F2
XP – Extrem Programmering 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 – lättrörlig, 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 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)
Parprogramming
Kollektivt kodägande
Kontinuerlig integration

Dessutom

Gemensamt utvecklingsrum
Nollte iterationen
Spikes

Planering i XP

Börja med en enkel plan

Planera om efter hand

Jfr köra bil till Italien ...

Planeringsspelet

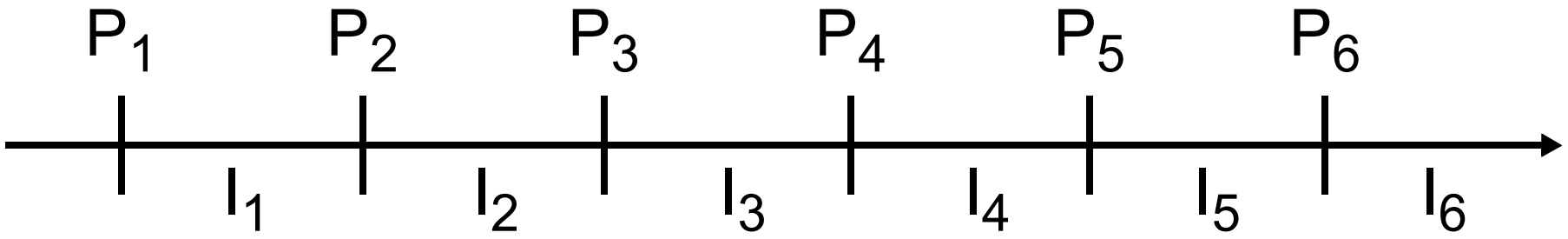
(The Planning Game)

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 “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)

T.ex.

> När telefonen ringer skall namnet på den som ringer visas på displayen – om namnet är inlagt i telefon-boken.
>
> Annars skall numret visas.

Enkel, informell. “Stories are promises for conversation”

Lagom stor – man bör hinna med flera stories varje iteration

Lab 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 alltid möjlighet att ladda ner 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 användare
- 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
- löpandebandmetafor för lönehanteringssystem

Varför?

- Kunder och utvecklare kan lättare diskutera
- Förenklar namngivning i programmet

Utvecklig i XP

Hög kvalité:

- identifiera fel så tidigt som möjligt

Snabb feedback till:

- utvecklarna
- andra i teamet
- kunden

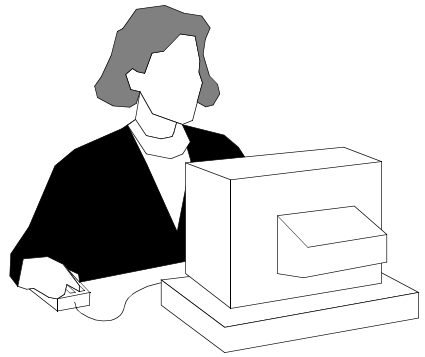
Testning

Testning är centralt i XP

- Enhetstester (unit tests) för varje klass/modul
- Ett testfall skrivs *innan* motsvarande kod (Test-Driven Development)
- Testfallen automatiseras (regressionstest)
  (vi kan enkelt köra om alla testfall efter en ändring)

