TNM094 – Medietekniskt kandidatprojekt

Modul- och programdesign

Programdesign

- Syfte
  - Bestämma klasstruktur, gränssnitt, anrop, etc.
  - Kommunicera/synkronisera idéer mellan utvecklare
- Vad är programdesign / moduldesign
  - Design på lägsta nivån innan programkod
  - Hur varje modul ska fungera internt
  - Klasstrukturer, variabler, funktioner, abstraktion, anrop

[Pfleeger]

Design-strategier

- Oftast kombination av tekniker
  - Top-down – dekomposition, rekursiv uppdelning
  - Bottom-up – syntes av bitar som behövs
  - Outside-in – börja med gränssnittet
  - Slices – delfunktionalitet hela vägen
- Diskussioner och modellering

- Steg-för-steg-tekniker
  - Five-step UML – UML-baserade design-steg
  - ”Faking the rational design process”

[Pfleeger]

Diskussioner och modellering

- Programdesign är svårt men viktigt
  - hur vet man om en design kan implementeras bra?
  - använd modellering av
    - klasstruktur, arv, abstraktion
    - runtime-struktur, kommunikation, anropssekvenser
    - interna tillstånd och övergångar
  - resonera kring
    - är designen entydig eller är delar underförstådda?
    - vad vill vi ska hända och vad kommer att hända?
    - vad händer om något anrop blir fel?

[Pfleeger]

Principer för programdesign

- Syfte
  - Riktlinjer för bra design
- Sex dominanta principer (enl. Pfleeger)
  - Modularitet
  - Gränssnitt
  - Dölja information
  - Inkrementell utveckling
  - Abstraktion
  - Generalisering

[Pfleeger]

OO-specifika principer

- OO försöker koda in ”best practice”
  - Komposition, arv, polymorfism, dynamisk binding, etc.
  - Det är upp till utvecklaren att göra det bästa av det

- Viktiga principer
  - Funktionalitetsutbyte via arv eller komposition
  - Substitutability (utbytbarhet)
  - Law of Demeter

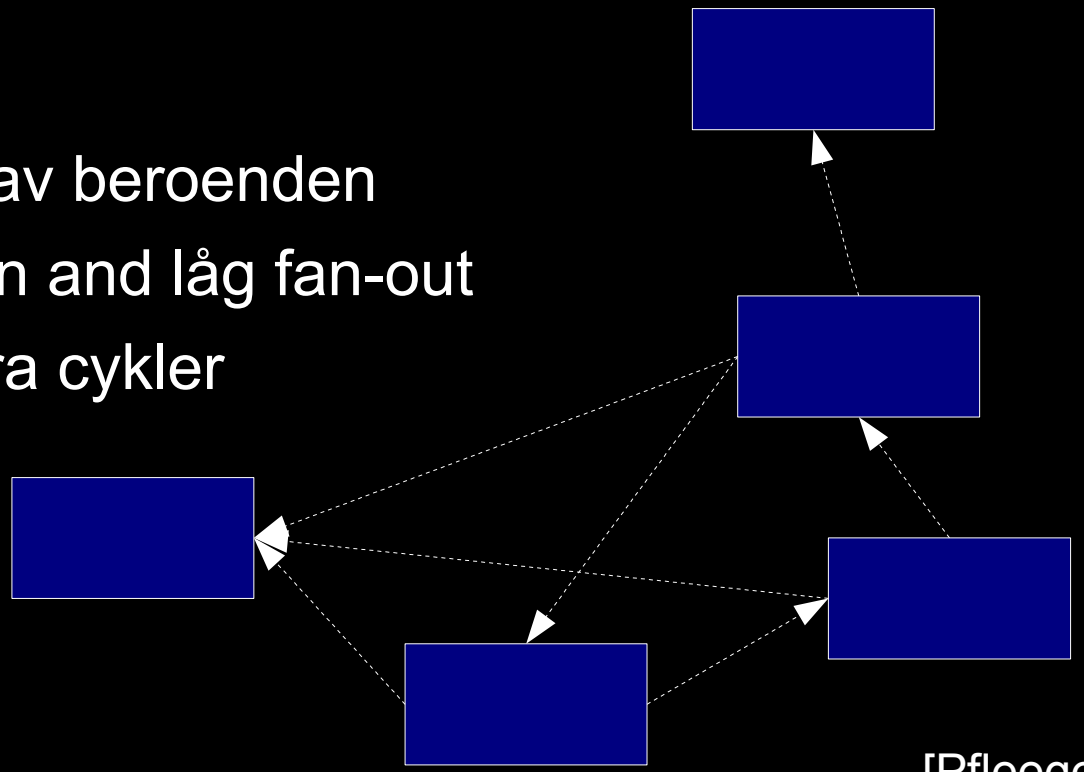
[Pfleeger]

