Agil developement

Grundidé

- Lättförändligt
- Kommer alltid finnas oförutserbara förändringar

- **Filosofin**
    - **Individuals and interactions** over process and tools
    - **Working software** over comprehensive documentation
    - **Custyomer collaboration** over contract negotiation
    - **Responding to change** overflowing a plan
- **Manifestot**
    - Early and continuous delivery of valuable software
    - Welcome change
    - Deliver software frequently
    - Bussinesspeople and developer must work togvether daily
    - Get motivated people, give them support and trust them
    - Face to face
    - Working software is progress
    - Self-organization
    - Reflect of how to be more effective

Viktiga begrepp

Detaljer

XP

Grundidé

- Pair programming
- Mycket kommunikation
- Exploratativ, prototttypande
- Del av problemet att flrstå problemet

- Högiterativt agil
- Körbar produkt tidigt
- Fokus på test
- Muntlig kommunikation
- Små inkrement - feedback i varje steg
- Designa inte i förväg. Ta små steg istället med testfall och refaktorisera

Enkel design

- Tydlig, lättläst (namn, inte duplicering, motiverad komplexitet utefter testfallen)
- Varför?
    - XP:
        - Komplicerad intiial tid gör att man slösar tid i slutet
        - Vi ska kunna/vilja ändra koden
        - Komplicerad slutlig design bör göras i smpå steg
- Hur?
    - Ej fullständig design från början
    - Diskutera möjligheter
    - Lägg till mönster efter behov
    - Diskutera designen när den kodas
- **Slogans**
    - **Do simplest thing that can possibly work**
    - **YAGNI**
        - You arent gonna need it (Tänjk inte på framtiden)
    - **Undvik big upforomt design**

Viktiga begrepp

Iterationsdelarna

1. **Analys**
    a. **Gör stufier vad vi vill ha (analys)**
        i. Chef/it ofganisation (fel synvinkel, bra översyn
        ii. Intervjua användsre (svårt att föreställa sig, ofta förenklas
        iii. Etnologiska studer (observera användarna)
        iv. Tar lång tid: ersätt delar med prototyping
    b. **Kravspecifikation**
        i. Skriv ner exakt vad vi vill ha
2. **Design**
    a. **Hur kan den byggas. Modular design**
    b. **Ritning (rita upp lösningen)**
3. **Produktion**

    a. **Varje steg ska avslutas efter nästa börjas**
        i. **Signeraing ochy godkännande**
            1. Dvs av analysen och designen för att produktionen är dyr och allt måste bli rätt
4. **testning**

När forumuleras kraven?

- Bra att komma på så många från början men de behöver uppdateras under preojektets gång

---

Utvärdering av systemet

- **Veriffiering**
    - Byggde vi systemet helt rätt enligt vår tolkning
- **Validering**
    - Byggde vi rätt system
    - DVS byggde vi vad kunden och användare vill ha
- **Hur?**
    - Kodgranskning, testning