- Acceptanstester för varje story
- Även acceptanstestfallen automatiseras

Enhetstester

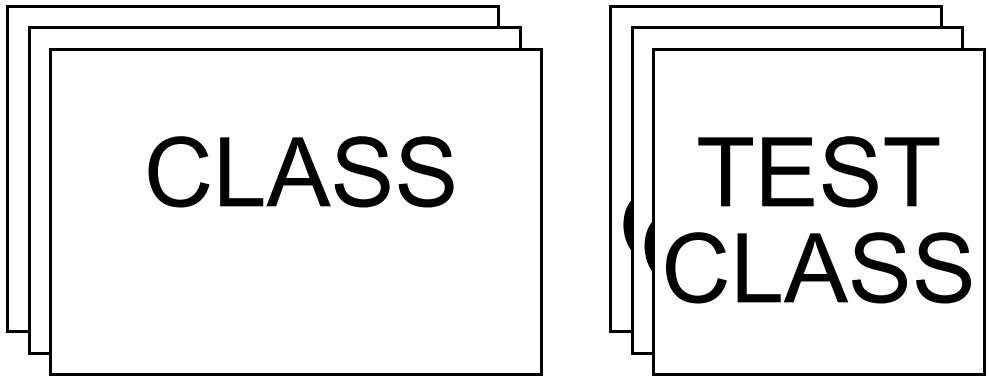


skrivs av programmeraren

måste fungera till 100% innan man får integrera

ackumulerar förtroende för koden

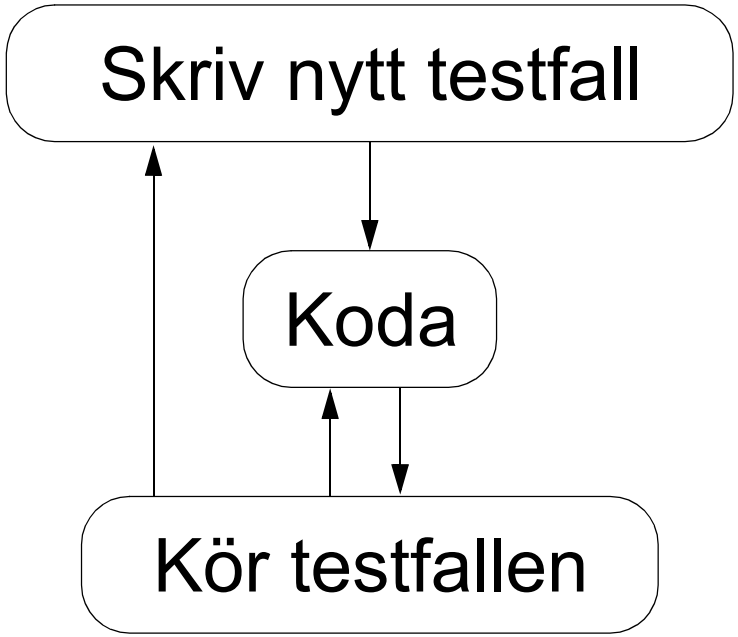
- gör att man vågar ändra i koden



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

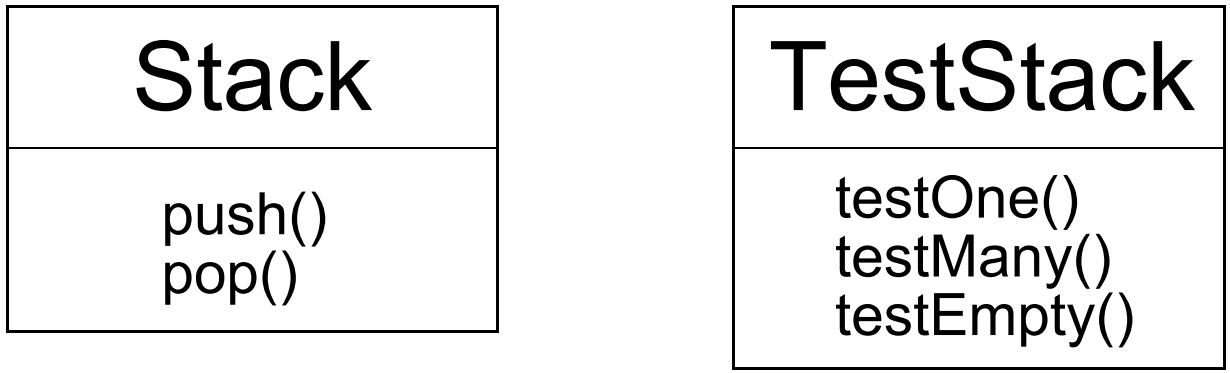
- jUnit – för Java
- sUnit – för Smalltalk
- cppUnit – för C++

