F6
Arkitektur, Planering

EDA260

Programvaruutveckling i grupp – Projekt

Boris Magnusson, Ulf Asklund
Datavetenskap, LTH

XP:s Deltekniker (Practices)

1. Planering **Planeringsspelet** **Regelbundna releaser** **Hållbart tempo** **Kund i teamet**	**2. Utveckling** Test-driven utveckling (TDD) Parprogrammering Kollektivt ägande Kontinuerlig integration
3. Kodning och Design Enkel design Refaktorisering Kodningsstandard **Gemensam vokabulär** **Arkitektur**	**4. Dessutom** Gemensamt utvecklingsrum **Nollte iterationen** Spikes

A) Mjukvaruarkitektur?

- Enkel Design och Refaktorisering
  - handlar i första hand om “design-in-the-small” och god kodkvalitet.

- Hur hanteras den överordnade designen i XP?

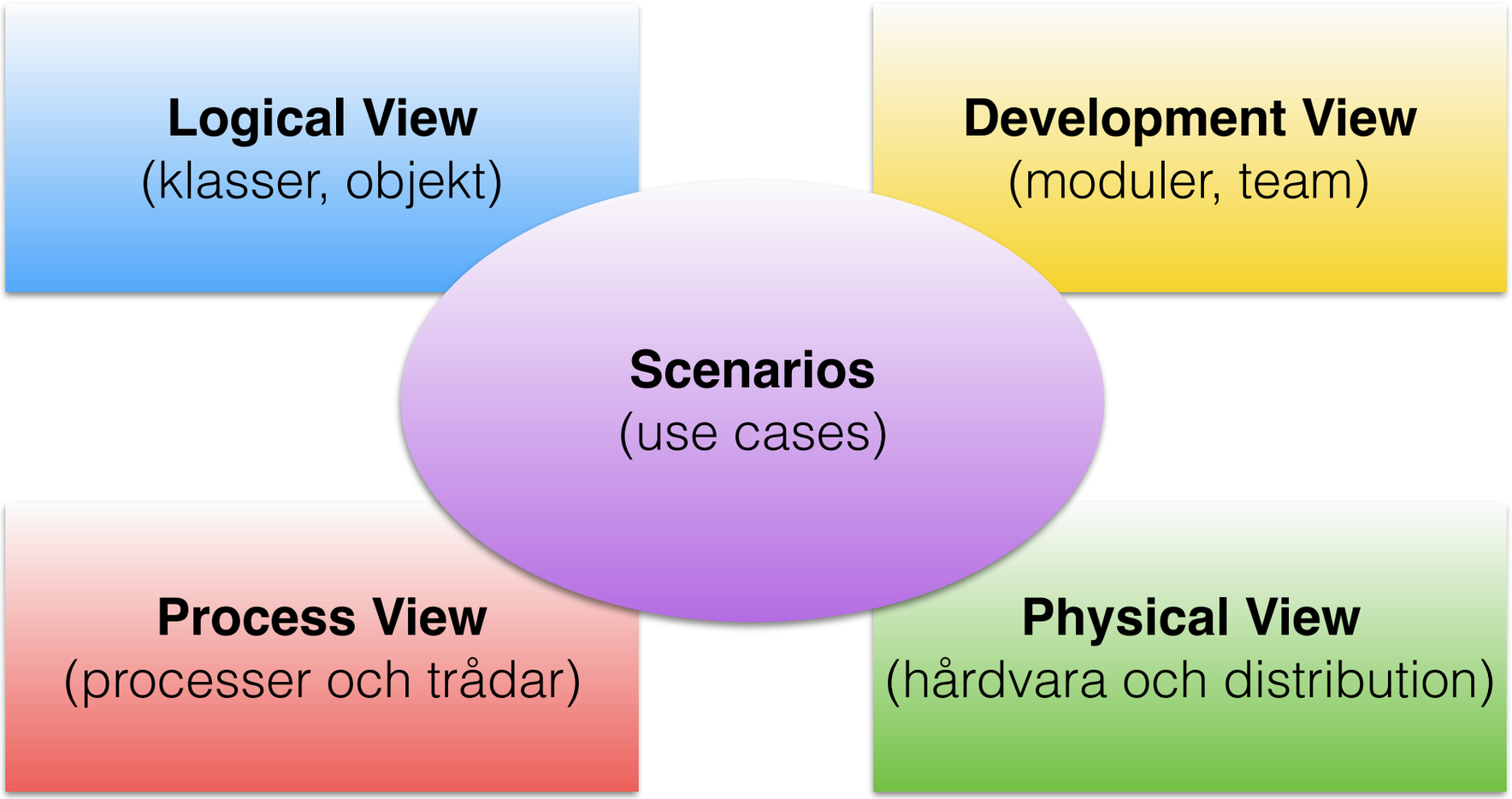
Vad är mjukvaruarkitektur?
Den övergripande designen

