F1
Introduktion

EDA260 (D2, C3): Programvaruutveckling i grupp – Projekt

Boris Magnusson, Görel Hedin
Datavetenskap, LTH

Programvaruutveckling i grupp

Produkt skall utvecklas och levereras till kund

- Hög kvalitet på programvaran
- Programvaran skall kunna vidareutvecklas efter kundens önskemål

Metod

- Extreme Programming (XP)

Extreme Programming (XP)

Ny metod (K. Beck, 2001)

Populär bland många företag

“Agile” (“Lättrörlig”, högiterativ utveckling)

Konkreta deltekniker för hela utvecklingsprocessen

Passar mindre projekt (cirka 10 personer)

Kursens mål

Praktisk erfarenhet av programutveckling i grupp

Inblick i hela utvecklingsprocessen

- krav och validering
- planering och tidsuppskattning
- testning
- design och utveckling
- leverans och versionshantering

Exempel på *en* konkret metod (XP)

Fördjupning inom tekniker för OO programutveckling

- refactoring, simple design, ...

Agenda

Administration

- Web-sida, litteratur, schema, ...

Översikt

- Programutvecklingsprocessen
- Extrem Programmering

Websida

http://cs.lth.se/EDA260/

OH-bilder läggs upp efter hand

Obs! Strikta förkunskapskrav

Godkänd tentamen på

- Programmeringsteknik fördjupningskurs *eller*
- OMD

och

Godkända obligatoriska moment på OMD lp 1.

Kurslitteratur

Kursbok

- chromatic: Extreme Programming Pocket Guide
  O'Reilly, 2003, (Handbok i XP)
- 85kr på KF Sigma (kommer snart in, begränsat antal)
- 4,99$ som eBook, www.oreilly.com (se hemsidan)

Kursmaterial

- Labbhandledningar (på kurswebben)
- Häfte med extra artiklar (hämta hos kurssekreterare Lena Ohlsson)

OH-bilder

- Läggs på kurswebben efter hand

Kursens form

Teoridel (lp 2)

- 7 föreläsningar
- 4 laborationer
- kontrollskrivning

Projektdel (lp 3)

- projektstartmöte
- 6 iterationer programutveckling
- redovisning

Schemat lp 2

Läsvecka		Må 8-10	On 10-12	On 15-17, To 8-10 eller To 15-17	Fr 13-15
4	16/11 - 20/11	F1: Introduktion	F2: Översikt över XP, **E:B**	L1: Extreme Hour	
5	23/11 - 27/11	F3: Konfigura-tionshantering		L2: CVS	
6	30/11 - 4/12	F4: Testning och Parpro-grammering		L3: Test First	F5: Refaktorering, Enkel design och Metafor
7	7/12- 11/12	F6: Planering, Arkitektur		L4: Refaktorisering	Kontrollskrivning (13:15-14:00), MA10 F7 Projektintro (14:15-15:00) Kårhusets hörsal

Föreläsningar i kårhusets hörsal (onsdag E:B)

Laborationerna lp 2

Anmälan till lab-tider

- Anmälning på webben

Laborationerna

- Kort labförhör i början av varje lab.
  Godkänt är krav för att få fullfölja labben.
- Obligatorisk närvaro
  (många moment är gruppövningar som ej kan göras individuellt)
- Fullgjorda laborationer krav för att få göra projektet.
- Vid sjukdom,maila omedelbart kursadministratör Görel Hedin:
  Gorel.Hedin@cs.lth.se
  (eller ring sekr. Lena Ohlsson 046-222 80 40 och be henne maila)

Kontrollskrivning

Fredag 11 december kl 13:15-14:00, MA10 (A-E)

Godkänd kontrollskrivning krav för att få göra projektet.