Grundidé:

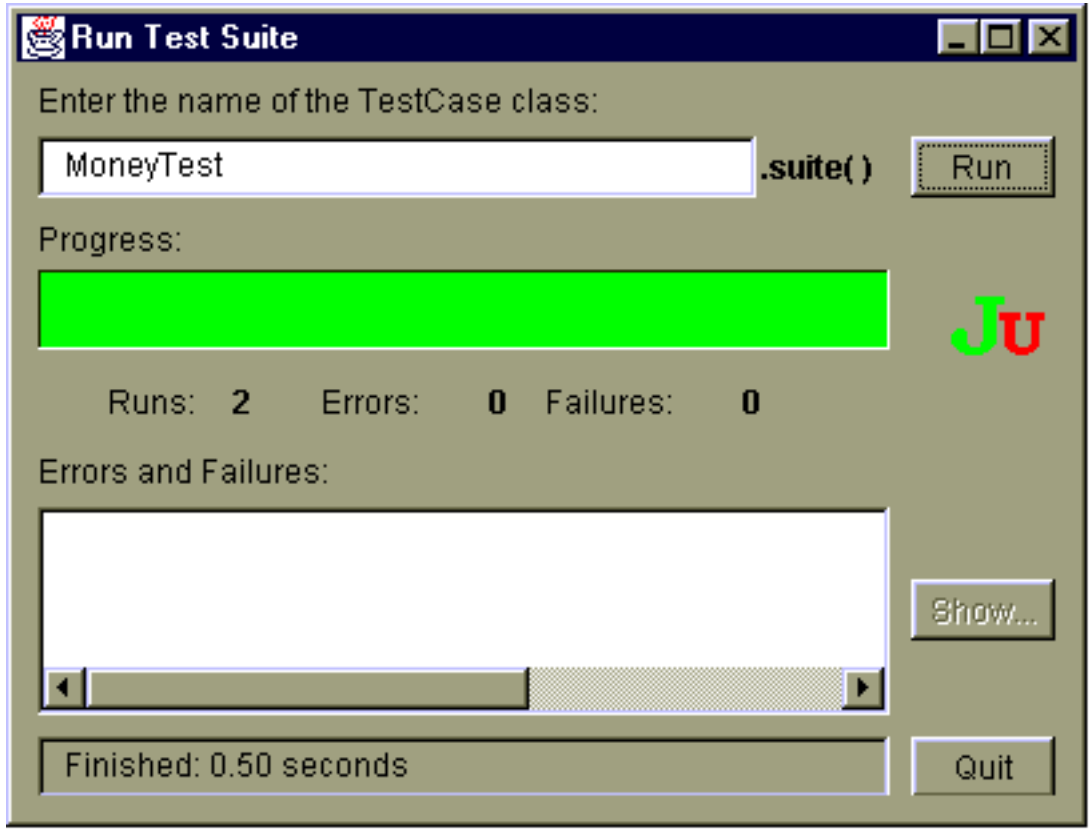
- Testfall kan läggas till mycket enkelt i koden
- Verktyget kan köra alla testfall
- Verktyget kan visa vilka tester som inte fungerar

Testfall

- Gör en test-klass TestA för varje klass A
- För varje testfall, gör en metod i TestA som anropar metoder i A och testar resultatet



jUnit användargränssnitt



"keep the bar green"

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
- Resulterar i mycket bättre design av klasserna
- Mycket lättare att skriva testprogrammen först
  (än att försöka skriva dem efteråt)
- Kräver erfarenhet för att hitta lagom nivå
  Vad skall man testa? Hur detaljerat?
- Det tar längre tid att skriva testerna än produktionskoden
- Test-koden cirka 25-50% av produktionskoden. Ibland ännu mer.

Acceptanstester

Kunden tänker ut testfall för stories

- "Vad skulle övertyga mig om att denna story är implementerad?"
- Programmerare hjälper kunden att implementera testfallen

Testfallen mäter hur projektet framskrider

Kodning och Design i XP

Komma igång så fort som möjligt

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 vi 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 en 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?

