Test Driven Development

Agenda

- Tradisjonell koding vs. TDD
- Red, Green Refactor
- Kodeeksempel
- Fordeler og ulemper med TDD
- Anbefalinger

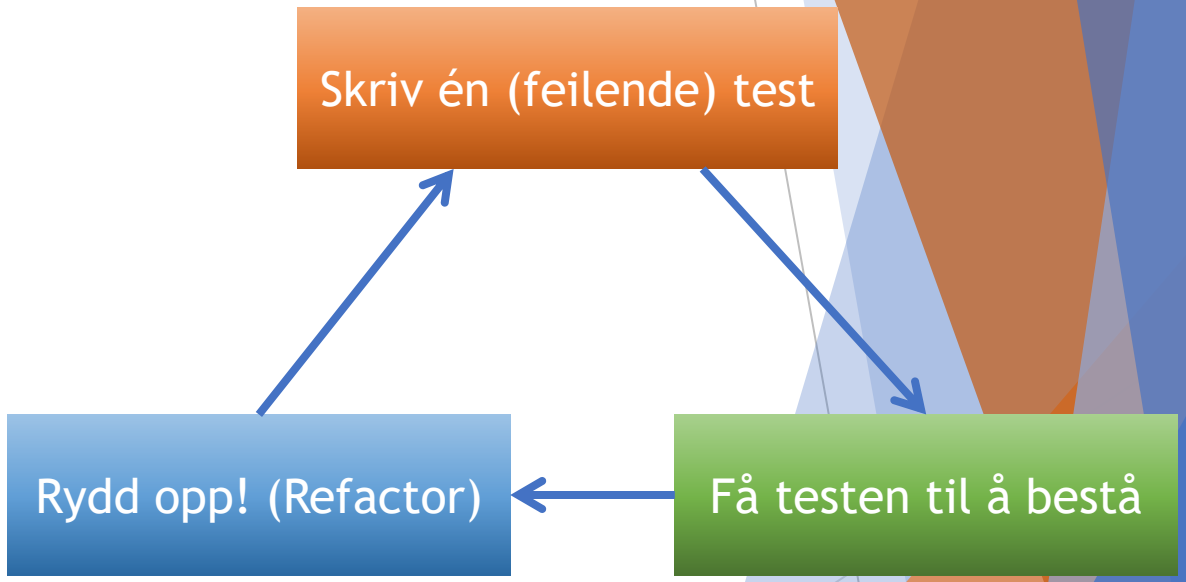
Tradisjonell koding

- Vi skriver koden først og tester i etterkant
  - Feilsøker og retter opp hvis testene feiler
- Altså: Testene blir en siste sjekk i stedet for en del av prosessen
- Dette kan medføre ulemper
  - Bugs blir typisk oppdaget sent og kan være slitsomme å fikse
  - Testene blir rasket sammen nærmere deadlines
  - Testene sjekker typisk bare at resultatet er riktig, uavhengig av om koden er overkomplisert
  - Vanskelig å vite om testene faktisk dekker hele koden

Test Driven Development (TDD)

- Test Driven Development (TDD) er en alternativ måte å skrive kode på

- Snur på «oppskriften» (Red, Green, Refactor)
  1. Vi starter med å skrive én test (basert på krav for feature-funksjonalitet)
  2. Koden skrives for å oppfylle testen på enklest mulig måte (uten å ødelegge andre tester)
     1. Dette kan noen ganger avsløre nye krav
  3. Etter å ha oppfylt testen rydder vi opp basert på en rekke prinsipper (Refaktorering)
  4. Hvis flere krav eller nyanser: Gå tilbake til 1.



Quick! Get to green!