- Oftast används flera separata vyer
- Jämför med byggnader och ritningar:
  - bärande delar
  - vatten o avlopp
  - el
  - ventilation
  - personflöde (t ex för en flygplats)

Kruchten's "4+1" Views

- **The logical view**
  - Vilka viktiga klasser och objekt finns i systemet? (översiktliga UML-klassdiagram)
- **The process view**
  - Vilka viktiga parallella processer finns i systemet?
- **The physical view**
  - Hur är mjukvaran distribuerad på hårdvaran? (Kanske olika subsystem körs på olika maskiner)
- **The development view**
  - Hur är mjukvaran modulariserad? Vilka viktiga subsystem finns (som typiskt kan utvecklas av olika team)?
- **"The 5th view" – scenarios**
  - Några få viktiga scenarion som illustrerar de övriga vyerna

Kruchten's "4+1" Views



Hur stödjs arkitektur i XP? [W. Wake http://xp123.com/xplor/xp0007b/]

- **Traditionell syn**
  - arkitekturen har global inverkan på systemet och är svår att ändra – därför, planera och designa arkitekturen *först*

- **XP's syn**
  - "embrace change" - *bygg efterhand*

XP’s stöd för arkitektur

- **Spikes**
  - quick throw-away explorations into potential solutions
- **System Metaphor**
  - a story of what the system is “like”
- **First Iteration**
  - pick stories that result in an end-to-end (skeleton) system
- **Small releases**
  - weak areas show up quickly and can be corrected
- **Refactoring**
  - the architecture can be changed and evolved
- **Team Practices**
  - No out-of-date document – team communicates informally

Spike

(experimentell prototyplösning)

- **Om vi inte vet** hur ett problem kan lösas – gör ett experiment
  - snabb prototyplösning
  - målet är att se om en viss väg är framkomlig
- En spike ger inte produktionskod
  - experimentell kod – testfall etc., behövs ej
  - koden kastas (eller sparas som inspirationskälla vid den riktiga programmeringen)
- **Spikes hjälper oss att göra arkitekturella designval**

First Iteration

- Normaltillstånd är ett system i drift som **vidareutvecklas** (”underhåll”)
  - XP:s metodik - kom dit så fort som möjligt!
- **Första iterationen är speciell**
  - Vi skall gå från ingenting till normaltillståndet.
  - Vi behöver få en initial arkitektur på plats att börja fylla med funktionalitet.
- Mål med första iterationen
  - Första releasen gör ingenting (användbart), men på ett sådant sätt att hela den **initiala arkitekturen är på plats**. “Zero Feature Release”
  - Se till att få **verktyg**, systemkonfigurationer och användning av extern programvara på plats.
  - Målet är att åstadkomma ett minimalt system som kan installeras köras och **levereras**, men med minimal funktionalitet.

Exempel

Utveckling av ett interaktivt kalkylbladsprogram

- Första iterationen skulle kunna innehålla
  - Ett tomt fönster med titeln “Kalkylblad”
  - Ett menyval där man kan göra ingenting
  - Ett internt tomt modell-objekt
  - Lagring av det tomma modell-objektet i en databas.
- Då har vi
  - något som vi kan skicka och Kunden kan **installera och provköra**
  - **kontroll på verktyg** vi behöver använda: testverktyg, kompilator, makefiles, etc.
  - kontroll på **extern programvara** vi har tänkt använda (fönstersystem, databas-paket, ...)
  - ett **skelett-program** som vi kan börja lägga in riktig funktionalitet i.
  - en **initial arkitektur**: lagerindelning med databas, modell, view
- ***Vi kan börja arbeta!***