För att förstå koden bättre

- när man läser den för att göra en ändring

För att lättare kunna införa en ändring

Hela tiden, i små steg

- omfattande refaktorisering har man sällan tid med

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 & Partner. Växlar ofta.

Paren kan växla flera gånger per dag

Parprogrammering innebär automatisk kodgranskning

Rapporterade erfarenheter

Parprogrammering:

- Man arbetar mer fokuserat (lägger inte tid på sidospår...)
- Man arbetar mer effektivt (inget paus-surfande...)
- Man blir trött!
- Högre kodkvalitet (bättre detalj-design)
- Man håller sig lättare till disciplinen (TDD, refaktorisering, ...)
- Man lär sig mycket av sin partner (design, verktyg, ...)
- Lätt att fasa in nya utvecklare
- Man blir inte avbruten av andra
- Många är negativa innan de provat, därefter positivt överraskade

Gör 2 personer 1 persons jobb?

Inte enligt XP-förespråkare som säger:

- 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 som andra får göra – 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 konfigurationshanterings-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 andra.

Varför

- XP bygger på muntlig kommunikation

Hållbart Tempo

Sustainable Pace / 40-timmarsvecka

Regeln i XP

- Arbeta högst 40 timmar i veckan närhelst detta är möjligt.
- Arbeta aldrig övertid två veckor i sträck

Varför?

- Trötta programmerare gör många misstag och skriver dålig kod

Delteknikerna förstärker varandra

Kund i teamet

Planeringsspel

Hållbart tempo

Gemensam vokabulär

Enkel Design

Refaktorisering

Regelbundna Releaser

Test-Driven Utveckling

Parprogrammering

Gemensamt Utvecklingsrum

Kodningsstandard

Kontinuerlig Integration

Kollektivt Ägande

XP's "värden"

Kommunikation

- alla kommunicerar med varandra, och ofta
  (Kund i Teamet, Parprorammering, ...)

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

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 av design kan tas fram, ofta i efterhand, beställd av kunden, planerad som stories

Rapporterade erfarenheter från lyckade XP-projekt

Utvecklarna uppskattar...

- ... att arbeta fokuserat
- ... att snabbt åstadkomma resultat som man får visa andra

Projektledningen uppskattar...

- ... att lätt kunna följa projektet
- ... att lätt kunna ändra riktning (omprioritera)

Kunderna uppskattar...

- ... att kontinuerligt se resultat
- ... att kontinuerligt kunna komma med nya ideer

Hur stora projekt kan man använda XP för?

Erfarenheter hittills

- Full 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?

