TNM094 – Medieteknisk kandidatprojekt

Refaktorering

Refaktorering

Refaktorering (*substantiv*): förändring som görs i den interna strukturen av programvara, för att göra den lättare att förstå och billigare att modifiera, utan att förändra dess observerbara beteende.

[Martin Fowler]

Refaktorering i systemutveckling

- Varför ska man refaktorera
  - Motverka kontinuerligt förfall av design
  - Anpassa design till ny funktionalitet
  - Rätt design gör det lättare och snabbare att programmera
  - Rätt design gör programmet mer robust
  - Refaktorering kan också avslöja fel
- När ska man refaktorera
  - Som en del av många aktiviteter:
    lägga till funktionalitet, fixa buggar, granska kod
  - Tredje gången gillt
  - När koden ”luktar illa” (“bad smell”)

[Martin Fowler]

“Dålig lukt”

- Oönskade egenskaper som kan identifieras
  - T ex duplicerad kod, långa funktioner, spekulativ generalisering
  - Inte *alltid* ett problem som *måste* tas bort
  - Fowler har en lista på 21 ”dåliga lukter”
- Refaktorering tar bort ”dålig lukt”
  - T ex dela upp metod, flytta metod, ersätt temporär med anrop, introducera parameter-objekt, kollapsa hierarki
  - Fowler har en lista på 72 refaktoreringar

[Martin Fowler]

Att genomföra refaktorering

1. Förstå koden (åtmintone dess syfte)
   - Läs kod och dokumentation, dokumentera, modellera
   - Identifiera design-mönster
2. Implementera test
3. Planera refaktoreringen
   - Skapa ny design och modellera vid behov
   - Använd nya design-mönster
4. Refaktorera
   - Använd CASE med automatisk refaktorering
   - Följ guider, flytta kod och komplettera vid behov

[Martin Fowler]

Lång metod

- Problemet
  - Långa metoder är svåra att läsa
  - Långa metoder kan inte köras partiellt
- Refaktorering
  - Bryt ut metod
    - Kod-fragment kan grupperas tillsammans
    - Förvandla fragmenten till metoder med beskrivande namn
  - Ersätt temp. med anrop
    - Temporära variabler håller resultat från beräkningar och anrop
    - Bryt ut beräkningar i metoder och ersätt temp. med anrop

[Martin Fowler]

Funktions-avund

- Problem
  - En metod intresserar sig mer för en annan klass än sin egen
  - Anropar en mängd get-funktioner för att göra sitt jobb
- Refaktorering
  - Flytta metod
    - En metod använder mer medlemmar i en annan klass
    - Skapa en ny metod i den andra klassen och flytta funktionaliteten
  - Bryt ut metod
    - En metod gör flera saker som vi vill kunna köra delar av
    - Bryt ut fragmenten till separata metoder med beskrivande namn
    - Låt den första metoden anropa de nya metoderna

[Martin Fowler]

Vägrad arvslott

- Problemet
  - En subklass vill inte ha vissa egenskaper som ärvs från basklassen
  - Detta betyder att klass-hierarkin är dålig
- Refaktorering
  - Bas-klassen ska bara innehålla vad som är gemensamt
  - Skapa en syskon-klass och gör följande:
  - Flytta ned metod
    - Funktion hos basklassen är bara relevant i vissa subklasser
    - Flytta ned den funktionen till de subklasserna
  - Flytta ned fält
    - Ett fält används bara av vissa subklasser
    - Flytta ned fältet till de subklasserna

[Martin Fowler]

När ska man inte refaktorera

- När du egentligen ska skriva om koden
  - Alternativt kan man kapsla in stora delar programkod
    → refaktorera sedan bit för bit
- Nära en deadline eller vid tajt schema
  - Fördelarna med refaktorering märks först efter deadline
- Innan nödvändiga test har implementerats
  - Test behövs för att säkerställa full funktionalitet
  - Ta bort test som gäller funktionalitet som inte längre behövs
  - Skriv nya test för att täcka tänkt funktionalitet

[Martin Fowler]

Att också tänka på

- Design
  - Planera design utifrån att man kommer att refaktorera
  - Tillåt enklare design tidigt i utvecklingen
  - Intern design kan vara enklare än för ett framework / API
- Prestanda-optimering
  - Ständigt prestanda-fokus leder till dålig kod
    (Bra prestanda kommer från anpassad design)
  - Skriv för läsbarhet och pålitlighet, inte för prestanda
  - Refaktorera när prestanda-test visar på problem
  - (Gäller inte nödvändigtvis inbäddade system och realtidssystem)

[Martin Fowler]