The SOLID Principles

- Mnemonic SOLID (common i Agile)
  - S SRP Single Responsibility Principle
  - O OCP Open Closed Principle
  - L LSP Liskov Subsitutability Principle
  - I ISP Interface Segregation Principle
  - D DIP Dependency Inversion Principle

Inkrementell utveckling

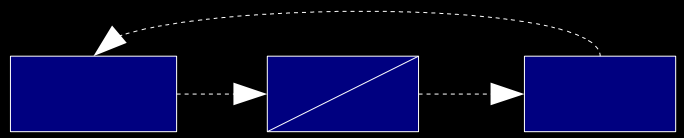
- Förhållanden och beroenden för planering
  - för utveckling i faser
  - för iterativ utveckling
  - för kontinuerlig testning och integration
- ”Uses graphs”
  - karta över alla typer av beroenden
  - sträva efter hög fan-in and låg fan-out
  - identifiera och hantera cykler

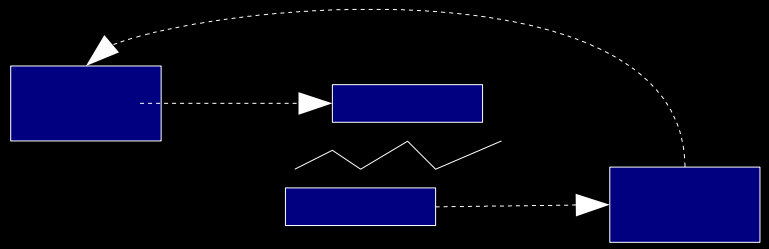


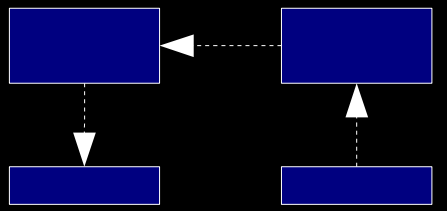
[Pfleeger]

Cirkulärt beroende

- Sandwiching
  - dela upp en modul i två oberoende moduler
  - används för att bryta cirkulärt beroende
  - används för att korta av långa beroende-kedjor



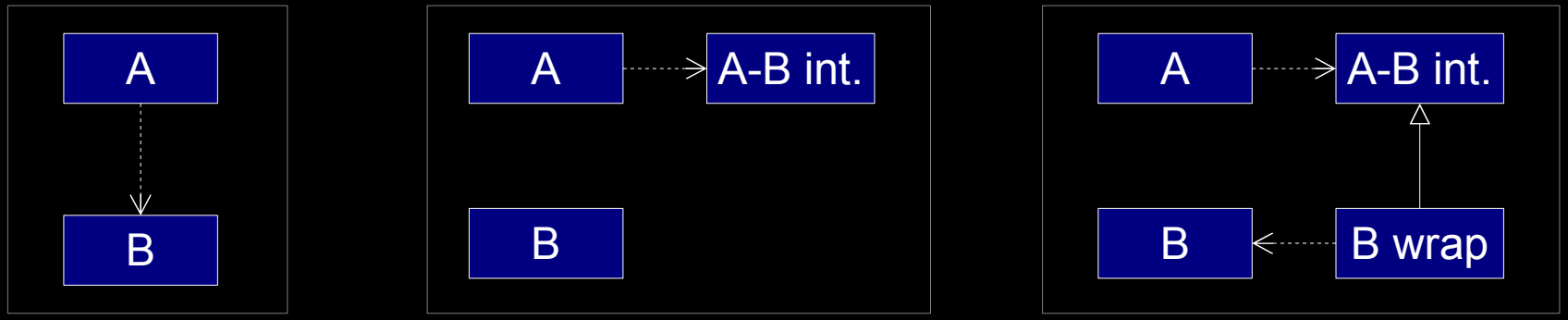




[Pfleeger]

Beroende i fel riktning

- Dependency Inversion
  - A använder B – skapa ett gränssnitt för A istället
  - Skapa en Wrapper för B som implementerar gränssnittet

