

F2

XP – Extremprogrammering översikt

EDA260

Programvaruutveckling i grupp – Projekt

Boris Magnusson, Görel Hedin
Datavetenskap, LTH

Vad är XP?

- En metod för hur man utvecklar programvara
 - i grupp
 - i nära samspel med kunden
 - med täta releaser
 - med hög kodkvalitet
 - som skall kunna förändras och leva under lång tid

XP – en agil metod

agile – lättroilig, vig

”The agile manifesto”:

viktigt	ännu viktigare
arbetsprocesser och verktyg	individer och interaktion
dokumentation	fungerande programvara
kontrakt med kunden	samarbete med kunden
följa en plan	kunna hantera ändringar i planen

Varifrån kommer XP?

Smalltalk-traditionen: 1972, 1980, ...

- dynamiskt typat OO språk och integrerad programmeringsmiljö
- "everything is an object"
- programmeringsstil: explorativ, prototypande, ...

Kent Beck & Ward Cunningham –
pionjärer inom OO design

- CRC-cards 1989
(Class, Responsibility, Collaboration)
- Patterns 1987
- XP 1996, 2001

XP's deltekniker ("practices")

Kodning och design

Enkel design

Refaktorisering

Kodningsstandard

Gemensam vokabulär

Planering

Kund i teamet

Planeringsspelet

Regelbundna releaser

Hållbart tempo

Utveckling

Test-driven utveckling (TDD)

Parprogrammering

Kollektivt kodägande

Kontinuerlig integration

Dessutom

Gemensamt utvecklingsrum

Nollte iterationen

Spikes

Planering i XP

Börja med en enkel plan
Planera om efter hand

Jämför: köra bil till Italien...

Planeringsspelet

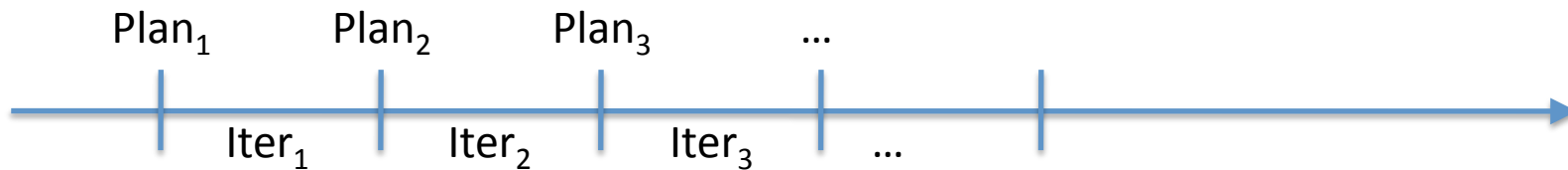
(The Planning Game)

Vad skall vi utveckla?

Hur lång tid tar det?

Vem bestämmer vad?

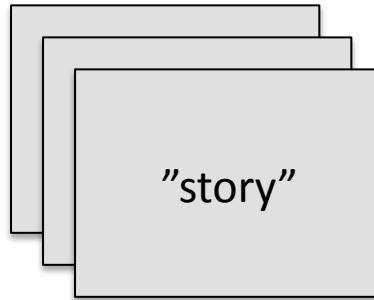
Kunder och utvecklare möts regelbundet för ett *planeringsspel* – för att planera nästa iteration



Exempelvis kan varje iteration vara 2 veckor

Kunder

Skriver "user stories"
(enkla användningsfall)
Prioriterar stories



Utvecklare

estimerar "poäng"
(motsvarande tid) för
varje story

Relativ estimering –
hur svår är denna "story" jämfört med andra?

Hur mycket hann vi sist? –
Så mycket hinner vi nog nästa iteration också.
"Yesterday's weather"

Prioritering – vad är viktigast just nu?

”User stories”

(användarberättelser)

Exempel:

Visa vem som ringer

När telefonen ringer skall namnet på den som ringer visas på displayen – om namnet är inlagt i telefonboken.

Annars skall numret visas.

