High cohesion – Low coupling

Objekt-orienterad programmering och design

Sólrún Halla Einarsdóttir & Alex Gerdes, 2018

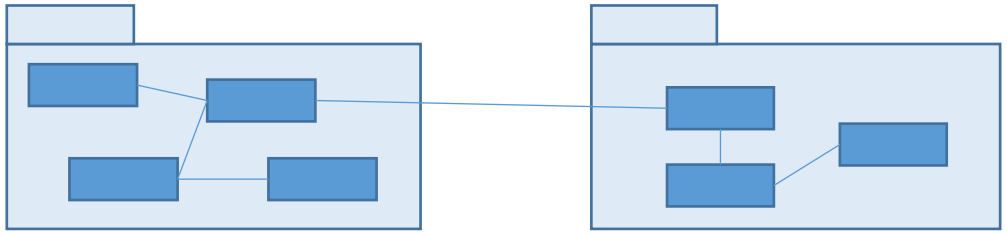
Modulär design

Fördelar med en *välgjord* modulär design:

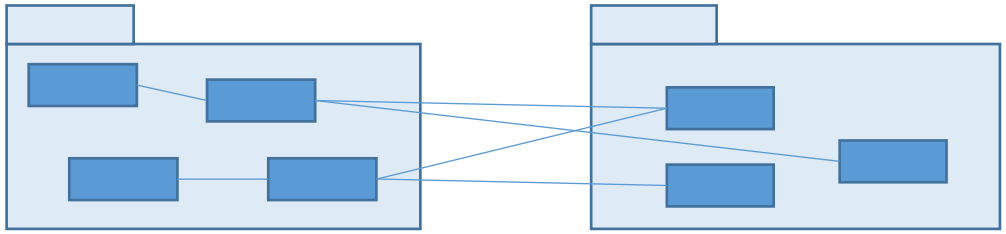
- Lätt att utvidga
- Moduler går att återanvända
- Uppdelning av ansvar
- Komplexiteten reduceras
- Moduler går att byta ut
- Tillåter parallell utveckling.

Cohesion

- *Cohesion* (”sammanhållning”) är ett mått på den inre sammanhållningen i en modul (e.g. metod, class, package, …).
  - Hur väl samverkar komponenterna inom modulen?
- Vi eftersträvar *high cohesion* – dvs när samtliga komponenter samverkar för att lösa modulens ansvarsområde, utan att behöva samverka med komponenter i andra moduler.
  - Hur autonom är modulen? Klarar den sitt uppdrag på egen hand?
  - Hur väldefinierat är modulens ansvarsområde?



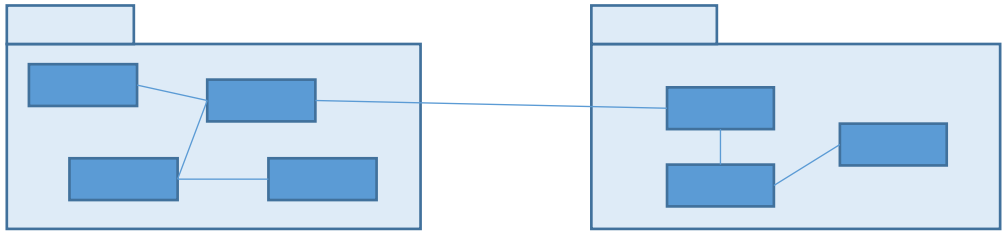
High cohesion



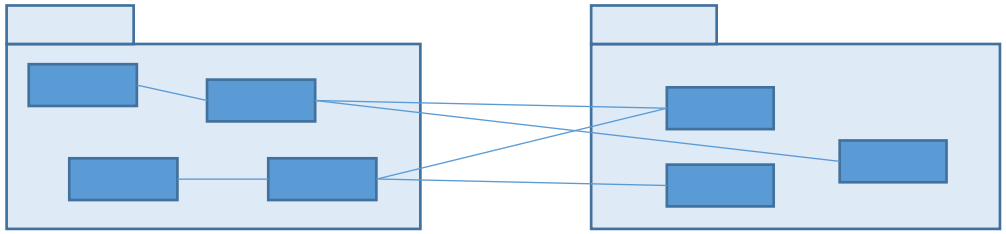
Low cohesion

Coupling

- *Coupling* (”sammanbindning”, ”koppling”) är ett mått på hur starkt beroendet är mellan två olika moduler.
  - Hur autonom är modulen? Klarar den sitt uppdrag på egen hand?
  - Hur väl avgränsad är modulen? Tillåter den andra moduler att bero på dess inre implementation?
- För att få en flexibel och modulär design måste ingående moduler vara så oberoende av varandra som möjligt. Vi eftersträvar *low coupling* mellan moduler.
  - Har vi starka kopplingar kommer förändringar i en modul framtvinga förändringar i andra moduler som är beroende av den – bryter mot OCP.



Low coupling



High coupling

Dependency Inversion Principle

Depend on abstractions, not on concrete implementations.

- Beroenden behövs! Men för en bra design gäller:
  - Minimera antalet beroenden.
  - Ha så lösa beroenden som möjligt (DIP).
  - Ha kontroll över beroendena.
    - Varje beroende ska vara avsiktligt.
    - Varje kopplingspunkt (dvs möjlighet att skapa beroenden) ska vara avsiktlig.
    - Varje kopplingspunkt ska ha ett väldefinierat gränssnitt.

Övning

- Börja från koden som finns att ladda ner från hemsidan.
- Koden för övningen är en utökning av tidigare kod för att rita polygoner; nu kan polygonerna röra på sig. Den nya koden bryter mot alla tänkbara principer – ert jobb är att analysera och förbättra den.
- Börja med att rita ett UML-diagram av designen. Kan ni identifiera några specifika problem?
- Diskutera och fundera över hur en bättre design skulle kunna se ut. Rita ett UML-diagram av denna bättre design.
- Vilka refactoring-steg behöver ni genomföra för att uppnå er bättre design? Think before you code!
- Refaktorera koden – men inte förrän ni har en uttalad plan!