Systemmetafor

- En beskrivning av vad systemets implementation **“liknar”**.
- Varför?
  - Ger en **gemensam syn** på systemet inom teamet
  - Kan fungera som **källa till namn** på viktiga klasser i systemet
  - Kan fungera som **stöd för arkitekturbeskrivning** genom att identifiera nyckelobjekt och deras relationer
  - Kan möjliggöra att även en icke-teknisk kund kan **förstå** implementationsstrukturen i stora drag

Exempel på Metaforer

- Den “naiva” metaforen
  - T.ex. Customer och Account för en bank-tillämpning.
    Ingen riktig metafor. Objekten representerar helt enkelt sig själva.
- Desktop-metaforen
  - För grafiska användargränssnitt (Desktop, Files, Folders, ...)
- Assembly Line
  - Med löpande band, delar, stationer... [Chrysler C3 payroll system]
- Pipes and Filters
  - som i Unix, för att köra data genom en kombination av program
- Layers
  - Program uppdelat i lager, t.ex. tillämpning, högnivåprotokoll, lågnivåprotokoll, eller Model-View, ...

Dokumentation av arkitekturen

- **Traditionellt**
  - arkitekturen **fixeras** innan programmering
  - dokumentera arkitekturen **noga i början av projektet**
- **XP**
  - se till att snabbt få konkret **feedback** på idéerna om arkitektur (genom zero-feature release)
  - låt arkitekturen **utvecklas och omstruktureras** efter hand
  - dokumentera så att man hittar (”tunnelbanekarta”)
  - arkitekturen kan dokumenteras **när den har stabiliserats** (om kunden önskar sådan dokumentation, planerat som vanlig story)

B) Planering?

- Vad är en plan?

- **Varför** planerar man?
  - För att övertyga sig om man inte missat något
  - För att få en uppfattning hur mycket tid/resurser som behövs
  - För att underlätta att dela upp en uppgift på parallella aktiviteter

Hur noga planerar man?

Exempel - du skall köra bil till Alperna

1. Kör söderut efter en karta, navigera efterhand, pausa vid behov.
2. Gör upp en exakt rutt, gör en plan med varje motorvägskorsning du skall svänga i etc, varje stopp, tankning, övernattning, tider.
3. Planera i förväg varje moment i körningen, filval, placering, växlingar, hastighet, inbromsningar ....

- Lagom är bäst, det kan bli för mycket. 1:a alternativet, plus lite mer detaljer för närmsta sträckan/tiden.
- Kombinerar rimlig arbetsinsats med flexibilitet (trafikstockningar, olyckor, vägarbeten ...)

Andra exempel

Hur planerar en taxichaufför sin dag?

- en tur (iteration) i taget
- beror på kundernas krav!

Hur planerar en tågluffare sin färd?

- inte alls kanske

Vad är värdet av att planera?

