser i ditt MVC-projekt.
- I testprojektet markeras klasser med [TestClass], metoder med [TestMethod].
- Unit-test returnera vanligtvis void, men kalla på metod i Assetklass, t ex Assert.AreEqual för att kontrollera resultatet.

.eerec

Specificera rätt mål

- När test double används för i unit-test för att testa controllers, är det viktigt att du har rätt mål, i två scenarions: unit-test och utanför testet.
- Unit-test används test double, i din programkod används Entity Framework.
- För att specificera rätt förråd att användas för test, använd controller constructor.
- Du kan även använda Inversion of Control container, ett ramverk som kan läggs till din applikation och som instansierar klasser.

.eerec

Mocking Framework

- Under faste Arrange, måste testinformation skapas.
- Du kan göra detta manuellt, eller genom att arbeta med ramverk för detta.
- Mocking, ger möjlighet att isolera programkod under text, eventuella beroende förutsätts att dessa fungerar.

.eerec

Övning Skapa Unit-Test



Lektion 2: Implementera strategi för hantering av undantag

- Hantera undantag.
- OnException.
- Felsida som används.
- Felsida som används -inställningar.
- Http Errors.
- Visual Studio IntelliTrace.
- Logga undantag.
- Monitorering av hälsa.
- Monitorering av hälsa -kategorier.

.eeec

Hantera undantag

- Error är en oväntad händelse som förhindrar slutförandet av applikationen.
- Ett undantag uppstår, undantaget är ett objekt av klass som ärver från System.Exception.
- Finns många andra undantag eller fel som kan uppstå i ASP.NET MVC, fel visas på en webbsida.
- Standardsida för fel, kan ersätta med egen.

.eeec

Hantera undantag (forts.)

- Vanligaste metoden är att använda block med try/catch.
- En annan teknik är att göra override på metoden OnException.
- En tredje metod är att använda [HandleError].

```
[HandleError(ExceptionType = typeof(NotImplementedException),  
View = "NotImplemented")]  
[HandleError]  
Public ActionResult Index()  
{  
    // Programkod här  
}
```

.cs/ce

OnException

- Metoden OnException är definierad i basklass för Controller.
- När du gör override på denna i den egna controller, kommer undantag inte att hanteras av try/catch-block.
- Genom att använda OnException, kan du arbeta med hantering av undantag, utan att använda try/catch.

.cs/ce

Felsida som används

- ASP.NET-baserade applikationer använder en standard HTML-sida för fel.
- Du kan skapa en egen, genom att ändra i filen web.config, använd web.config-filen som ligger överst i din struktur.

```
<system.web>  
    <customErrors mode="On" defaultRedirect="CustomError" />  
</system.web>
```

.cs/ce

Felsida som används -inställningar

- *Off*, fel som inte är hanterade visas med en standardsida.
- *On*, fel som inte är hanterade visas på sida specificerad med attributet `defaultRedirect`.
- *RemoteOnly*, visas olika för datorer på insidan eller utsidan. För datorer på utsidan, visas sidan som specificeras i `defaultRedirect`.
- View skall finnas i katalogen `View/Shared`.

.eeec

Http Errors

- Visa Http Errors kan inte fångas med skräddarsydda MVC views, specificerade i `web.config` eller med `[HandleError]`.
- För denna typ av fel kan elementet `<error>` användas i `web.config`.

```
<customError mode="On" defaultRedirect="CustomError" >  
  <error statusCode="500" redirect="~/Error/Error500.html" />  
</customError>
```

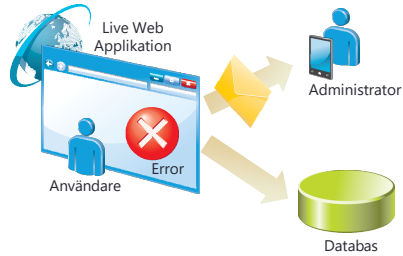
.eeec

Visual Studio IntelliTrace

- IntelliTrace kan användas till:
 - Förbättra arbetsflödet för avlusning, genom att spela in när kod exekveras.
 - Undersöka händelser som har innan nuvarande exekveringspunkt.
 - Spara händelser i en fil när du startar avlusningsläget.

.eeec

Logga undantag



.carec

Monitorering av hälsa

- Följande funktioner, för att monitorera hälsa för din applikation, kan användas:
 - EventCategories, hjälper till att skäddarsy kategorier som funktionen monitorerar.
 - Health providers, skräddarsyr plats där funktionen lagrar händelsedetaljer.
- Elementet <healthMonitoring> används i filen web.config.

.carec

Monitorering av hälsa -kategorier

Händelsekategori	Beskrivning
Application Lifetime Events	Inkluderar t ex när applikation startas eller avslutas.
Audit Events	Inkluderar t ex logon, logoff eller fel vid inloggningsprocessen.
Error Events	Inkluderar undantag i .NET Framework eller HTTP error.
Request Processing Events	Inkluderar händelser som sker för controllers, models och views.
Heartbeat	Används för att kontrollera om applikation körs eller inte.

.carec

Övning Logga till Application Log



Repetitionsfrågor