Omkontrollskrivning:
Tisdag 12 januari, kl 13:15-14:00, E:3336

Schema lp 3

Vecka		Må 8-17 Långlabbar	On 13-15 eller To 10-12 Planeringsmöte	Fre 10-12 Föreläsning
1	18/1 - 22/1	Projektstart 8-10 eller 10-12	P1	
2	25/1 - 29/1	LL1	P2	
3	1/2 - 5/2	LL2	P3	
4	8/2 - 12/2	LL3	P4	
5	15/2 - 19/2	LL4	P5	
6	22/2 - 26/2	LL5	P6	
7	1/3 - 5/3	LL6	Redovisning	Avslutning

Projektet

Varje team:
8-10 utvecklare (EDA260) och 1-2 coacher (EDA270)

Projektstartmöte

- 2 timmars projektstartmöte (oblig. närvaro)

XP metodik i 6 iterationer

- 2 timmars planeringsmöte (oblig. närvaro)
- 6 timmars individuellt arbete (experiment, litteraturstudier)
- 8 timmars långlaboration med programutveckling (oblig. närvaro)

Avslutande projektredovisning

- 2 timmars redovisning (oblig. närvaro)
- 2 timmars avslutning (oblig. närvaro)

Personal

Boris Magnusson– föreläsare, kursansvarig, superkund

Lars Bendix – gästföreläsare (CM)

Görel Hedin – kursadmininstratör, supercoach

Labbhandledare:

- Lars Bendix
- Lennart Andersson
- . . .

Kunder

- ... ännu ej fastställt ...