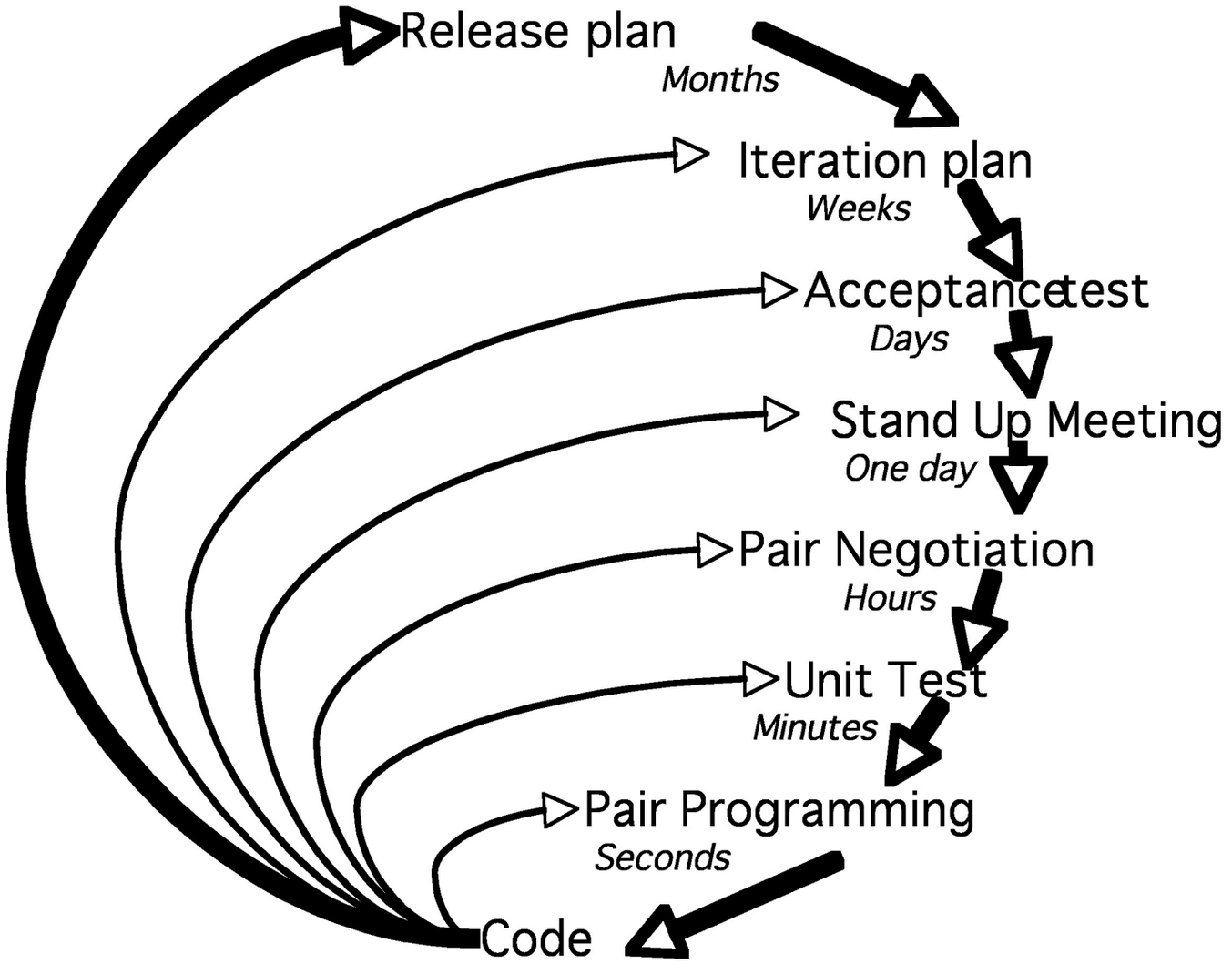
- Snarare att man **tänkt igenom** än själva planen
- Planen behöver ändras ofta
  - Ju mindre detaljer den innehåller desto mindre planeringsarbete går förlorat vid ändringar.
- Detaljer bara för nära aktiviteter, grov på långt håll
  - Något att jämföra med - larmklocka vid problem

Planering av programvaruprojekt

- Med lång tidshorisont - Releaseplaner (kund)
- Med kort tidshorisont - Iterationer (utvecklare)

- För att kunna leverera i delar - Stories (kund)
- För det dela upp arbetet - Tasks (utveckl.)
- För att driva arbetet - Testfall (par)

Planering - ökande detaljer



Vad följer man upp tidsmässigt?

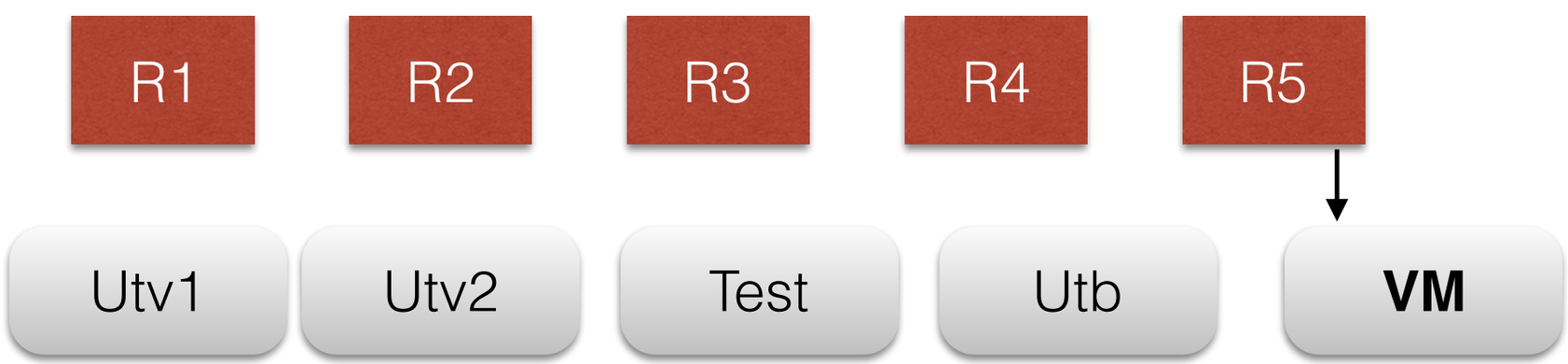
- *”Jag är klar till 50%, 90%”* - ofta meningslöst!
  - En stor uppgift blir aldrig mer än 90% klar!
- Bättre att dela upp i mindre delar och bara registrera sådana som är **helt klara**.
- För oss innebär det Tasks
  - Uppskattar tid i förväg
  - Registrera hur lång tid det tog
  - Räknar ut hur fort vi “kör”