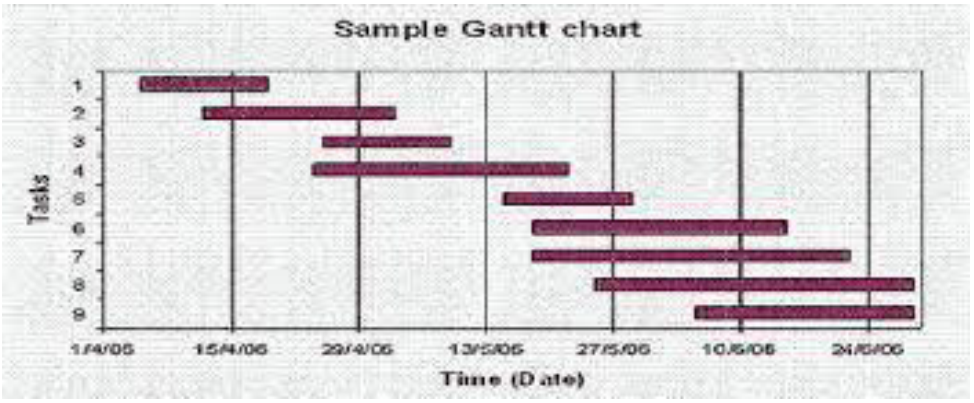
Iterationerna

1. **Planering**
    a. **Planering och deluppgifter**
    b. **Priopritera kvalité och tid överr funktionalitet**
    c. **Hållbart tempo**
    d. **Praktiskt**
        i. Skriver story, planerar stoy, esstimerar tid för story
        ii. Utvefcklarna lär sig hur många enheter de klarar
        iii. first iteration (version zero): Planera för att ha ett system i drift asap
            1. Den gör inget användbart, bara intiuakla arkitekturen (struktur, verktyg, osv påå plats)
            2. Mål är minimalt systerm som kan köras och **levereras**, med minimalfuktionalitet
        iv. Vill planera metaforer av hur det är inte koden
            1. Gemensam syn i teeam, källa till namn, kund förståelse, bra akritekturbeskrivning som undviker planering i förvg
        v. Team communicates informally (undviker oput of date document)
            1. Documentation: inte fast, man gör istället snabb zero release
            2. Ändras efter hand
            3. Kan dokumentera lite noggranare när allt är klart
            4. Dokumentera så man hittar
        vi. Spikes: quick throw-away explorations into potential solutions
            1. Inte vet hur problem löses: snabb prototyplösning
            2. Inga test: det är inte produktionskod: den kastas bort
        vii. Small releases
2. **Utveckling**
    a. **Skriv testning**
    b. **(inte) Uppdelning av ansvar**
        i. Vill inte ha en specifik grupp som skaa äga en del av koden eftersom det ger brist på förståelse i resten av teamet
3. **Kod och design**

    a. **Enkel design, begriplig kod**
    b. **Refaktionering**
    c. **Kodningstandards**
    d. **Gemensamt vokabulär**
4. **Dokumentation**
    a. **Kort, överenskommen**
        i. Enkel att förändra (både fysisk och mentalt lol)

Stories

- Korta, enkla, informationsrika
- Fråga frågor till kunden



GANTT diagram

[565 × 231]

- Tid för vad allt tid tar och dess beroenden
    - Hierkin av vad som måste göras före vad
    - Om saker görs parallellt skriv det

Critical chain

- Ta med olika mänskliga beteende för att skapa en metod
    - Student syndrome
        - Starting the last possible moment
    - Parkinsons law
        - Pacing the completion of the task throughout the full duration assigned to it
    - Multitasking: vi sugar på det

Den kritiska linjen är största linjen av saker som är beroende av varrande

- Tyå det minsta kravet men most essential kraven för att klara projektet (the main path)

På den kritiska linjen kör man så snabbt man kan

De andra uppgifterna är side tasks som görs parallellt till den kritiska kedjan

- Dessa ska ha en feeding buffer
    - Sidotasken ska aldrig hindra den kritiska linjen (därför används buffern)

Dev Ops

- Development Co-oporations

DevOps toolchain:

- Slå ihop så att Development och Co-oporations
- En chef som har koll på de båda sakerna
- Prata med varandra väldigt mycket
- Den ska snurra framåt

- Om något blir fel, snurra den baklänges
-

Configuration mangement

- **Software CM**
  - organisera, controller och manage utvecklingen av system
  - Hjälpoa till med all den där tiden när utvecklare inte sitter och kodar
- Består av flera delar. Se nedan. Vill bestämma hur man gör allt det.
- Behövs för maintence
- Behövs för att education, kudnen, developers, cheferna osv ska ha koll och kan enkelt gör input