Enkel, informell. ”Stories are promises for conversation”
Lagom stor – man bör hinna med flera stories varje iteration.

Labb 1: Extreme Hour

Ett rollspel som illustrerar planeringsspelet.

- en "produkt" tas fram på en timma
- vi ritar i stället för att implementera på riktigt

Regelbundna releaser

Release Regularly / Small releases

- Vad innebär det?
 - Releaser till kund skall göras *ofta*, och med små inkrement
 - Den första releasen begränsas i storlek så mycket som möjligt så att normalfallet är att vi har en releasad produkt
- När?
 - Typiskt efter varje iteration
 - Ibland kontinuerligt (kunden har möjlighet att ladda ner den senaste integrerade versionen)
- Varför?
 - Programmerarna får snabb feedback
 - Nya krav kan påverka vidareutvecklingen
 - Lätt att göra omprioriteringar
 - Kunden kan tidigt börja använda nya funktioner

Kund i teamet

Add a Customer to the Team / On-site customer

- Vad innebär det?
 - En ”kund” (eller representant för kundens intressen) finns på plats i projektet
- Varför?
 - Utvecklarna får omedelbar tillgång till en presumtiv kravställare
 - Stories behöver inte utarbetas så detaljerat
 - detaljer kan besvaras direkt
 - Muntlig kommunikation är MYCKET snabbare än skriftlig
 - Det blir en låg tröskel för att överhuvudtaget fråga

Gemensam vokabulär

Common vocabulary / System Metaphor

- En enkel beskrivning av hur systemet fungerar och som kan förstås av både kunder och utvecklare
- Ofta i form av en metafor, t.ex.
 - skrivbordsmetaforen för grafiska operativsystem
 - kundvagnsmetaforen för on-line shopping-system
- Varför?
 - Kunder och utvecklare kan lättare diskutera
 - Förenklar namngivning i programmet

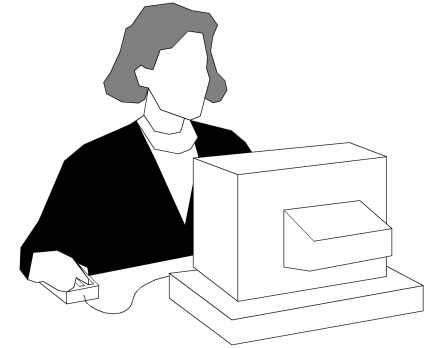
Utveckling i XP

- Hög kodkvalitet:
 - Enkel att förstå, enkel att modifiera
- Snabb feedback till:
 - utvecklaren själv
 - andra i teamet
 - kunden

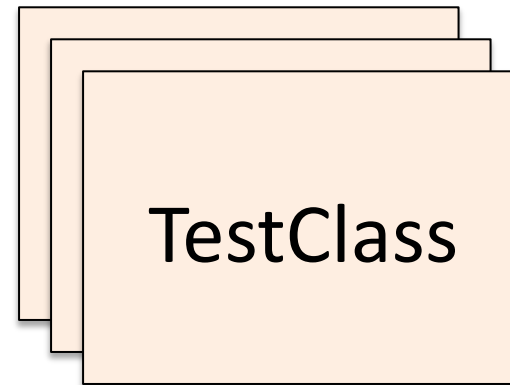
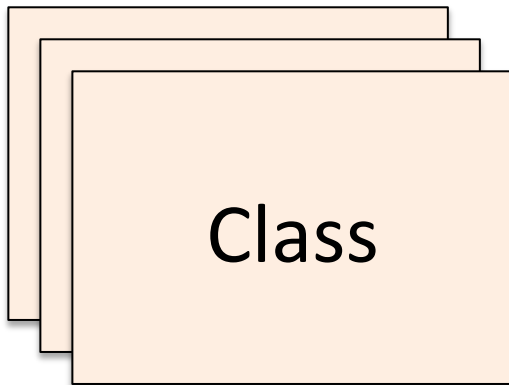
Testning i XP

