[ simula . research laboratory ]

INF3120 H2005

# Agile Software Development

# eXtreme Programming (XP)

# Pair Programming

Hans Gallis
hansga@simula.no

03.10.2005

[ simula . research laboratory ]

## Pensum

- Kap. 17 i Sommerville (temaet inngår også i kategorien "systemutviklingsprosesser")
- Kursorisk:
  - Kent Beck; *Embracing Change with Extreme Programming*; IEEE Computer, October 1999
  - Barry Boehm; *Get Ready for Agile Methods, with Care*; IEEE Computer, January 2002
- For spesielt interesserte, se bok om agile metoder (pdf): http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf

2

[ simula . research laboratory ]

## Mål

- Bevissthet tilknyttet “Agile Software Development”
- Deltaljert kjennskap til én “Agile Method”: eXtreme Programming
- Detaljert kjennskap til én teknikk innen eXtreme Programming: Parprogrammering





[ simula . research laboratory ]

## Historisk perspektiv

- Software Crisis (1960’s)
  - Software intensive systems delivered late, over budget and do not meet the quality requirements
- Solution attempt #1: **Structured Methods** (in 1980’s)
- Solution attempt #2: **Object-oriented methodologies**
- Chronic Software Crisis (1990’s)
  - Software intensive systems still delivered late, over budget and do not meet the quality requirements
- Solution attempt #3: **Software process improvement**
- Solution attempt #4: **Agile development methodologies**
- Solution attempt #n: ?

Pekka Abrahamsson 2005

## Ofte tilfelle i den “virkelige” verden……

- Krav endrer seg hele tiden
- Rask time-to-market (Internet)
- Lite langsiktige planer
- 60% av funksjonaliteten blir sjeldent eller aldri benyttet
  - Jim Johnson, Standish Group







## Dette medfører:

- Må ha hyppige leveranser
- Må kontinuerlig endre koden (og designet)
  - Forutsetter at systemet er i en “sunn” tilstand (hele tiden!)
- Må kommunisere kontinuerlig (utvikle “riktig” produkt)

[ simula . research laboratory ]

## Paradigmeskifte?

Før:

- Internett lite “modent”
- (Rask) time-to-market
- **Vannfall vs iterativ/inkrementell utv.**

Nå:

- Internett ”modent”
- Raskere time-to-market
- **Agile vs plan-driven utv.**





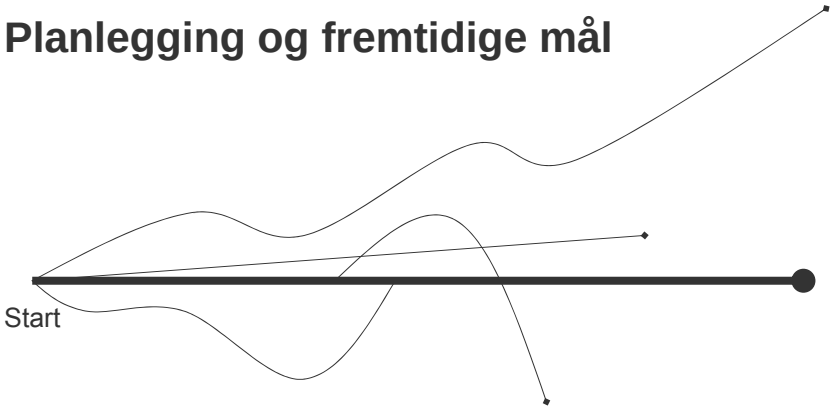
[ simula . research laboratory ]

## Agile vs plan-driven methodologies

- Agile (= smidig på norsk?):
  - Planlegger ikke for fremtiden
  - Ikke dokumentdrevet
- Plan-driven:
  - Planlegger for fremtiden
  - Dokumentdrevet

[ simula . research laboratory ]

# Planlegging og fremtidige mål



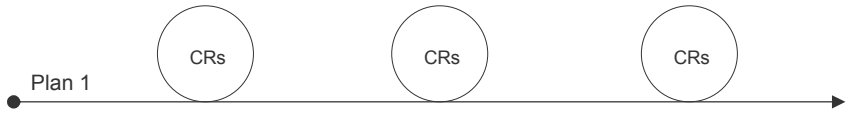

→ = Mål

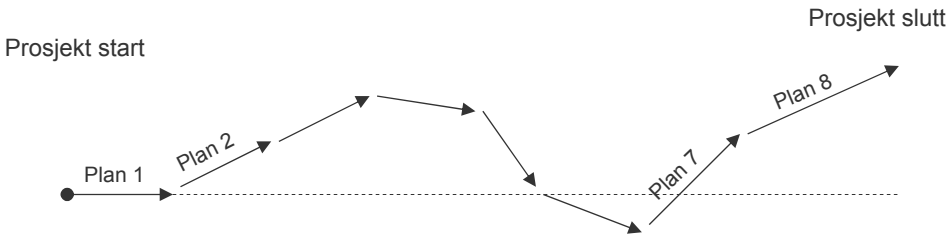




[ simula . research laboratory ]

# Agile vs plan-driven



[ simula . research laboratory ]

## Evolusjon og læring

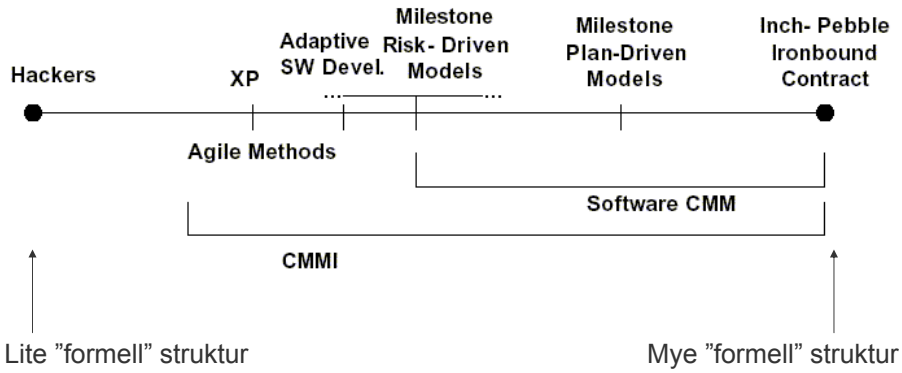
- Usikkerhet krever læring (hyppige tilbakemeldinger)
  - Fra kollegaer
  - Fra iterasjoner i prosjektet
  - Fra testing av kode/system
  - Fra andre prosjekter
  - Fra kunder og sluttbrukere
  - Fra andre organisasjoner/systemer
  - Fra forskning!





[ simula . research laboratory ]

## Agile vs plan-driven methodologies



[ simula . research laboratory ]

| Agile Home Ground | Plan-Driven Home Ground |
|---|---|
| • Agile, knowledgeable, collocated, collaborative developers | • Plan-oriented developers, mix of skills |
| • Above plus representative, empowered customers | • Mix of customer capability levels |
| • Reliance on tacit interpersonal knowledge | • Reliance on explicit documented knowledge |
| • Largely emergent requirements, rapid change | • Requirements knowable early; largely stable |
| • Architected for current requirements | • Architected for current and foreseeable requirements |
| • Refactoring inexpensive | • Refactoring expensive |
| • Smaller teams, products | • Larger teams, products |
| • Premium on rapid value | • Premium on high-assurance |





[ simula . research laboratory ]

# The Agile Manifesto

- **Individer og interaksjon** fremfor prosesser og verktøy
- **Fungerende software** fremfor utstrakt (omfattende) dokumentasjon
- **Kundemedvirkning** fremfor kontraktsforhandlinger
- **Reagere på endringer** (fleksibilitet) fremfor å følge en plan
- Se www.agilemanifesto.org og www.agilealliance.org

NB! Uthevede verdier er vektlagt, men verdiene til høyre i setningene er også viktig!

[ simula . research laboratory ]

## Eksisterende agile metoder

- Methods for agile software development:
  - Agile software process model [Ayoama, 1998]
  - Adaptive Software Development [Highsmith, 2000]
  - Crystal Family of Methodologies [Cockburn, 2000]
  - Dynamic Systems Development Method [Stapleton, 1997]
  - Extreme Programming [Beck, 1999]
  - Feature-Driven Development [Palmer & Felsing, 2002]
  - Lean software development [Poppendieck x 2, 2003]
  - Scrum [Schwaber, 1995; 2002]
  - … the list is growing every year…
- Combination of approaches:
  - Agile Modeling [Ambler, 2002]
  - Internet-Speed Development [Cusumano & Yoffie, 1999; Baskerville et al., 2001; Truex et al., 1999]
  - Pragmatic Programming [Hunt & Thomas, 2000]





[ simula . research laboratory ]

## eXtreme Programming (XP)

- Utviklet av Kent Beck og Ward Cunningham (1996)
- En kodenær disiplin for programvare-utvikling
  - 4 verdier
  - 12 prinsipper/praksiser
  - 4 kjerneaktiviteter
- Kode er det eneste man er helt avhengig av å produsere!
- Et sett av ”best practises” som sees i sammenheng og støtter hverandre (ikke noen nye praksiser!)

[ simula . research laboratory ]

## De fire verdiene

- Kommunikasjon
  - Problemer i et prosjekt kan ofte spores tilbake til en eller flere som ikke snakket med en eller flere andre!
- Enkelhet
  - Det er bedre å gjøre en enkel ting i dag, og betale litt mer i morgen, enn å gjøre en komplisert ting i dag som kanskje må endres i morgen (eller som kanskje aldri blir brukt)
- Tilbakemelding
  - Tilbakemeldinger på alle nivåer (hele tiden) hjelper oss å holde prosjektet på rett vei
- Mot
  - Sammen med de tre første verdiene hjelper 'mot' oss til å gjøre høyrisiko eksperimentering (gjøre ting på en annen måte), noe som også kan gi høy belønning/avkastning hvis det lykkes.





[ simula . research laboratory ]

## De 12 prinsippene (teknikker)

1. Planning game
2. Små releaser
3. Metaforer
4. Enkelt design
5. Testing
6. Refactoring
7. Parprogrammering
8. Kollektivt eierskap
9. Kontinuerlig integrasjon
10. 40-timers uke
11. On-site kunde
12. Kodestandarder

[ simula . research laboratory ]

## XP – mer en filosofi enn metode!

- XP sier ikke:
  - Hvordan praksisene skal gjennomføres
  - Hvem som skal utføre dem (roller)
  - Hvilke leveranser til hvilken tid
    - Hva skal f.eks. leveransene inneholde?
      - Dokumentasjon, designmodeller, kode, etc. etc.
- XP er:
  - Et sett av 12 enkeltstående praksiser
    - Skal gjennomføres **ekstremt**
  - Funksjonell prototyping?





[ simula . research laboratory ]

## XP's "extreme" filosofi

- Hvis testing er bra – ja, da gjør vi det hele tiden
- Hvis kodelesing/inspeksjoner er bra – ja, da gjør vi det hele tiden
- Hvis iterasjoner er bra – ja, da gjør vi det så ofte som mulig (dager og uker istedenfor måneder og år)
- Hvis endringer skjer uansett – ja, da tilrettelegger vi for det istedenfor å motvirke at det skjer (rigide planer)
- Hvis kommunikasjon/samarbeid er viktig – ja, da gjør vi det hele tiden

## Planning game

- Klarlegge
  - hva er ønskelig (forretningshensyn)
  - hva er mulig (tekniske hensyn)
- Estimere
- Prioritere
- Planlegge







## Små releaser

- Minst mulig kode
- Mest mulig forretningsverdi
- 1-2 måneder (mellom hver gang systemet settes i produksjon)
- Risikoreduserende (får hyppig feedback)

## Metaforer

- Få et vokabular (et sett av felles begreper) for tekniske ting uten å bruke tekniske termer
- La prosjektet være styrt av én metafor
  - Operativsystem → skrivebord
  - Forhandlersystem → flytog-billettsystem
- Gjør det lettere å kommunisere og utarbeide arkitekturen







## Enkelt design

- Møt dagens krav:
  - YAGNI = You aren't gonna need it!
  - Ikke bruk tid på "nice-to-have-features"
  - For mye up-front design vil medføre MYE unødvendig arbeid!
- Det riktige designet
  - kjører alle testene
  - har ikke duplikat logikk
  - kommuniserer godt
  - har færrest mulig metoder

[ simula . research laboratory ]

## Testing

- Funksjonalitet uten automatiske tester finnes ikke
- Programmerere
  - skriver enhetstester (test-first)
  - gir programmereren tillit til programmet
- Kunder
  - skriver funksjonelle tester
  - gir kunden tillit til programmet





[ simula . research laboratory ]

## Kontinuerlig integrasjon

- Kode integreres og testes hver dag
  - Skal hele tiden ha et stabilt system
  - Skal finne feil tidligst mulig
- Hvis integrasjon feiler har det å rette opp dette høyeste prioritet

## Refactoring

- = forbedre kodestrukturen uten å påvirke dens eksterne oppførsel
- Under implementasjon
  - kan koden endres for å gjøre det enklere å implementere denne funksjonaliteten?
- Etter implementasjon
  - kan koden skrives om til et enklere design, uten at testene ødelegges?







## Parprogrammering

- Kommer til slutt i forelesningen!

## Kollektivt eierskap

- Alle er ansvarlig for hele systemet
- Alle har rett til å endre all kode
- Likevel, ikke alle kjenner alle deler like godt







## On-site kunde

- Kunden
  - Sitter sammen med utviklerne
  - Svarer på spørsmål
  - Gjør prioriteringer (på lavt nivå)
  - Deltar i diskusjon av løsninger
- Ikke nødvendigvis 100% stilling
  - Hvis kunden ikke er villig til å investere skikkelig med tid i prosjektet er det kanskje et tegn på at prosjektet ikke er viktig nok?

## 40-timers uke

- XP er krevende
  - Intensivt med korte leveranser
  - Ekstremt mye kommunikasjon
  - Mye tenking
- Viktig å være uthvilt
- 40-timers uke skal være normalen (ingen overtid!)
  - Får du ikke gjort jobben på 40 timer har du for mye å gjøre (justeres i neste release/iterasjon)
  - Mye overtid = symptom på usunt prosjekt (noe er galt!)





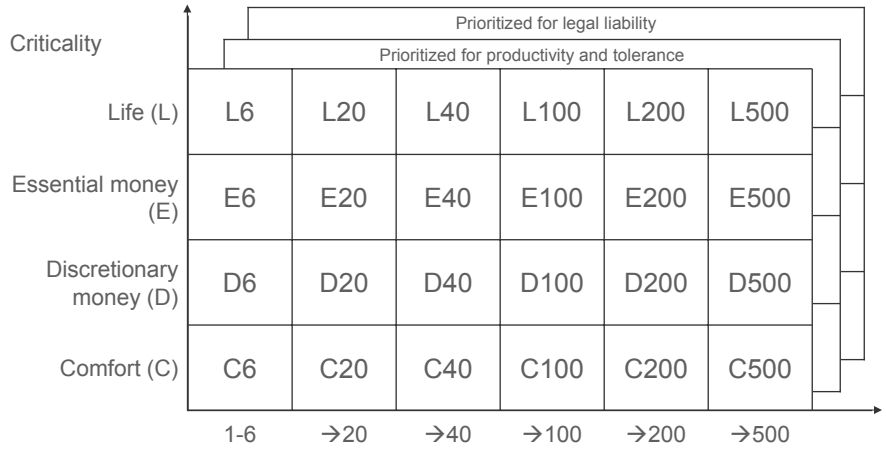


## Kodestandarder

- Nødvendig for
  - Parprogrammering
  - Kollektivt eierskap
  - Disiplin
- Skal være enkel men dekkende

[ simula . research laboratory ]

# Når kan man benytte XP?



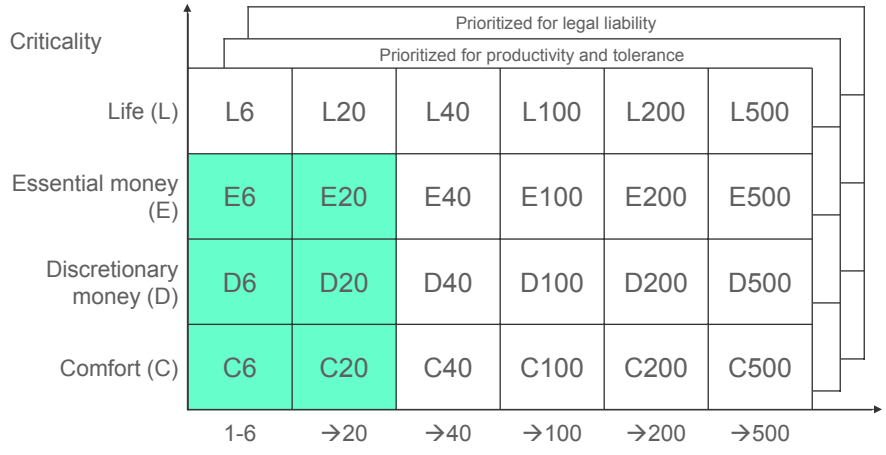

Alistair Cockburn, 2002





[ simula . research laboratory ]

# Når kan man benytte XP?




Alistair Cockburn, 2002

[ simula . research laboratory ]

## XP og Design

- Det er ingen som sier at design er nedprioritert i XP!
  - Gjør så mye design du trenger for å få oversikt og for å forstå
- UML-modeller forekommer på tavle, ark etc.





[ simula . research laboratory ]

## Oppsummering

- Agile ≈ pragmatisk systemutvikling
  - Velg praksiser/løsninger som passer dine behov (prosjekt)
  - Verktøykasse bestående av teknikker og praksiser som settes sammen til en helhetlig prosess
- XP:
  - Mer en filosofi enn en komplett utviklingsmetodikk
  - Kodenær
  - Menneskeorientert
  - Lettvekt (prøver å fjerne overhead)
  - Praksisene er ikke nye, men ekstreme!
  - Praktisk Knowledge Management

[ simula . research laboratory ]

## Parprogrammering (PP)





[ simula . research laboratory ]

## PP – Samarbeid og kommunikasjon

- Opptil 70% av totaltiden i et IT-prosjekt går med til kommunikasjon – designmøter, oppklaring av misforståelser etc. (Rapid Software Development through Team Collocation – Teasley *et al. 2002*)
- En utvikler bruker generelt (Peopleware – DeMarco and Lister 1999):
  - 30% av sin tid til individuelt arbeid,
  - 50% av sin tid til å arbeide med en annen person, og
  - 20% av sin tid til å arbeide med to eller flere personer
- Det er vanlig å bruke 30-70% av totaltiden til å lokalisere og rette feil (Gibson – Managing Computer Projects)

[ simula . research laboratory ]

## PP – Definisjon

- **To utviklere** (partnere) arbeider på **samme oppgave** (v.h.a. **én skjerm og ett tastatur)**
- Involverer ikke bare **koding**, men også andre faser av systemutviklingsprosessen som f.eks. **design** og **testing**
- "Stand-alone" praksis (uavhengig av type prosess – ikke bare passende for XP)
- Burde heller vært kalt: **Parutvikling**!





[ simula . research laboratory ]

## PP – Roller

- **Sjåfør**:
  - Sitter med keybordet eller tegner ned designet
- **Navigatør (kartleser)**:
  - Observerer aktivt arbeidet til sjåføren
  - Ser etter taktiske og strategiske feil
  - Tenker og søker etter alternativer
  - Skriver ned ting man må huske å gjøre
  - Slår opp i referanser (manualer, web-sider osv.)

[ simula . research laboratory ]

## PP – Roller

- Bytting av roller regelmessig (sjåfør vs navigatør)
- Rotere partnere (innad i teamet) → Spre kunnskap/informasjon





[ simula . research laboratory ]

## Fordeler ved bruk av PP (sammenliknet med individuell programmering)

- Redusert "time-to-market"
- Reduserte utviklingskostnader
- Økt kvalitet (kode og system)
- Økt informasjons- og kunnskapsoverføring
- Redusert kostnad ved trening av nye ansatte
- Redusert kostnad ved rekruttering av nye personer til pågående prosjekter
- Forbedret kommunikasjon mellom utviklerne
- Forbedret tillit mellom utviklerne
- Høyere fokus på oppgaven som skal løses
- Kontinuerlig læring
- Økt motivasjon (trivsel)
- Redusert risiko for å miste nøkkelutviklere
- Enkel teknikk å lære og bruke

[ simula . research laboratory ]

## Kostnader ved bruk av PP (sammenliknet med indiviuell programmering)

- PP er bare kontinuerlig kodeinspeksjon med lite eller ingen forbedret kvalitet i produktet som utvikles sammenliknet med individuell programmering og kodeinspeksjon
- PP medfører dobbel kostnad (to personer på samme oppgave uten ekstra kvalitet)
- Vanskelig teknikk å innføre og bruke pga stadige forstyrrelser (møter, telefoner, osv.)
- Vanskelig å finne gode og effektive par
- Komplekse oppgaver løses best alene!





[ simula . research laboratory ]

## PPE (Pair Programming Experiment)

- 295 junior, intermediate and senior profesjonelle Java konsulenter totalt
- 99 løste oppgavene individuelt
- 98 løste oppgaven vha PP
  - Norge: 41
  - Sverige: 28
  - Storbritannia: 29

[ simula . research laboratory ]

## Hypoteser

Bruk av PP resulterer i

- Raskere utviklingstid (kalendertid)
- Lavere totaltid (PP: 2 x utviklingstid)
- Færre logiske feil (korrekthet)

sammenliknet med individuell programmering





[ simula . research laboratory ]

## Initielle resultater

- Effekten av PP (mht utviklingstid, totaltid og korrekthet) avhenger av ekspertisen til utviklerne og oppgavekompleksitet:
  - Fordelene ved PP reduseres med økende programmeringsferdigheter
  - Fordelene ved PP øker med økende oppgavekompleksitet
- PP krever signifikant mer totaltid enn individuell programmering (uavhengig av utviklerkategori)
- Junior-utviklere har størst fordel av PP, spesielt på komplekse oppgaver :
  - PP øker sannsynligheten for å få korrekt løsning (færre logiske feil)
  - Effekten av PP mht korrekthet er størst for komplekse oppgaver
- Junior par har omtrent samme sannsynlighet for å få korrekt løsning som intermediate og senior par (≈ 80 prosent korrekt)
- Seniorer:
  - Ingen klare fordeler av PP

[ simula . research laboratory ]

## Diskusjon: Ekspertise versus oppgave-kompleksitet

- På hvilke oppgaver er man junior, intermediate eller senior utvikler?
- Avhengig av bl.a.
  - Programmeringsferdigheter
  - Domenekunnskap
  - Grad av “wicked problems” (problemer med mange løsningsrom og mange “stakeholders” – ingen fasit!)
  - + mye mer (kognitiv psykologi)





[ simula . research laboratory ]

## Case study: Innføring av PP

- Vanskelig å finne ”uforstyrret” tid
  - Lag kjernetid for parprogrammering, f.eks. fra kl. 10-14
  - Aktivt samarbeid krever regelmessige pauser
- Alle oppgaver egner seg ikke for PP?
  - Bestem graden av parprogrammering (og partnere) gjennom regelmessige og korte stand-up møter
  - Paret deler samme kontorplass og har kontinuerlig dialog
- Kan man droppe QA-rutinene når man benytter PP?
  - Nei, man kan ikke gjøre enten PP eller individuell programmering (riktig spørsmål: når og hvordan skal man benytte PP?)
  - Dokumentering av kode etc. kan bli utelatt ved PP
  - Hvis ”test-first” praktiseres: Lag testene i par → implementer individuelt → gjør QA-rutiner i par til slutt

[ simula . research laboratory ]

# Takk for oppmerksomheten ☺