1. **Configuration label**
   - **consists of setting and maintaining baselines, which define the system or subsystem architecture, components, and any developments at any point in time. It is the basis by which changes to any part of a system are identified, documented, and later tracked through design, development, testing, and final delivery. CI incrementally establishes and maintains the definitive current basis for Configuration Status Accounting (CSA) of a system and its configuration items (CIs) throughout their lifecycle (development, production, deployment, and operational support) until disposal.**
2. **Configuration control**
   a. **Change management**
      i. Sätta upp processen för hantera förändring
      ii. …
   b. **Traceability**
      i. Spårbarhet av förändringar
      ii. vad ändrades, vad ledde till beslutet, varför ändrades kod, av vem, syfte
3. **Configuation Status Accounting**
      i. Att hålla reda på status av allt
4. **Configuration auditing**
   a. **Physical audit**
      i. Granskat att hålla koll på allt som borde finnas med finns med
      ii. EX: Kom allt som vi skickade till kunden med?
   b. **Fuctional audit**
      i. Gör alla tester, kolla om funktioner fungerar (och ör rätt)
5. **Release management**
      i. Vad ska vi göra innan release.
      ii. Exempelvist skrivas ner det vi skickade till kunden
   b. **Build**
      i. **Bygga ihop produkten**

Refactoring

Ändra kod utan att ändra beteende

Krychten 4+1 views on design (hur man gör en blueprint på design?)

- Logical view
  - Vilka viktiga klasser och objekt finns (översiktliga UML klassdiagram)
- The process view

        - Vilka viktiga parallella processer finns
- The development view
    - Hur är mjukvaran modulariserad? vilka viktiga subsystem finns (som kan skapas av olika team)
- The 5th view - scenarios
    - Några få viktiga scenarion som illustrwera ovan

Participatory design
Evolutionary protoping med ständing medverkkan av användare
ASnvändare bidrar med åde problem och lösning
Bra att kombinmera med XP

Detaljer

aIntern beställare
- Annan del av samma företag

Extern beställare
- Annat företag

Utvecklarana själva
- Tänk open souce projekt

Funktionella krav
- Vad systemet ska göra

Icke-funktionellt krav
- Egenskaper och begränsingar
    - Ex minnesstorlek och svarstid
-

Problems of co-ordination
Shared data
- Om man delar data så är man inte nödvändigtvist överens om vad den ska ändras
- Team B vill ändra, Team A har en release nästa vecka och databasen får inte ändras

Double maintenance
- Kan inte bara kopiera den för att systemet inte längre är synkat
    - Måste nu ändra funktionaliteten på två ställen
- Måste maintain koden till både databaserna

Simultaneous update
- Därför vi har versionshanteringssystem
- Om man råkar jobba på saker samtidigt
- Inte ska skriva över någon annans ändringar med sin kod

Stategier
- Konservativ är bra för viktiga/komplicerade filer

Concurrent development

- **Optimisitsk**
    -
- **Konservativ**
    - Bara en ändring i koden

Developement strategy

- **Optimisitsk**
    - Optimistic, snab, bygger med iterationerna.
        - Byfgger på en puzzelbit i taget
- **Konservativ**
    - Långsamt, gör allt individuellt. Har massor protocol av hur alla puzzelbitar ska sitta ihop. Tillsits sätter man ihop allt

Update strategy

- **Optimisitsk**
    - Skickar in förändringar live eller iaf snabbt
- **Konservativ**
    - Git bestämmer själv när man får andras ädnrignar

Models

Long transactions

- Tar mer tid innan man fördelar information med varandra
- Det är dock skönt att committa så tidigt som möjligt
- Incintament för att ta mindre stories med korta iterationer
    - annars får man ju en mergeconflict

Strict long transactions

- Tar ut hela historiken på det som har hänt
- När man sedan pushar så pushar man hela historiken
- Jämför inte bara indviduella filer
- Lite oklart
-

Centralized rep vs Distributed version

- **Centr:**
- **Distr:**
    - **Git**

Shotgun surgery

- När man vill ändra (refactor) något måste man attackera på flera ställen
- lösning: Flytta alla påverkade metoder till en ny klass
    - KALLAS MOVE METHOD