- Enhetstestning
 - Enhetstester (unit tests) för varje klass/modul
 - Ett testfall skrivs *innan* motsvarande kod (TDD – Test-Driven Development)
 - Testfallen automatiseras (regressionstestning) (vi kan enkelt köra om alla testfall efter en ändring)
- Acceptanstestning
 - Acceptanstester för varje story
 - Även acceptanstestfallen automatiseras

Enhetstester

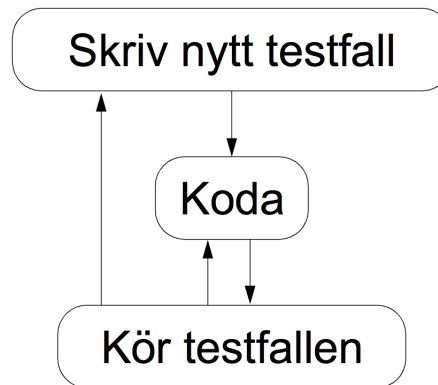


- Skrivs av programmeraren
- Måste fungera till 100% innan man får integrera
- Ackumulerar förtroende för koden
 - gör att man vågar ändra i koden



Test-Driven Development

- Testfallet skrivs *innan* koden!
 - fungerar som specifikationer – vad skall koden göra?
 - skrivs i mycket små iterationer: testa...koda...testa...koda...
 - körs automatiserat – alla testfall körs efter en ändring (så ser man att man inte förstört något som fungerade tidigare)
 - när testprogrammen fungerar är man färdig!



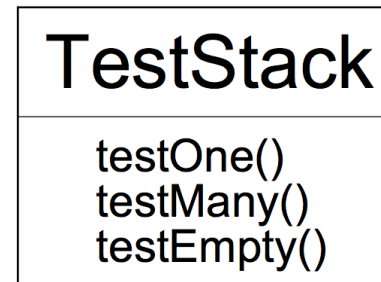
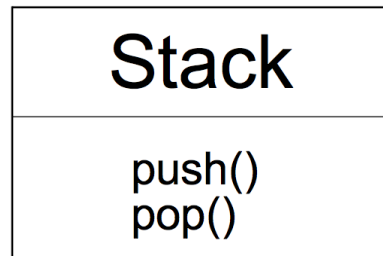
xUnit

- Enkelt ramverk och interaktivt verktyg för att automatisera testning
- Finns för flera språk
 - junit – för Java
 - sunit – för Smalltalk
 - cppunit – för C++
 - ...
- Grundidé:
 - Testfall kan läggas till mycket enkelt i koden
 - Verktöget kan köra alla testfall
 - Verktöget kan visa vilka testfall som inte fungerar

