

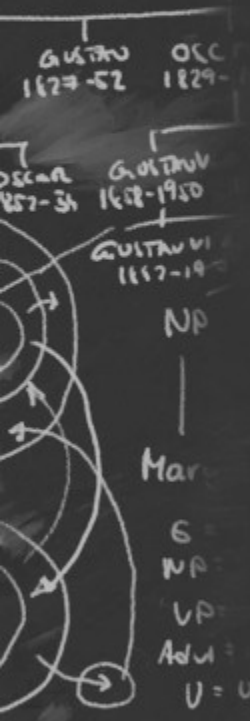
den paltbrödsmerka åkerjorden.  
 Det finns en lada grön vägg.  
 Och det finns marker som blanda



7200

HIRAGANA	KATAKANA	ma
マ	マ	mi
ミ	ミ	mu
メ	メ	me
モ	モ	mo

In Hawaii, where  
 is the same thing  
 I wonder how  
 conversation



# TDP005 Projekt: objektorienterade system



eaux qui chantent

ominativ  
 enitiv  
 ativ  
 chusativ

servus  
 servi  
 servo  
 servum

ΚΜΒΙΛ \* ΤΑΥ \* ΤΝΤ \*  
 ΡΙΥΠΔ \* ΥΝΥΥ \* ΠΝ



Hic sita sum,  
 quo frugiteo

# Idag

- Introduktion till kursen
- Introduktion till systemutveckling

# Lärare

Examinator: Eric Elfving

Kursledare: Jonas Lindgren

Övriga assistenter:

- Torbjörn Lönnemark
- Jóhann Lindell

# Uppgift

- Designa och implementera ett 2-dimensionellt arkadspel
  - Simulering av en värld
  - Figurerna med ett beteende över tid
  - Spelaren styr (minst) en av dessa figurer
  
- Dokumentera spelet/processen

# Spel, minimikrav

- › Spelet ska simulera en värld som innehåller olika typer av objekt
- › Minst tre typer av objekt
  - › Flera instanser av minst två objekttyper
- › Objekten ska röra sig över skärmen
  - › Minst spelaren och ytterligare en objekttyp ska vara rörliga
- › En figur ska styras av spelaren (åtminstone)
- › 2D-grafik
- › Kollisionshantering ska finnas
- › Spelet ska upplevas som ett sammanhängande spel
- › Även krav på implementationen, se hemsida!

# Kursmoment

- Introduktion till programutveckling (Software Engineering)
  - Systemutveckling och testning
  - Eclipse
  - Make
  - Versionshantering
- SDL (Simple DirectMedia layer) för utveckling av GUIs i C++
- UML (Unified Markup Language) för objektorienterad design
- **Projekt i C++**
  - Objektorientering
  - Doxygen

# Kursplanering 1

- › Föreläsningar
  - › Kursintroduktion och Software Engineering (den här!)
  - › SDL
  - › UML
- › Labbar
  - › Eclipse
  - › make
  - › SDL
- › Projekthandledning
  - › Vid behov (bokas med assistenterna)

# Kursplanering 2

## › Kodgranskning

- › 2-3 projektgrupper ska granska varandras kod
- › Ungefär halvvägs in i kodningen
- › Schemalagt tillfälle, men kan även göras vid annan tidpunkt
- › Kodgranskningsgrupper meddelas senare på hemsidan

## › Slutseminarium

- › Demotillfälle
- › Obligatoriskt
- › Alla visar upp sitt spel för alla andra
- › Examinator går runt och pratar med alla om spelet
- › Båda personerna i gruppen ska kunna redogöra för alla delar av spelet



# Projekt, moment

Ett projekt, som även inkluderar ett antal skriftliga rapporter:

- Kravspecifikation. Minst 4 sidor. Klar senast 2013-11-18.
- Designspecifikation. UML-klassdiagram, mm. Klar senast 2013-12-18.
- Kodgranskning.
  - Kodgranskningsmöte med 2-3 projektgrupper
  - Kodgranskningsprotokoll klar senast 2013-12-11.
- Program och projektdokumentation
  - Kod och användarhandledning klar senast 2013-12-18.
  - Demo (obligatorisk) 2013-12-18.
- Reflektionsrapport (individuell), klar senast 2013-12-20.

# Kravspec

- › Ska beskriva spelet, och vad ni ska göra
- › Ett "kontrakt" med kursledningen, det ni beskriver där är det som ska implementeras
- › Två typer av krav
  - › Absoluta krav: Ska uppfyllas för att bli godkänd på kursen
  - › Tilläggskrav: Uppfylls i mån av tid. Några av dessa ska uppfyllas för högre betyg
- › Om ni märker under projektets gång att ni inte kommer att kunna uppfylla er kravspecifikation kan den omförhandlas med kursledare. En ny version ska då lämnas in i samband med slutinlämningen.