Tillbakablick för kontext

Scrum

Grundidé

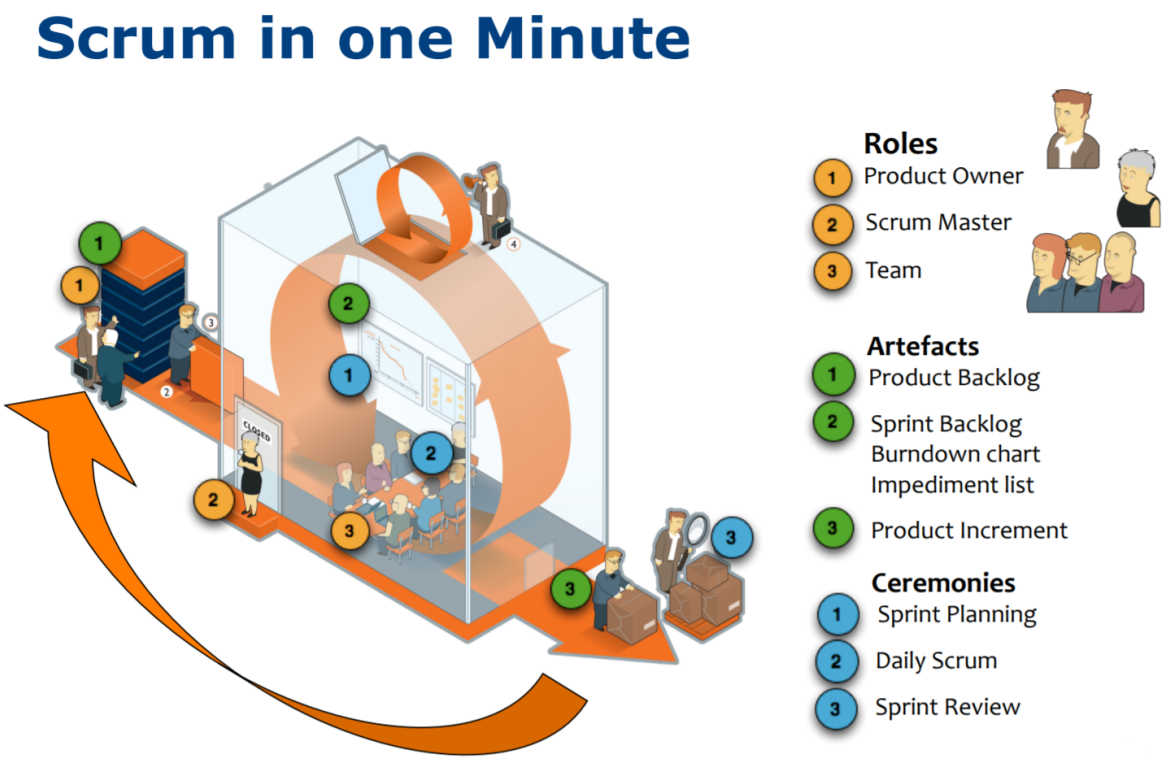
;;Mindre flexibel än XP
Man tillåter inte förändringar under en 0.5 till 1 mpnad sprint
I XP görs prioteringen av ostumer
I scrum görs prioriterting av aprojekt manager

- **Även om teamet oprganiserar hurdom ska uppnå det**

- Scrumteam
  - 5-10 självorganiserade
- Produktägare
  - Fysisk eprson
    - Styr priproteting, ofta endel av företaget. Ekonomisk ansvar
- Scrummaster
  - Coach, projektledare, skyddar teamet

Sprint: ta något i backlog ochj lös dety
Sprint review: ge tills takeholders
Sprint retrospective: effetctiviswera programemt'
daily scrum: kort tid av att diskutera processsen i nuläget

-



Viktiga begrepp

Detaljer