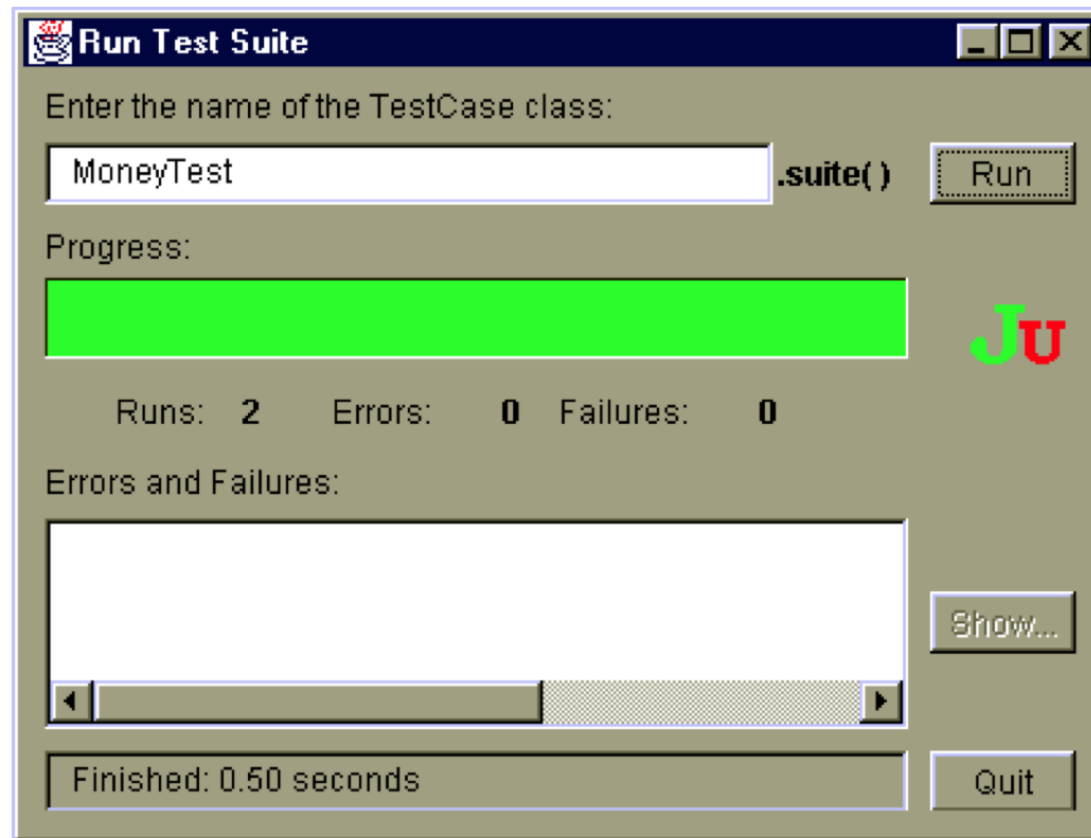
Testfall

- För varje klass A i produktionskoden, gör en testklass TestA
- För varje testfall, gör en metod i TestA som anropar metoder i A och testat resultatet

Exempel:



jUnit användargränssnitt



”keep the bar green,
keep the code clean”

Erfarenheter av TDD

jämfört med att skriva testerna *efter* kodningen

- Utvecklarna gillar det
 - känner sig säkrare på att man implementerat rätt
 - vet när de är färdiga
- Bättre design
 - man tänker igenom användning innan man implementerar
 - koden blir designad för test
(ofta svårt att lägga till testfall i efterhand)
- Hur mycket testar man?
 - Krävs erfarenhet för att skriva bra testfall, på lagom detaljnivå.
 - Det tar längre tid att skriva testfallen än att skriva produktionskoden.
 - Testkoden utgör cirka 25-50% av all kod. Ibland ännu mer.

Acceptanstester

- Kunden tänker ut testfall för stories
 - ”Vad skulle övertyga mig om att denna story är implementerad?”
 - Utvecklare hjälper kunden att implementera testfallen
- Testfallen mäter hur projektet framskrider

Kodning och Design i XP

- Komma igång så fort som möjligt
- Hålla hög kodkvalitet

Enkel design

Code and Design Simply / Simple Design

- Vad innebär det?
 - Ren tydlig kod, goda namn
 - Ingen duplicerad kod
 - Ingen onödig komplexitet (all komplexitet skall vara motiverad av dagens behov – testfallen)
- Enkel design innebär att programmet är lätt att förstå och ändra i

Enkel design växer fram

Designen växer fram för att passa de testfall som finns idag

- I motsats till "Big upfront design"
(när man designar för morgondagens behov)
- Varför?
 - Vi vet inte om en "big upfront design" verkligen kommer att passa förrän vi har implementerat kraven.
 - Vi vet inte om en "big upfront design" verkligen kommer att behövas – kanske alla delar av designen inte kommer att utnyttjas, kanske ändrar projektet riktning
 - Vi är inte rädda för att ändra kod och design när vi behöver det
 - En komplicerad design blir bättre om man gör den i många små steg, med feedback från testfallen i varje steg.

Slogans om Enkel Design

”Avoid Big Upfront Design”

”Don’t design on speculation”

”Do the simplest thing that could possibly work”

Obs! Detta betyder inte att man kan nöja sig med att göra enklast möjliga *ändring*. Efter att i har gjort en ändring måste vi ta ett steg tillbaka och kontrollera att vår design fortfarande är enkel, dvs har god struktur, bra namn, ingen duplicerad kod, etc. Har den inte det måste vi *refaktorisera*.