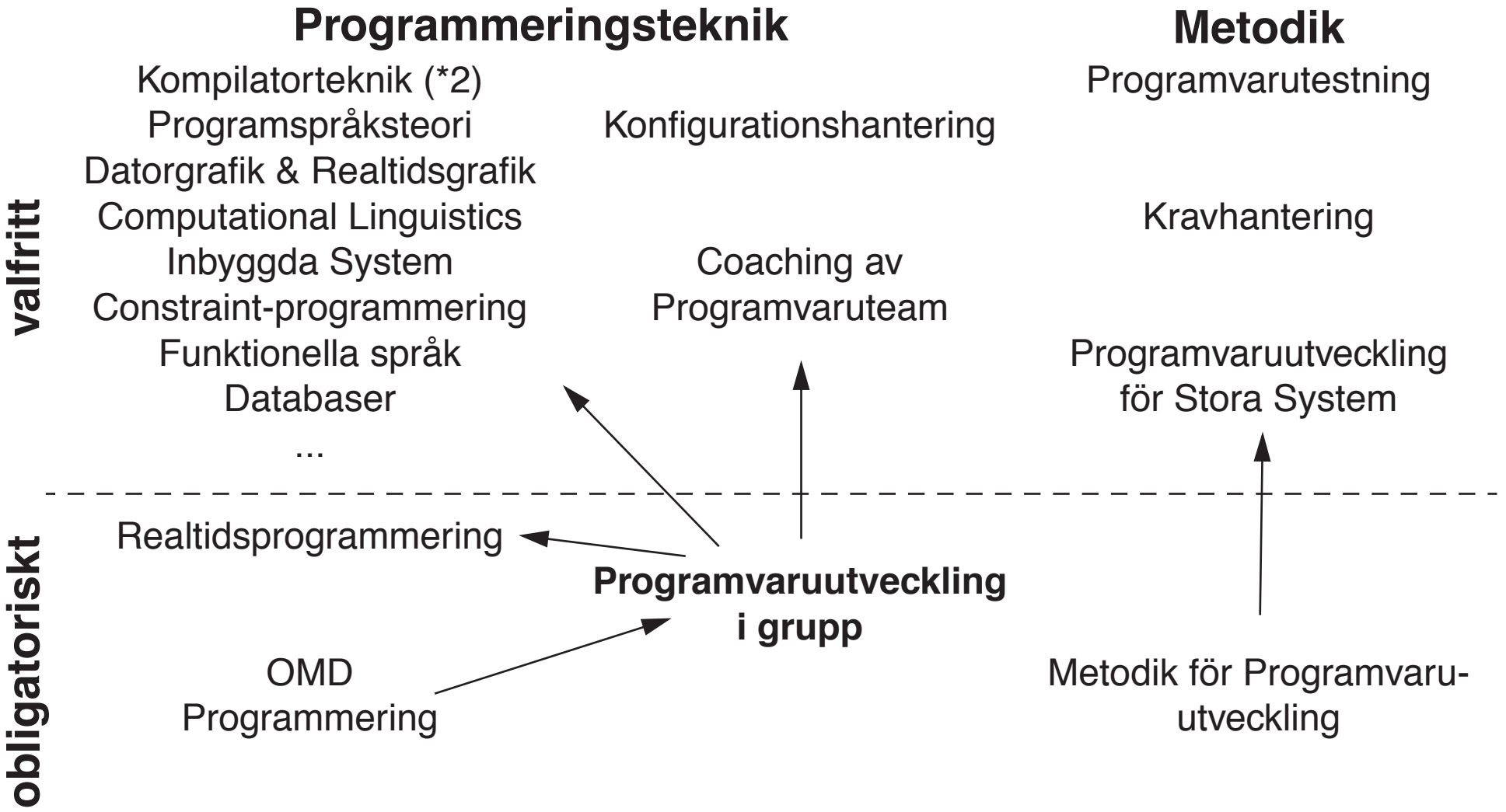
Examination

Godkänt / Icke godkänt

För godkänt krävs

- fullgjorda laborationer i ht2
- godkänt på kontrollskrivningen i ht2
- aktivt deltagande i möten och långlaborationer under vt1
- godkänd projektredovisning och avslutning under vt1

Relaterade kurser, EDA260



Programmeringskurserna hittills

Väsentligen ensam programmerare

Små uppgifter

Givna förutsättningar (av oss)

Ingen som vill ha lösningarna, egentligen

Verkligheten

Större uppgifter

Många utvecklare

Kunder som ställer (otydliga) krav

Systemet förändras,

- levereras,
- modifieras, många gånger

I denna kurs skall vi närma oss denna situation

Roller i programutveckling

(förenklat)

Kund
- betalar utvecklingen
- formulerar kraven

Användare
- använder programmet
- ger feedback

Utvecklare
- realiserar programmet

Exempel på typisk rollfördelning

Vem är kund, användare, utvecklare?

- Biljettbokningssystem
- Ordbehandlingssystem
- Adressboken i en mobiltelefon
- Spelprogram
- Öppen källkod

Exempel på kunder

extern beställare

- t.ex. annat företag

marknadsavdelningen på vårt företag

- om det är en generisk produkt vi tar fram

intern beställare (annan del av vårt företag)

- t.ex. om vår produkt är en del i en större

utvecklarna själva

- i open source projekt (utvecklarna bidrar frivilligt)

Olika typer av krav

funktionella krav

- beskriver vad systemet skall göra
- ett funktionskrav svarar ofta mot en viss del av koden
- kan ofta implementeras som en enhet

icke-funktionella krav

- beskriver egenskaper och begränsningar hos systemet t.ex. krav på svarstider, minnesstorlek, etc.
- påverkar ofta den övergripande designen (arkitekturen)
- behöver ofta beaktas vid implementationen av de funktionella kraven

Exempel på krav

(adressbok i mobiltelefon)

Funktionella krav

- Man skall kunna lägga till och ta bort namn & telnr
- Man skall kunna söka upp ett namn och ringa
- Ett nummer man precis har ringt skall kunna läggas in i adressboken utan att man behöver knappa in numret igen
- ...

Icke-funktionella krav

- Programkoden måste vara mycket kompakt (helst mindre än 50 KB)
- Svarstiderna vid interaktion får inte vara märkbara
- Alla kommandon skall vara enhetligt utformade och lätta att lära sig
- ...

När formuleras kraven?

Bra att försöka hitta så många viktiga krav som möjligt från början. Men...

... oftast behöver kraven uppdateras under projektets gång

Varför uppdateras kraven?

I praktiken vet oftast “kunden” inte exakt vad han/hon vill ha från början, ändrar sig efter hand.

Efter release av systemet får kunden mer insikt i problemen och möjligheterna med systemet.

Kunden är inte alltid densamma som användaren – som har andra behov.

Omvärlden förändras. T.ex., konkurrenten kommer ut med en ny tjänst som vi måste få in i vår produkt också.

Hur formuleras kraven

Notation?

- Formell matematisk notation? (Sällan tillämpligt)
- Naturligt språk?
- Scenarier för typiska användningsfall?
- ...

Hur detaljerat?

- Fullständig precis specifikation? (Sällan möjligt)
- ...
- Korta rubriker, detaljerna i muntlig dialog? (XP “stories”)

Utvärdering av systemet?