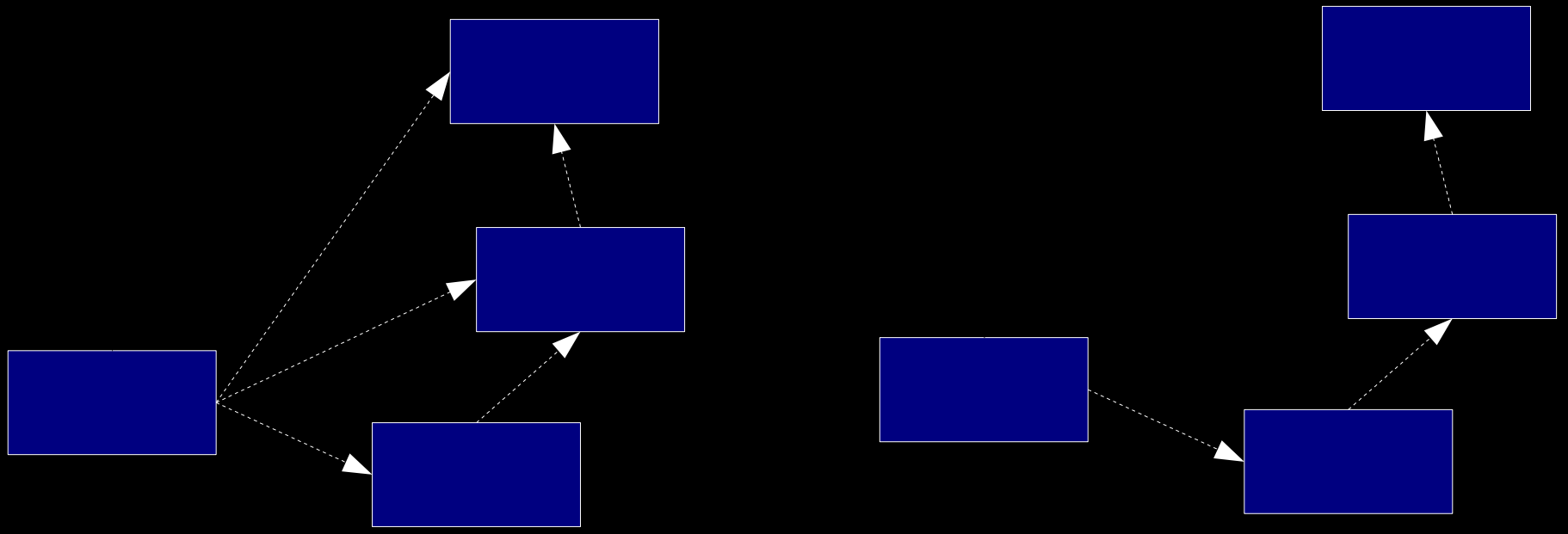


- Dependency Inversion Principle (DIP)
  - hög-nivå-moduler ska inte bero på låg-nivå-moduler
    båda ska bero på abstraktioner
  - abstraktioner ska inte bero på implementationsdetaljer
    implementationsdetaljer ska bero på abstraktioner

[Pfleeger]

Law of Demeter

- “Prata inte med främlingar”-princip
  - reducerar beroenden för färre fel och enklare ändringar



[Pfleeger]

Annat som påverkar design-val

- Datahantering
  - identifiera och kapsla in datahantering
- Hantering av exception
  - räkna med att det kommer att bli fel i anrop
  - hantera alla möjliga exception som kastas vid körning
- Multi-trådning
  - trådsäker datahantering och trådsäkra gränssnitt
  - identifiera och kapsla in behov av trådning
- Ramverk
  - externa ramverk begränsar design-valen
  - men har oftast bra design för den typen av problem

[Pfleeger]

Hur väljer jag design?

- Trade-off-analys
  - Endast som jämförelse mellan föreslagna designval
  - Informell analys – fundera och diskutera
  - Jämförelse-tabeller
    - Prioritera kvalitetsattribut och bestäm viktning
    - Bedöm och gradera kvalitetsattributen för varje modell
    - Beräkna den viktade summan

[Pfleeger]

Programdesign i Agil utveckling

- Bör ske vid varje(!) påbörjad post
- Viktigt att tänka på
  - program-/moduldesign sker alltid, oavsett metodik
  - gemensamma rutiner underlättar samarbetet
  - lös problemen *innan* ni börjar programmera
  - gör så lite som möjligt, *men inte mindre än så*!
    (just barely good enough)
- Tekniker
  - Agile Modelling – använd diagram för att resonera
  - Testa olika design med små bitar kod
  - Whiteboard – diskutera och resonera