# Introduksjonskurs i TDD i .NET

**Mål**
*Du skal lære nok om TDD til at du har et grunnlag for å jobbe videre med TDD i prosjekter som enten bruker det eller som ønsker å starte å bruke det.*

**Forkunnskaper**
Det er en fordel med grunnleggende kunnskaper i .NET, C#, og Visual Studio.

**Varighet**
Kurset har blitt kjørt som en 3-timers sesjon på Skuret. Dette var for lite til å gjennomføre samtlige av totalt 4 oppgaver. Med TDD er det slik at jo mer tid man investerer, jo bedre er det.

**Krav**
Visual Studio 2010 eller nyere må være installert på maskinen. Dersom dette ikke er installert sjekk MSDN siden i Confluence med instruks om hvordan du kan laste ned og installere det.

**Ressurser**
Eksempelkode og presentasjon ligger under http://github.com/bekk/dotnetkurs.
Dette er et privat repo under BEKK på Github og krever derfor at du blir lagt til i Team dotnetkurs for å få tilgang. Brukernavn og passord til bekkadmin på github finnes på Confluence.

**Introduksjon**
Ideen med TDD er at man skal skrive testene før man skriver selve implementasjonen. Presentasjonen som ligger på github viser red-green-refactor prinsippet i tillegg til å innholde oppgavene som er tiltenkt dette kurset. Det anbefales å bla gå gjennom de delene av presentasjonen som er lagt før hver oppgave, før man starter på oppgaven.

**Komme i gang**
I oppgavene som er laget for dette kurset er hovedvekten lagt på at man skal få trening i å skrive testene først. Det er vanlig å ha testene til en implementasjon i et eget prosjekt innenfor en Solution. For oppgavene i kurset kan man derfor starte med å opprette et nytt prosjekt i Visual Studio, gjerne av typen Console Application eller Class Library. Det første prosjektet skal være selve implementasjonen. Når det er gjort oppretter man et prosjekt til i samme solution (høyreklikk på solution og velg Add New Project). Dette prosjektet skal være test-prosjektet, og skal da være av typen Test Project. Da skal det være klart for å skrive første test og du bør ha en struktur som ser slik ut:

<Solution>
    <ClassLibraryProject>
    <TestProject>

For at testprosjektet skal få tilgang til det du ønsker å teste, må man legge inn en referanse fra test prosjektet. Høyreklikk på *References* på testprosjektet og velg *Add*

*Reference*. Fra dialogboksen som kommer opp velger du å legge til referanse til prosjektet som du skal teste.

Visual Studio har sitt eget testrammeverk, MSTest. Ved å klikke på Test->Windows kan du få frem både Test View og Test Result. Test View gir deg en oversikt over alle testene innenfor den Solution som du har åpen, du kan markere tester å kjøre disse. Test Result viser resultatene av et gitt antall tester som du har kjørt. Her kan du få callstack fra tester som feiler og kjøre de på nytt.

Visual Studio har en rekke hurtigtaster, og for testing benytter man Ctrl+R,T når man står i kontekst av en test som vil gjør at testen starter.

**Oppbygning av en test**

En test ser ut som en vanlig metode i C#, den er void, tar ingen parametere og er dekorert med en attributt for å signalisere til testrammeverket at det er en test. Hva attributen heter avhenger av testrammeverket, f.eks i MSTest heter den [TestMethod].
Videre er det blitt mer og mer vanlig å dele testen inn i tre deler; arrange, act og assert. Når det kommer til navngiving av testene varierer dette ofte fra prosjekt til prosjekt, det viktigste er at navnet er beskrivende nok til at du kan lese navnet på testen og raskt forstå hva som har gått galt dersom testen feiler – og selvfølgelig finnes det en standard på prosjektet så følger vi den.

**Tips og triks**

*”Do the simplest thing that could possibly work”*
Når man har skrevet en test så skal man bare gjøre så mange endringer i koden at testen faktisk blir grønn. Ønsker man å gjøre mer, må man skrive en ny test som feiler, det gjelder når man endrer funksjonaliteten, ikke for opprydding/restrukturering av koden.

*En test skal bare teste èn ting*
En god huskeregel er at det skal være èn assert per test, men man vil komme borti situasjoner der man trenger to eller flere assert for at testen skal gi mening. Huskeregelen man alltid kan tenke på er at testen kun skal teste èn ting – dersom du oppdager at testen gjør mer enn èn test skal denne splittes i flere tester. Bakgrunnen for dette er at når en test feiler skal det være raskt å sette seg inn i hva som gjør at den feiler. Er det en test som tester flere ting kan det være vanskelig å vite hva som faktisk feiler uten å bruke mye tid på å forstå både testoppsett og gjennomføring.

*ReSharper Live Template*
Dersom man har en fast form på testene, f.eks navnet skal alltid inneholde navn på metode man tester og forventet resultat kan man sette opp live templates i ReSharper som hjelper deg å sikre at testene blir ”like” hver gang. Se mer om ReSharper Live Templates her: http://www.jetbrains.com/resharper/features/codeTemplate.html