“Byggde vi systemet rätt?” (Verifiering)

- Dvs fungerar systemet enligt (vår tolkning av) kravspecen?
- Är systemet (tillräckligt) felfritt?
- Är systemet (tillräckligt) väldesignat? (så att vi kan modifiera det)
- Vi (utvecklare) utvärderar systemet
  – kodgranskning
  – testning

“Byggde vi rätt system?” (Validering)

- Dvs byggde vi det som kunden förväntade sig?
- Är användarna nöjda med systemet?
- Kund och användare utvärderar systemet (testkörningar)

Kodgranskning

(inspection, code review)

Någon läser kod som en annan person har skrivit

- dålig design kan upptäckas
- buggar kan upptäckas
- effektivt sätt att öka kodkvaliteten
- sprider kunskap om systemet till flera personer

Granskningar i XP: parprogrammering

Testning

kör (del av) programmet på test-indata,
kontrollera att det ger förväntad utdata

- tester på olika nivåer: metod, klass, delsystem, hela systemet, ...
- *regressionstestning* – kör igenom gamla testfall för att kontrollera att ändringar inte har förstört sådant som fungerat tidigare
- testkörningar kan *automatiseras* –
  ett program kör igenom alla testfall och kontrollerar att de ger rätt resultat
  (en förutsättning för effektiv regressionstestning)

Obs! Med testning kan vi hitta fel,
men inte bevisa att programmet är felfritt.

Testmetodik

Vem skriver testfallen?

- Utvecklare?
- Särskilda testare?

När skriver man testfallen?

- Innan/samtidigt som man kodar?
- Efter man kodar?

När kör man testfallen?

- En gång, i samband med att man skriver relaterad kod?
- Inför varje release?
- Efter varje ändring?

Vad testar man?

- Allt man kan komma på?
- Vanliga fall? Ovanliga fall?

Hur skall systemet organiseras?

Mjukvaruarkitektur och design

- Skall systemet delas upp i flera kommunicerande program?
- Behöver systemet köras distribuerat?
- Kan man använda några färdiga delar?
- Skall vi utveckla delar som kan återanvändas i andra system?

När bestämmer man arkitekturen?

- Innan man kör igång utvecklingen?
- Initial enkel arkitektur som växer under utvecklingen? (XP)

Många utvecklare

Inget system idag utvecklas av en person

- Hur delar man upp arbetet?
- Hur fördelar man ansvaret för olika uppgifter?
- Hur synkroniserar man olika utvecklares insatser?
- Finns alla utvecklare på en ort, eller är de geografiskt distribuerade?

Beror på: är man 10, 100, 1000 personer?

Hur delar man upp ansvaret för koden?

Olika strategier

Personer/grupper ansvarar för vissa delsystem

- Innebär ofta att den/de “äger” koden
- Andra får övertyga den/dem om vad som skall göras
- Vad gör man när någon slutar?

Personer/grupper ansvarar för deluppgifter (rätta fel, ny funktionalitet)

- Genemsamt ägd kod - alla kan ändra
- Hur förhindra att de inte förstör för varandra?
- Vissa delar kanske kräver specialkompetens?

Parallell utveckling

Utvecklare arbetar parallellt på olika funktioner eller delar i systemet. Olika strategier:

Utvecklingsfas – integrationsfas?

- Divergerande kopior av systemet som “mergas”
- Utvecklarna stör inte varandra under utvecklingsfasen
- Integrationen kan ta lång tid och innebära att man måste vänta på varann

Successiv integration?

- Varje fungerande del görs tillgänglig för de andra så fort som möjlgt
- Varje ny deluppgift startar från den senaste versionen
- Nyutveckling kommer tidigare i bruk. Integrationsproblem upptäcks tidigt.

Konfigurations- och versionshantering! Viktigt oavsett strategi!

Hur planerar man arbetet?

Deluppgifter

- hur delar man upp arbetet i smådelar?
- hur tidsuppskattar man de olika delarna?

Planering

- hur prioriterar man mellan olika delar?
- när finns rätt personal tillgänglig?
- kan man minska “ledtiden” (kalendertiden) till release?
- hur kan man följa upp och planera om successivt?

Vad är viktigast?