Osäkerhetsrelation för programvarutveckling

- Av de fyra storheterna:
  - **Bemanning**, antal utvecklare
  - **Tidåtgång**, kalendertid
  - **Funktion** hos systemet
  - **Kvalitet** på funktionaliteten
- Man inte bestämma alla på en gång !
- I XP väljer vi att hålla fast Kvalitet, Tid och Bemanning och låter **Funktion variera** (”det kom inte med”)
- *Många andra metoder låter Tid variera (“Systemet ej klart i tid”).*

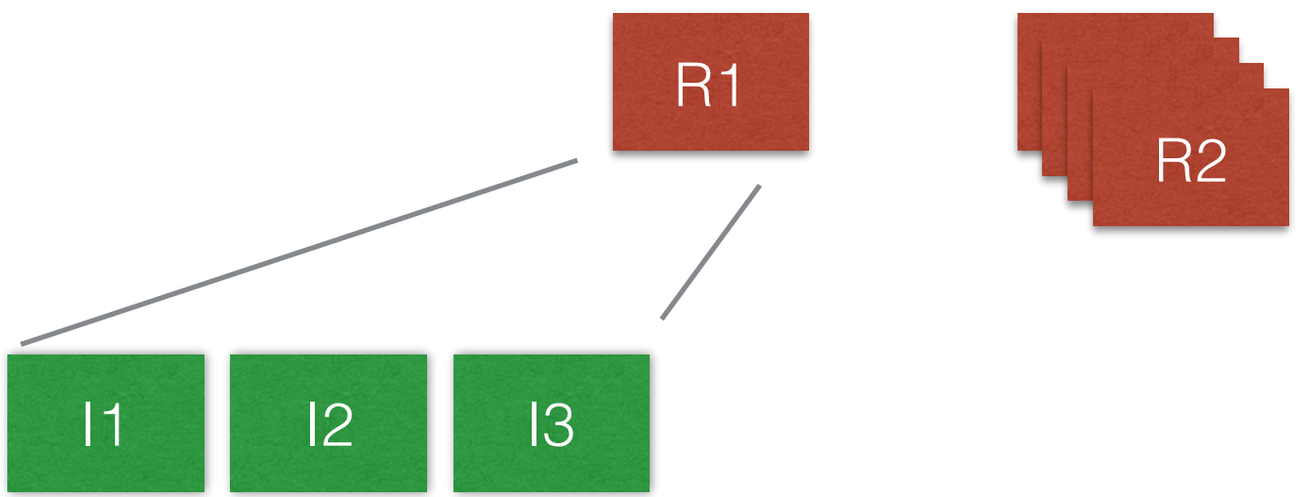
Release-plan

- När ett OS (VM etc) startar måste datorsystemen vara användbara.
  - Deadline!
- Det är kunden och deras prioriteringar som bestämmer vilken funktion som skall vara med.