Refaktorisering

(Refactoring)

- Omstrukturering av koden *utan att ändra beteendet*
- Exempel
 - **Rename Method** (byt namn på metod, och alla anrop till den)
 - **Extract Method** (bryt ut ett stycke kod till en egen metod)
 - **Move Method** (flytta en metod från en klass till en annan)
- Varför?
 - Åstadkomma och upprätthålla Enkel Design
- Hur?
 - med verktyg (Eclipse, Smalltalk refactoring browser, ...)
 - för hand (jobbigt)

När gör man refaktorisering?

- När man läser koden
 - för att förstå koden bättre
- När man skall införa en ändring
 - för att lättare kunna göra ändringen
- Medan man implementerar ny funktionalitet
 - för att förbättra sin egen lösning
- När koden börjat ”lukta illa”
 - och man inte längre har Enkel Design

”Bad smells in code”

- Duplicated code
- Long method
- Large class
- Long parameter list
- Speculative generality
- ...

Parprogrammering

(pair programming)

- Två personer vid en maskin!
 - Driver och Partner. Växlar ofta roller.
 - Paren kan växla flera gånger per dag.
- Parprogrammering innebär automatisk kodgranskning



Rapporterade erfarenheter

Parprogrammering:

- Fokuserat arbete
 - Man lägger inte tid på sidospår
 - Man arbetar mer effektivt (inget paus-surfande)
 - Man blir trött!
 - Man blir inte avbruten av andra
- Kvalitet
 - Högre kodkvalitet (bättre detalj-design)
 - Man håller sig lättare till disciplinen (TDD, refaktoriseringar, ...)
 - Man lär sig mycket av sin partner (design, verktyg, ...)
- En del är negativa innan de provat, därefter positivt överraskade
- Speciellt användbart
 - när man debuggar svåra problem
 - i inlärningsituationer
 - när man fasar in nya utvecklare

Gör 2 personer 1 persons jobb?

- Inte enligt XP-förespråkare som hävdar:
 - Utvecklare med god parprogrammeringsvana kan programmera dubbelt så snabbt i par som när de programmerar själva
 - Koden blir av mycket högre kvalitet

Kollektivt ägande

(Collective ownership)

- Vad innebär det?
 - Koden "ägs" gemensamt av alla i gruppen
 - Alla i gruppen får ändra i all kod
- Varför är det bra?
 - Man slipper "beställa" ändringar
 - man gör dem helt enkelt själv
- Kodningsstandard
 - Hur koden formatteras
 - Hur man namnger klasser, metoder, variabler

Kontinuerlig integration

(Continuous integration)

- När skall man integrera sina ändringar med huvudversionen?
 - Så snart ett nytt testfall fungerar!
 - Flera gånger varje dag!
 - Kräver smidiga versionshanterings-verktyg

Gemensamt utvecklingsrum

(Bullpen / Open Workplace)

- Hur
 - Alla som utvecklar kod sitter i samma rum.
 - Maskinerna är placerade så att man lätt kan prata med varandra.
- Varför?
 - XP bygger på muntlig kommunikation

Hållbart Tempo

(Sustainable Pace / 40-timmars vecka)

- Regeln i XP
 - Arbeta högst 40 timmar i veckan närhelst detta är möjligt.
 - Arbeta aldrig övertid mer än två veckor i sträck
- Varför?
 - Trötta programmerare gör många misstag och skriver dålig kod

Delteknikerna förstärker varandra



XP's "värden"

- *Kommunikation*
 - alla kommunicerar med varandra, och ofta (Kund i Teamet, Parprogrammering, ...)
- *Enkelhet*
 - enkla lösningar är bättre än komplicerade (Enkel Design, ...)
- *Feedback*
 - alla får feedback: kunder, utvecklare, projektledare (TDD, Planeringsspelet, ...)
- *Mod*
 - Utvecklarna vågar ändra i koden för att förbättra koden eller tillgodose nya krav (Refaktorisering, Kollektivt ägande, ...)