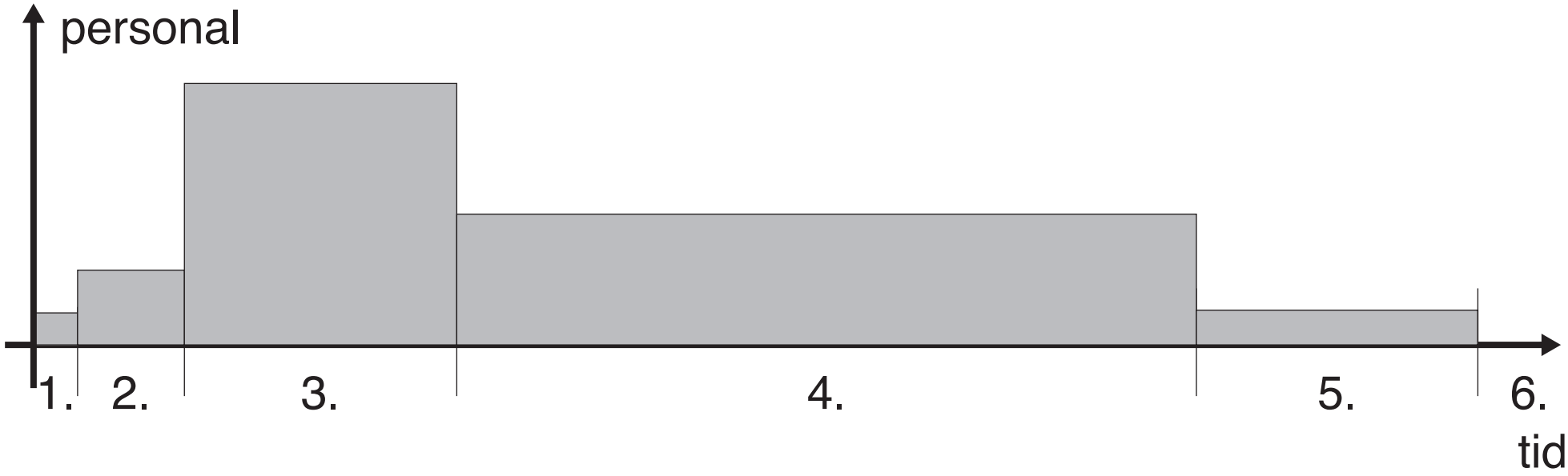
- Fullständigt system (men kanske försenat)?
- Deadline för release (men kanske med begränsad funktionalitet)?

När får kunden systemet?

Hur ofta gör man release (leverans av ny version)?

- En gång med all funktionalitet?
- Regelbundet med successivt ökande funktionalitet?
- Kontinuerligt?
  (Kunden kan hela tiden ladda ner den senaste versionen)

Ett programs “livscykel” (exempel)



1. Initial idéfas
2. Utveckling av första minimala körbara systemet
3. Utveckling av första skarpa releasen
4. Vidareutveckling, ny funktionalitet
5. Utfasning, inga nya användare, enbart felrättning
6. Slut, produkten underhålls inte längre

Dokumentation

Vad behöver man dokumentera?

- Kravspecifikation?
- Arkitekturbeskrivningar?
- Designbeskrivningar?
- API:er och implementationer?