Iterationer

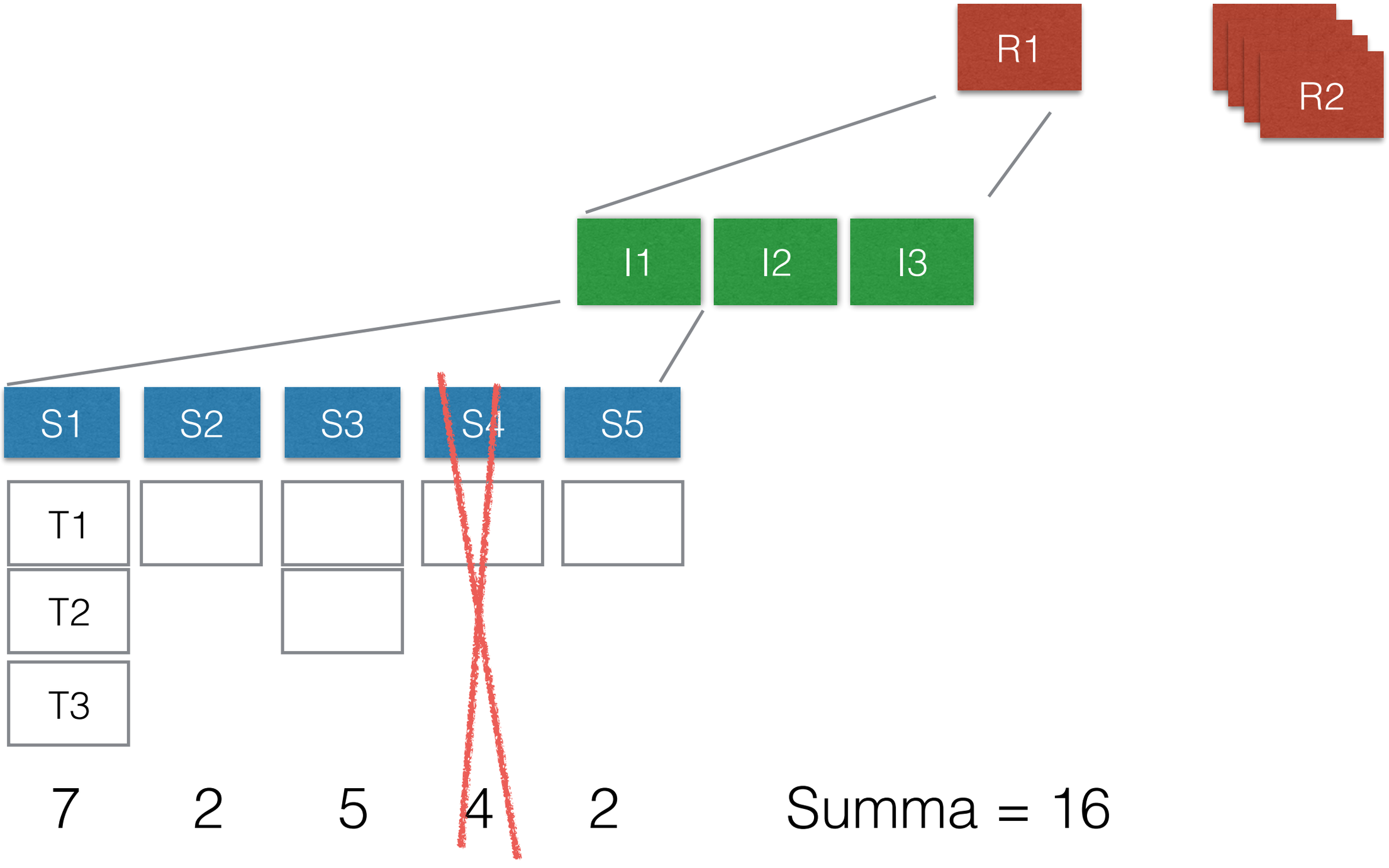
- Vi delar upp vägen till nästa (första) release i ett antal steg, iterationer.
- Precis som releaser bestäms iterationer av tid-punkter, t ex 2-3 veckor, snarare än innehåll.
- *(I projektet kommer iterationerna vara en vecka)*



Iterationsplan
Planning game

- Vi detaljplanerar bara första (nästa) iteration.
  - Kunden presenterar stories och svarar på frågor
  - Utvecklarna delar upp varje story i tasks (deluppgifter)
  - För varje deluppgift uppskattar utvecklarna hur mycket “tid” den tar och tar samtidigt på sig att göra den deluppgiften.
  - Om det blir för mycket/för lite för att fylla iterationen bestämmer kunden vad som skall bort/läggas till.
- Vi har bestämt vad som skall med i iterationen.
- Varje utvecklare vet vad han/hon skall göra under iterationen
- Sen kanske man måste ändra men det gör man med planer.