- Det finnes en rekke strategier for hvordan man raskest får testen til å passere

- Kompliering – Generer klasser og metoder som testen krever
- Obvious Implementation – Slik vi vanligvis koder logikk
  - Når vi åpenbart ser neste steg for å få testen til å passere skriver vi en ekte kode-implementasjon
- Fake it!
  - Hvis noe feiler eller vi ikke åpenbart ser veien videre kan vi sette ikke-dynamiske verdier for å få testene til å funke, og deretter refaktorere
- Triangulation
  - Vi generaliserer koden (bare) hvis 2 eller flere tester/assertions krever det
  - Nyttig hvis man ikke intuitivt klarer å se en nødvendighet for abstraksjon
    - Test for flere tilfeller og se om koden fortsette å passere testen

Refaktorering

- Refaktorering: Endre en bit med kode uten å endre dens oppførsel
  - Gjøres kun hvis testene er grønne
  - Handler mye om å være kjent med prinsippene bak god kode
  - Det kreves dessverre en del erfaring for å være god på dette

Refaktorering

- Generelle retningslinjer for refaktorering:

- Forsøk å begrense kodelinjer
  - Kan jeg skrive dette mer konsist (uten å betydelig miste lesbarhet)?
- Unngå duplisert kode – Generaliser/abstraher
  - Reduser antall If-tester
  - Ny metode
  - Ny klasse
  - Arv
- Unngå unødvendig kode
  - Ubrukte metoder
- Skriv om eksisterende tester – Evaluer om testene fremdeles representerer ønsket resultat

Kodeeksempel – Telle karakterer

- Vi ønsker å ha en enhet med navn ExamTool
  - Skal blant annet skal kunne telle antallet av en spesifisert karakter (A-F) inneholdt i en gitt liste med forskjellige karakterer

- Bruk TDD til å utvikle denne funksjonaliteten
  - 1. Red
  - 2. Green
  - 3. Refactor
  - 4. Tilbake til 1.

Test Driven Development - Fordeler

- Lar utvikleren først fokusere på hva resultatet skal være (ønsket bruk)
- Fører naturlig til at koden deles opp i små testbare enheter
- Fordi koden dannes basert på testene kan du være meget sikker på at hele koden er testet
- Unngår i prinsipp overkomplisert kode
  - Vi skriver bare den koden vi må for å oppfylle testene
- Testene fungerer essensielt som en dokumentasjon på koden
  - Men det er likevel viktig å kommentere
- Man oppdager bugs raskere fordi man tester iterativt gjennom kode-prosessen
- Refaktorering er trygt fordi testene vil feile hvis effekten endres

Test Driven Development – Ulemper / bekymringer

- Det er likevel ikke bare solskinn og regnbuer med TDD

- Det er en uvant og ofte lite intuitiv kodestrategi for mange utviklere
  - Føles som «koding, men med ekstra steg»
- Man ungår fort å gjøre store (men nødvendige) endringer fordi tester vil feile
- Det blir fort et større fokus på implementasjonsdetaljer i stedet for det helhetlige problemet (I motsetning til tanken bak TDD)
- Man kan bli «lurt» til å redefinere problemet for å kunne lage enklere tester

Mine tanker og anbefalinger om TDD

- TDD har mange interessante fordeler, men...
  - Det passer ikke for alle og krever typisk mye tid for å bli vant til
- Bruk gjerne litt av tiden i prosjektet til å eksperimentere med TDD
  - Kan være en alternativ måte å programmere på som er meget nyttig når man står fast
  - Testene kommer som en del av prosessen
  - Banker inn dette med refaktorering (Viktig!)
  - ... Ikke et krav, men kan være komme på skriftlig eksamen
- Take-awayen med TDD er vel egentlig ...
  - Test ofte og mens du skriver kode – Reduser bugs og etterarbeid
    - Om du skriver en test før eller etter koden er opp til deg
  - Reflekter hyppig over hvor god koden er – Refaktorer underveis
    - Tenk ofte på duplisering, unødvendig kode, navngivning, antall linjer osv.
- Vil du vite mer? – Les «Test-Driven Development By Example» av Kent Beck