Dokumentation?

- XP fokuserar inte på *detaljerad* dokumentation
- Enkel informell dokumentation
 - Story-kort, diagram på whiteboard, enkla beskrivningar, ...
- Koden är det viktigaste dokumentet
 - Ren och tydlig kod ett måste
 - God namngivning på variabler, klasser, etc. ett måste så att koden blir läsbar
 - Acceptanstesterna fungerar som precis kravspecifikation
- Kundbeställd dokumentation
 - Mer noggrann dokumentation kan tas fram, beställd av kunden, planerad som stories

Rapporterade erfarenheter från lyckade XP-projekt

- Utvecklarna uppskattar...
 - ... att arbeta fokuserat
 - ... att snabbt åstadkomma resultat man får visa andra
- Projektledningen uppskattar...
 - ... att lätt kunna följa projektet
 - ... att lätt kunna ändra inriktning
- Kunderna uppskattar...
 - ... att kontinuerligt se resultat
 - ... att kontinuerligt kunna komma med nya ideer

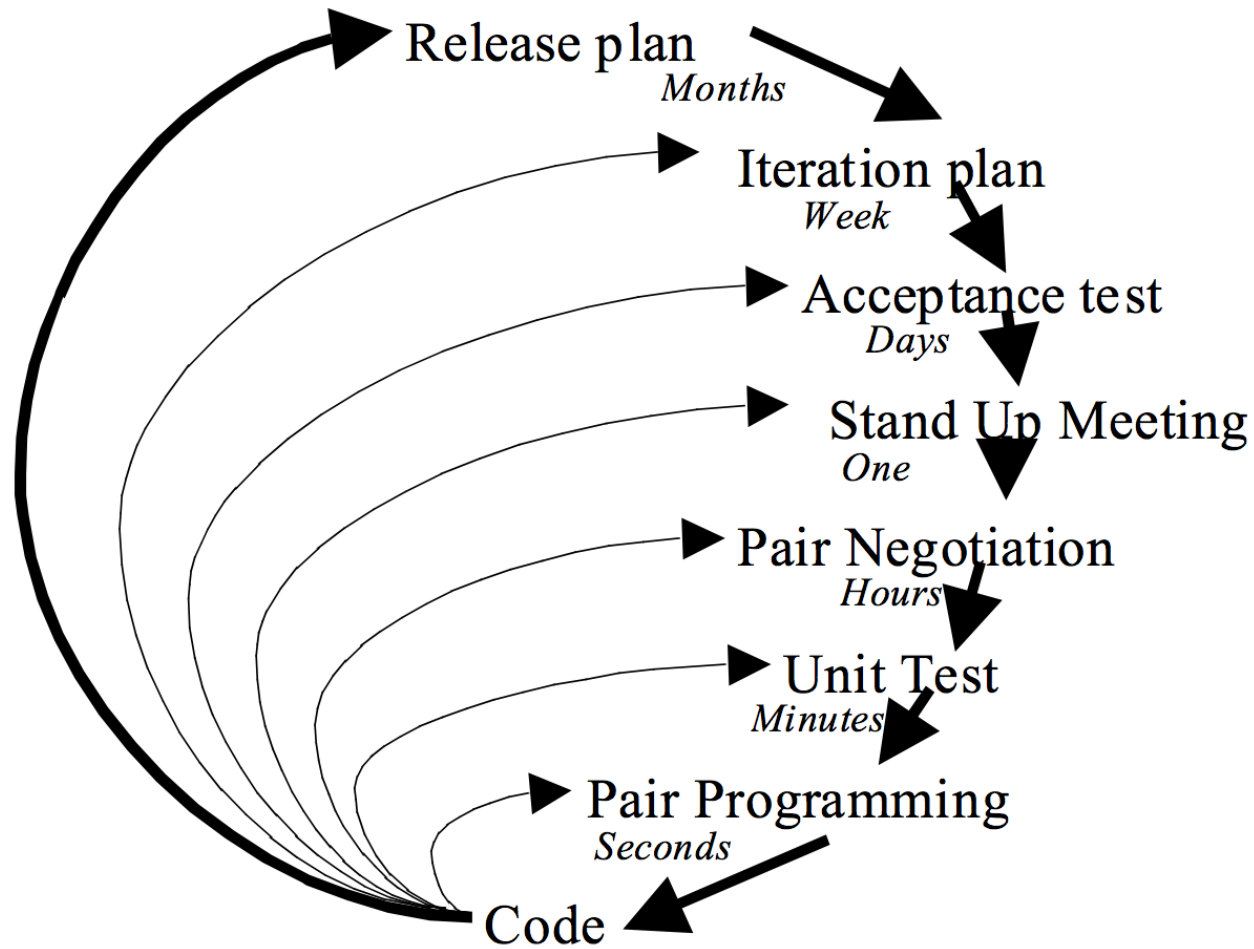
Hur stora XP-projekt kan man ha?