Release-Iteration-Story-Task



Tidsuppskattning

- Att tidsuppskatta implementationsarbete är svårt.
- En del arbetar med **ideal tid**.
- Andra gör **relativa bedömning** av deras svårighet (enhet, poäng gummibjörnar, hallonbåtar) och mäter i efterhand hur många enheter man hinner på given tid.
    1. Sätt ett värde på varje task
    2. “Task 39 verkar vara dubbelt så svår som Task 16 – då får den dubbelt så många hallonbåtar”
    3. Dela upp stora tasks i mindre, små tasks är lättare att skatta.
- Känns det alltför osäkert vad en task innebär gör man en “Spike”
- Ofta bra om alla tasks från en story hanteras av en utvecklare.
- Om man använder ideal tid är faktorn till riktig tid ofta 3, “optimistfaktor”.

Planering 1:a gången

- Svårt första gången - när man inte vet hur mycket man hinner - med uppföljning lär man sig efterhand. I början hjälper det med coachens erfarenhet.
- Gången som på Extreme hour:
  - Kunden definerar stories, och förslår de som skall vara med i första release
  - Utvecklarna uppskattar deras svårighet (skala 1-10)
  - Utvecklarna uppskattar hur mycket de hinner med (30)
  - Kunden väljer stories för första release
- Deadline drivet snarare än funktionsdrivet.

Tracking
Uppföljning av Iteration

- Bara hela stories räknas - om en task saknas är storyn av inget värde för kunden.
  - Viktigt att identifiera tasks som hängt upp sig.
  - Fördröjer sin Story och mindre kommer med
- “Tracker” följer upp hur långt alla task kommit jämfört med uppskattning. Om någon task blir försenad ta upp det med hela gruppen.
  - Det är inte den som sitter med Svartepetters fel
  - Han/hon bara gjorde en optimistisk uppskattning och ingen reagerade
  - Omplanera - vad skall ut?

Uppföljning av hastighet

- Efter varje iteration - för varje task färdig - summera enheter färdiga.
  - Ger första mått på “hastighet” (antal enheter / utvecklare / tid).
  - Nästa iteration har vi ett bättre mått på hur många enheter vi kan klara (“yesterday’s weather”)
  - Om ni uppskattar i ideal tid kan man räkna ut “optimistfaktorn”
- Efter några iterationer lär det stabilisera sig.

Uppföljning av release

- Målet är att få med de **viktigaste stories** (ur kundens synpunkt) i varje release.
- Bokför hur många stories som kommer med i varje release.
- När man känner hastigheten kan man uppskatta hur många stories som kan komma med i varje release. Och kunden kan prioritera.