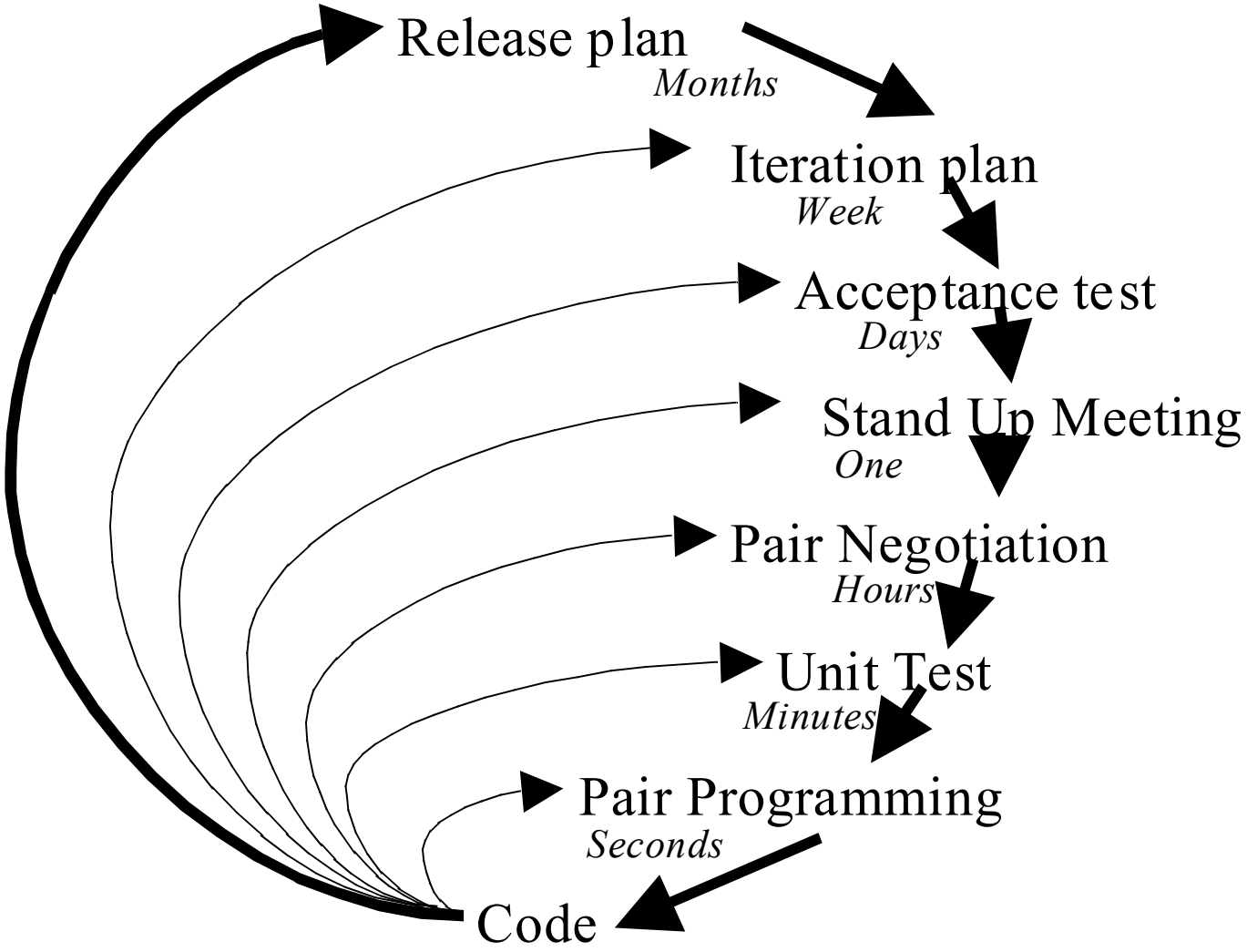
- Stora projekt kan drivas som många små XP-projekt
- Teamen behöver koordineras (kräver metoder som ligger utanför XP)

Varför kallas det “Extrem programmering”?

Officiellt

- 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
- ...
- “Turn the knob up to 10”

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)
Parprogramming
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)
- CRYSTAL
- ...

Traditionella (dokumentfokuserade)

- Vattenfall
- Spiral
- Iterativ
- RUP (Rational Unified Process) (Agila RUP-varianter diskuteras)
- ...

Lab 1 - Extreme Hour

Förberedelser

- Repetera F1 och F2
- Läs artikeln av Kent Beck
- chromatic: Läs Part 1 (Why XP?)
- Ta med papper och penna

- Skriftligt labförhör. Underkänd => boka ny labtid
- Ej i tid (>15 min för sent) => boka ny labtid

Labadministratör är:
gorel.hedin@cs.lth.se, tel. 2229644

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:

- Wayne A. Babich: Software Configuration Management - Coordination for Team Productivity, chapter 1.
- R. Jeffries et.al.: Extreme Programming Installed, chapter 11, 15, 21
- chromatic: p 18-20 (Refactor Mercilessly), p 32-36 (Collective Code Ownership, Integrate Continually), p 41-41 (Release Regularly)

Read before Lab 2 (CVS):

- Christian Andersson: Introduction to Configuration Management with Concurrent Versions System.
- Labhandledning