# Betygskrav

G/VG på fyra av rapporterna:

- Kravspecifikation.
- Designspecifikation.
- Kodgranskning.
- Reflektionsrapport

3/4/5 på projektet

- Program, demo, kod och projektdokumentation

Totalbetyg:

- 3: Godkänd på samtliga moment.
- 4: 4 på projekt + minst ett VG.
- 5: 5 på projekt + VG på reflektion + ytterligare ett VG

# Projektgrupper

- 2 personer per grupp
- Ha gärna samma grupper som i TDP004
- Anmäl i webreg idag!

# Redovisning

- All kod och rapporter redovisas via GitLab:
  - <http://gitlab.ida.liu.se/>
  - Dokument i PDF-format!
  - Maila er assistent plus kursledare när ni lagt upp en redovisning.
- Använd en tydlig struktur och bra namn på dokument och kataloger, så att det är lätt att hitta och identifiera allt!

# Tidsplan

Vecka	Projekt	Annat
45	Projektidé	
46	Kravspecskrivande	Intro, Eclipse, make, SDL
47	Kravspec klar, kodning	SDL, UML
48-50	Kodning	Kodgranskning
51	Demo, dokumentation klar	

# Utvecklingsprocess

- Kravspecifikation och analys
- Systemdesign
- Objektdesign
- Implementation
- Testning

# Kravspecifikation och analys

- Förväntat beteende, inte implementation (*vad* inte *hur*)
- Process:
  1. Samla in användarkrav
  2. Analysera för att förstå och modellera systemets beteende
  3. Specificera systemets beteende i en specifikation
  4. Validera specifikationen mot kraven



# Systemdesign

- Omforma problemet till en lösning
- Konceptuell design
  - **Vad** systemet gör, dess funktion
  - Skrivet i kundens språk utan teknisk jargong
  - Implementationsoberoende
  - Kopplat till kravspecifikationen
- Teknisk design
  - **Hur** man gör det
  - Platform
  - Hierarki och funktion hos programkomponenter
  - Datastrukturer och dataflöde

# Objektdesign

- Före ~1970 fanns i princip ingen teknik utan individer gjorde som de ville
- 1975-1985 Strukturerad programmering (klassiska paradigmet)
  - Strukturerad systemanalys
  - Dataflödesanalys
  - Strukturerad programmering och testning
- Problem: tekniken skalade inte upp.
  - Problematiskt för stora program > 500 000 rader eller mer
  - Svårt att underhålla programmen

# Objektdesign

- Objektorienterad design
- Problemet och lösningen organiseras som en samling av diskreta objekt, inkluderande både datastruktur och beteende
- Karaktäristika
  - Identitet hos enskilda objekt
  - Abstraktion för att förstå problemet
  - Klassificering av grupper av objekt med gemensamma attribut och beteende
  - Inkapsling för att dölja implementationsdetaljer
  - Arv som tillåter att attribut delas mellan snarlika objekt
  - Polymorfism så att relaterade objekt kan ha olika beteende
  - Persistens så att objektets namn, tillstånd och beteende består över tid och rum

# Implementation

- Standarder
- Kodningsriktlinjer
  - Stilguider för ökad läsbarhet och förståelse
- Dokumentation
  - Intern dokumentation
    - Header comments
    - Kodkommentarer
    - Variabelnamn
    - Formattering
    - Datadokumentation
  - Extern dokumentation
    - Översikt av systemets komponenter
      - Ingår i systemdokumentationen

# Testning

- Enhetstestning
  - Fokuserar på de enskilda modulerna
- Integrationstestning
  - Testar när moduler sätts samman till mer komplexa enheter
- Användbarhetstestning
  - Gränssnittstestning
- Systemtestning
  - Funktion
  - Prestanda
  - Acceptans

# Objektorientering, historia

- › Simula 67, 60-tal, objekt som formella koncept
- › Smalltalk, 70-tal
  - › Introducerade begreppet objektorientering
  - › Objekt och meddelanden
  - › Dynamiskt
  - › Spreds i stor skala 1981
- › Objektorienterad LISP, 80-tal
- › C++, tillägg av objekt till C, 1980
- › Dominant programmeringsparadigm på 90-talet
  - › t.ex. Delphi, Java, 1995
- › 2000-talet??

# Objektorienterad design, karakteristik

- › Objekt är en abstrakt representation av världen
- › Objekt har ansvar för sitt eget tillstånd
- › Objekt är oberoende enheter
- › Systemfunktionalitet uttrycks m.h.a. operationer som associeras med olika objekt
- › Delad data undviks
- › Objekt kommunicerar genom att anropa operationer hos andra objekt
- › Objekt kan vara distribuerade, kan exekveras antingen sekventiellt eller parallellt

# Objektorienterad utveckling

- › Objektorienterad analys
  - › Designkrav
  - › Högnivåarkitektur
- › Objektorienterad design
  - › Översätter en arkitektur till programmeringskonstruktioner
  - › Modellerar klasser, metoder, etc
- › Objektorienterad programmering
  - › Implementerar designen
- › