INF3120 H2004

Agile Software Development

eXtreme Programming (XP)

Hans Gallis hansga@simula.no

Stein Grimstad steingr@simula.no

30.09.2004

[simula . research laboratory]

Pensum

- Kap. 17 i Sommerville (temaet inngår også i kategorien “systemutviklingsprosesser”)
- Kursorisk: artikkel av Kent Beck (se link under “forelesningsplan” på hjemmesiden til kurset)

Mål

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

Ofte tilfelle i den “virkelige” verden……

- Krav endrer seg hele tiden
- Rask time-to-market (Internet)
- Lite langsiktige planer

Dette medfører:

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

Paradigmeskifte?

Før:

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

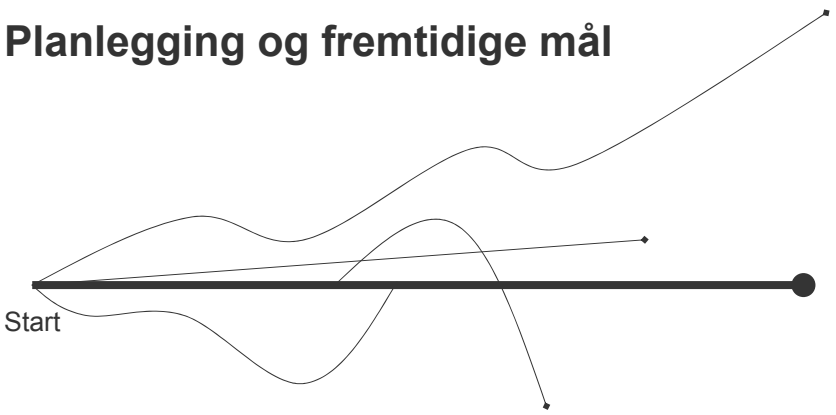
Nå:

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

Agile vs plan-driven methodologies

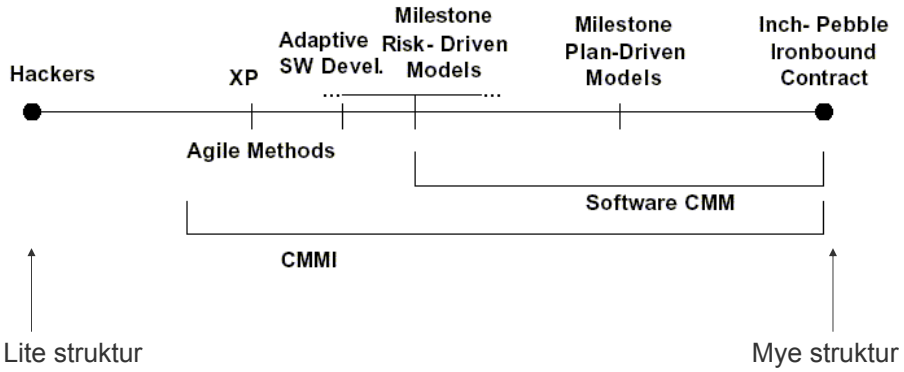
- Agile (= smidig på norsk?):
  - Planlegger ikke for fremtiden
- Plan-driven:
  - Planlegger for fremtiden

Planlegging og fremtidige mål



[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
• Refacoring inexpensive	• Refactoring expensive
• Smaller teams, products	• Larger teams, products
• Premium on rapid value	• Premium on high-assurance

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!

The “Agile movement” – hva finnes av metoder?

- Scrum
- Dynamic Systems Development Method (DSDM)
- Crystal family of methodologies
- Extreme Programming (XP)
- Feature Driven Development
- Adaptive Software Development
- Open Source Software development

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!

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!)

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 kan også kan gi høy belønning/avkastning hvis det lykkes.

De 12 prinsippene

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

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

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

Planning game (Eggen)

- Det spiller ingen rolle hvem som kommer med de gode ideene - bare de kommer

Små releaser

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

Små releaser (Eggen)

- Like gode resultatprestasjoner vil oppleves dårligere hvis ikke resultatet blir tilført nye opplevelseselementer

Metaforer

- Få et vokabular (et sett 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

Enkelt design (Eggen)

- Det du ikke klarer å utføre på fredagstreninga, klarer du heller ikke å prestere i søndagskampen
- Hvis du ikke engang kan antyde løsninga på et problem du reiser, så er det ikke noe problem, ikke engang for deg sjøl!