- Erfarenheter av XP:
 - XP passar små projekt (4-20 utvecklare).
Muntlig kommunikation och Kollektivt kodägande begränsar.
 - Planeringsspelet och Regelbundna Releaser passar även stora projekt (150-400 utvecklare)
- Hur kan man skala upp XP?
 - Stora projekt kan drivas som många små XP-projekt
 - Teamen behöver koordineras (XP behöver då kompletteras med tekniker för detta)

Varför kallas det ”Extrem programmering”?

- Korta utvecklingscykler är bra: I XP gör man dem extremt korta
- Testning är bra: I XP testar man hela tiden.
- Design är bra: I XP designar man hela tiden
- Kodgranskning är bra: I XP kodgranskar man hela tiden
- ...

Feedback i XP



Deltekniker i XP

Kodning och design

Enkel design

Refaktorisering

Kodningsstandard

Gemensam vokabulär

Planering

Kund i teamet

Planeringsspelet

Regelbundna releaser

Hållbart tempo

Utveckling

Test-driven utveckling (TDD)

Parprogrammering

Kollektivt kodägande

Kontinuerlig integration

Dessutom

Gemensamt utvecklingsrum

Nollte iterationen

Spikes

Målet med XP

- Mycket hög kodkvalitet
- Kod som kan ändras i takt med förändrade krav

Det finns många metoder

- Agila
 - XP
 - FDD (Feature-Driven Development)
 - SCRUM (kan kombineras med XP)
 - Lean
 - ...
- Traditionella (dokumentfokuserade)
 - Vattenfall
 - Spiral
 - Iterativ
 - RUP (Rational Unified Process)
 - ...

Labb 1 – Extreme Hour

- Förberedelser
 - Repetera F1 och F2 (OH-bilder på kurswebben)
 - Läs artikeln av Kent Beck (i häftet)
 - chromatic: Läs Part I (Why XP?)
 - Ta med papper och penna

 - Skriftligt labbförhör. Underkänd => ny labbtid
 - Ej i tid => ny labbtid

F3: Konfigurationshantering

(Nästa veckas föreläsning, Lars Bendix)

- Please note that:
 - the lecture will not cover all the literature
 - the lecture will cover more than the literature
 - part of the lecture is introduction to the CVS-lab
- Read *before* the lecture:
 - Häftet: Utdrag ur bok av Babich: Software Configuration Management – Coordination for Team Productivity
 - chromatic: Delar av Part II (XP Practices):
 - p 18-20 (Refactor Mercilessly)
 - p32-36 (Collective Code Ownership, Integrate Continually)
 - p 41-42 (Release Regularly)
 - Häftet: Utdrag ur bok av Jeffries et al: Extreme Programming Installed. Delar av Kap 11 + 15.
- Additionally, read *before* Lab 2 (CVS):
 - Kurswebben, labbsidan: Introduction to CVS
 - Kurswebben, labbsidan: Labbhandledning labb 2