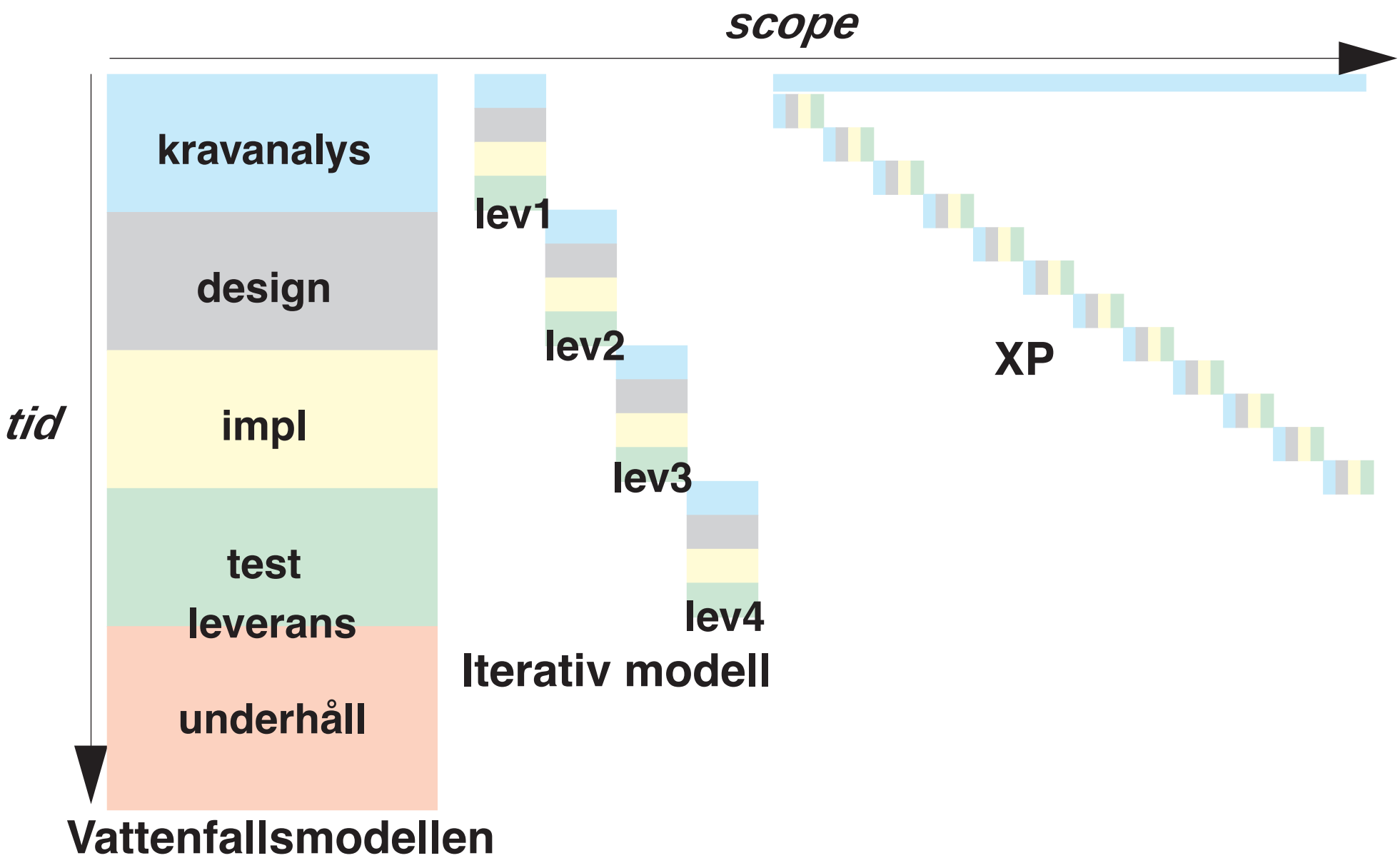
Hur detaljerat?

- Få sidor, informella beskrivningar?
- Tjocka pärmar enligt föreskrivna metoder och notationer?

När dokumenterar man?

- I förväg, för att föreskriva det fortsatta arbetet?
- I efterhand, för att dokumentera hur systemet faktiskt blev?
- Hela tiden, för att hålla all dokumentation i takt med koden?

Utvecklingsmodeller



XP-metoden

Högiterativ “agil” metod

De traditionella “faserna” (kravanalys, design, impl, test) vävs samman

Körbar produkt så tidigt som möjligt. Vidareutveckling är normalfallet.

Fokus på test och test-driven utveckling

Muntlig kommunikation hellre än skriftlig

Små inkrement – feedback i varje steg

Konkreta deltekniker

Projektet i kursen

I kursen fokuserar vi på arbete i grupp om 10

Det behövs en hel del metod för att arbeta effektivt redan i denna storlek

Men inte “tunga” metoder med mycket dokumentation

Senare kurser fokuserar på problemen i större organisationer och tyngre metoder

Sammanfattning

Utveckling av programvara är komplext – många olika synsätt och metoder

Några aspekter återkommer i alla projekt:

- kravhantering, design, test, implementation
- release, användarfeedback, validering
- versionshantering, parallell utveckling

I kursen lär vi oss *en* metod på djupet: XP

- metoden är ganska ny, men enormt inflytelserik
- konkreta användbara deltekniker
- ger helhetsbild för programutvecklingsprojekt och grund för att förstå andra synsätt och metoder

Läsanvisningar

- Häftet: Artikel av Kent Beck
- Kursboken: Part I (Why XP)