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

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

Testing og kontinuerlig integrasjon (Eggen)

- Bare spillere som ofte er foran mål - scorer ofte mål
- Utsett aldri til i morra, det du kan gjøre i dag

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

Kollektivt eierskap (Eggen)

- Den viktigste motivasjonen er å føle seg som en viktig del av et samhandlingsmønster som virker
- Du kan ikke mene og si noe om en person til andre, som du ikke kan mene og si direkte til denne personen sammen med de andre

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?

On-site kunde (Eggen)

- Det skal være trygt og trivelig på Rosenborg-kamper, for alle på tribunen

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!)

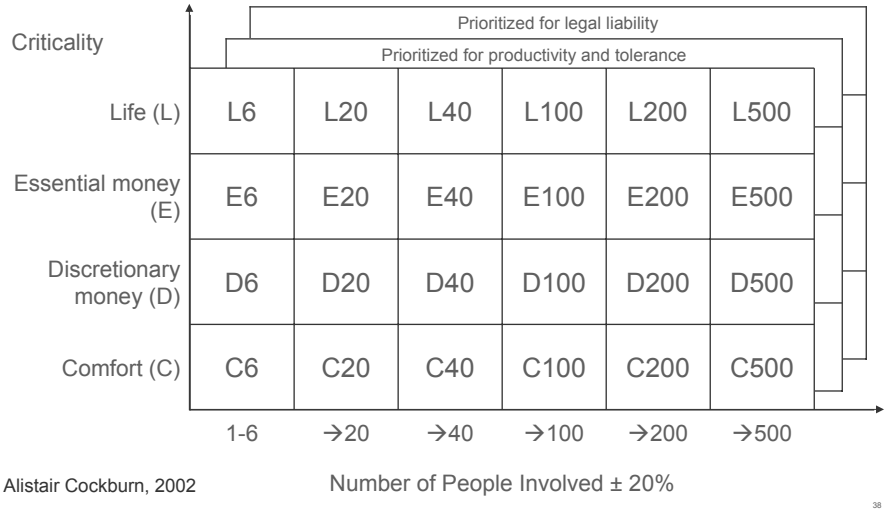
40-timers uke (Eggen)

- Bruk ego-energien til kollektivets beste

Kodestandarder

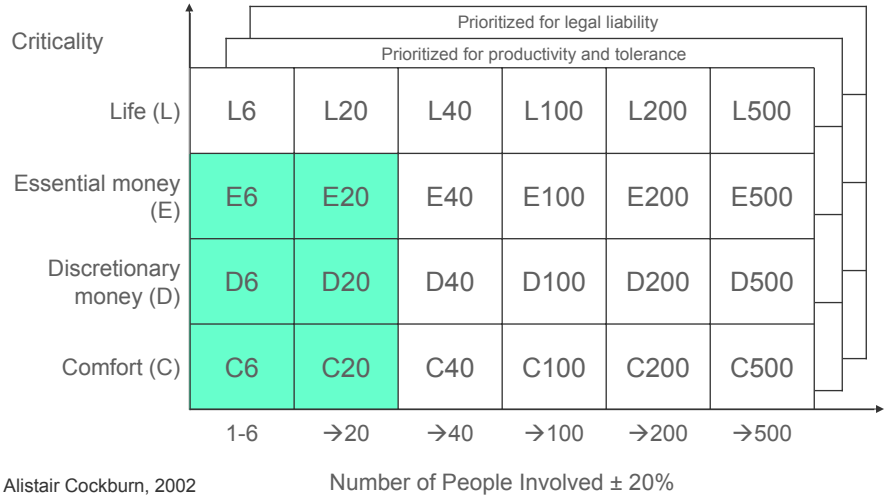
- Nødvendig for
  - Parprogrammering
  - Kollektivt eierskap
  - Disiplin
- Skal være enkel men dekkende

Når kan man benytte XP?



Alistair Cockburn, 2002

Når kan man benytte XP?



Alistair Cockburn, 2002

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.

Oppsummering

- Agile ≈ pragmatisk systemutvikling
- XP:
  - Mer en filosofi enn en komplett utviklingsmetodikk
  - Kodenær
  - Menneskeorientert
  - Lettvekt (prøver å fjerne overhead)
  - Praksisene er ikke nye, men ekstreme!
  - Praktisk Knowledge Management

Parprogrammering (PP)

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**!

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.)