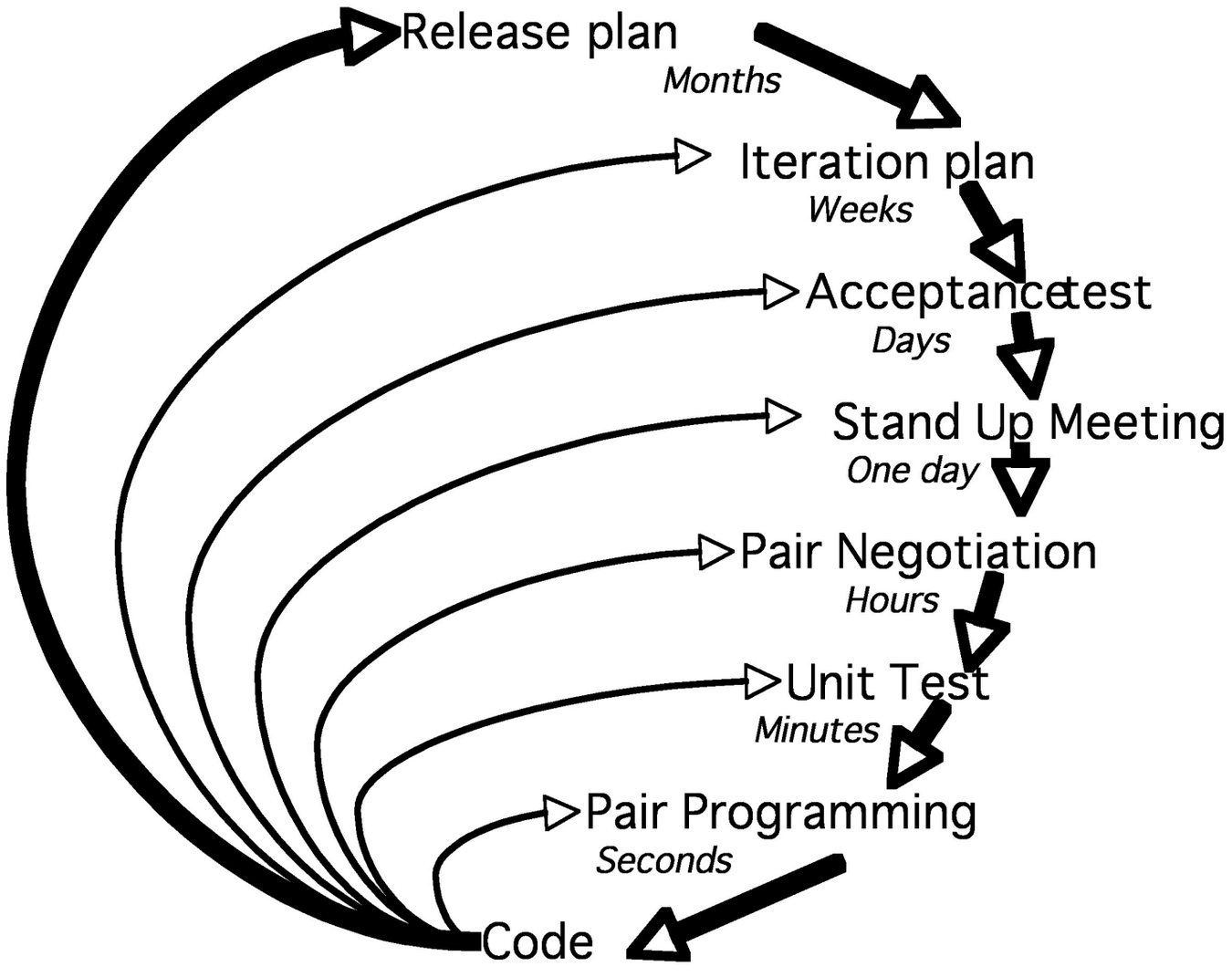
Vad göra om det blir problem ?

- Traditionellt: **systemet blir inte klart i tid**
  - Stoppa in mer folk ?
  - Projektet blir ännu mer fördröjt (de gamla måste först lära upp de nya och får inget gjort i början).
- XP: **systemet får inte med alla funktioner**
  - De väsentligaste funktionerna är med
  - Mer folk ? - kan gå med parprogrammering

Sammanfattning

- Planering för:
  - att tänka igenom svårigheterna - inte för att få en detaljerad plan
  - för att få underlag till uppföljning - upptäcka missade svårigheter - de tar längre tid än uppskattat
  - för att organisera arbetet
- Planera inte allt i detalj - bara den aktuella releasen, iterationen, storyn, tasken, testfallet
- Planera där det är motiverat - så lite som “possibly” kan funka

Feedback - hela tiden!



XP Values

- Communication
- Feedback
- Simplicity
- Courage
- All XP practices support these values

Läsanvisningar

- chromatic: Delar av Part II, p 36-44 (Business Practices)
- chromatic: Part III (XP Events) och
- chromatic: Part IV (Extreme Programming Artifacts)

- W.Wake:Where’s the Architecture?
- W. Wake: The System Mehaphor

Projektdelen LP3

- **Planeringsmöten**
  - Onsdag 10-12 ELLER 13-15
  - D2: placeras slumpmässigt ena eller andra tiden
  - Viktigt vilken tid? **email** till Ulf senast ***igår!***
- **Fördelning på team**
  - Efter tid för planeringsmöte
  - Därefter slumpmässigt
    - ”Kompisteam” undviks efter synpunkter från tidigare år.
- **Krav** för att få delta i projektdelen
  - Alla labbar fullgjorda
  - Fullgjord Kontrollskrivning

På Fredag - 18 Dec

- **Kontrollskrivning**
    - För att säkerställa att projektdeltagarna har nödvändiga förkunskaper för att ingå i ett XP-team.
    - **Gasqsalen kårhuset 14:15-15:00 Fredag 18 december**
    - Obligatoriskt!
    - Förbered dig genom att läsa igenom:
        - Kursboken enl läsanvisningar
        - OH-bilderna
        - Utdelat lab-material och artiklar
- **Introduktion** till Projektet
    - **Timman efter dvs 15.15-16:00 E:B**

Omkontrollskrivning Januari 2016