PP – Roller

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

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)

PP (Eggen)

- Det er viktig å gå på banen for å være best mulig sjøl, men enda viktigere er innstillinga om å gjøre medspillerne gode
- Hvis du bare hører, glemmer du. Hvis du hører og ser, husker du. Men hvis du både hører og ser og samtidig handler, forstår du!

PP – Historie

- **1971**: *Egoless programming* (Weinberg)
- **1975**: *Surgical team and chief programmer* (Brooks – "The Mythical Man-Month")
- **1991**: *Collaborative Programming* (Flor and Hutchins)
- **1995**: *Dynamic Duos* (Constantine)
- **1995**: *Programming in pairs* organizational pattern (Coplien)
- **1996**: *Pair programming* (Kent Beck, XP)

PP – Påstander

I litteraturen hevdes parprogrammering å ha følgende fordeler (sammenliknet med individuell programmering):

- **Økt kvalitet** – par produserer kode med færre feil.
- **Reduserer leveransetiden** – par produserer kode med høyere kvalitet på omtrent halvparten av tiden.
- **Økt moral** – parprogrammerere er lykkeligere enn andre programmerere.
- **Bygger tillit og forbedrer team-arbeidet** – parprogrammering medfører bedre tillit til partneren og dermed økt team-"spirit".
- **Fasiliterer kunnskapsoverføring** – parprogrammerere, spesielt hvis man roterer partnere, vet mer om systemet (som helhet).
- **Øker læringen** – par lærer kontinuerlig av hverandre ved å diskutere ulike løsninger og ved å se på hverandres teknikker.

PP – påstander

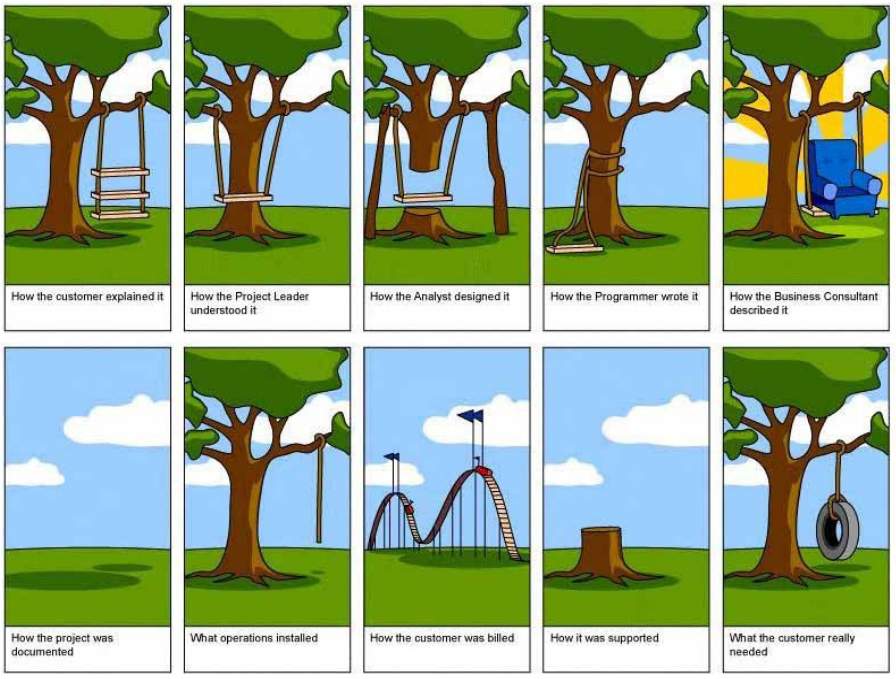
- Parprogrammering medfører dobbel kostnad (en oppgave – to utviklere)
- Parprogrammering er bare kontinuerlig kodelesing
- Komplekse oppgaver løses best alene
- Vanskelig å finne "gode" og effektive par

PP – oppsummering

- er en teknikk for å øke samarbeidet og kommunikasjonen mellom utviklerne
- er en egenskap som må læres og som krever mye ulik kompetanse
- Trenger et sett med retningslinjer for hvordan det skal gjennomføres (for prosjektledelse og utviklere)

PP – Praktiske tips

- Lag kjernetid for parprogrammering, f.eks. fra kl. 10-14
- Bestem graden av parprogrammering (og partnere) gjennom regelmessige stand-up møter
- Hvis test-first praktiseres: Lag testene i par og implementer individuelt!
- Paret deler samme kontor og har kontinuerlig dialog
- Aktivt samarbeid krever regelmessige pauser!



30.09.2004

[simula . research laboratory]

Takk for oppmerksomheten ☺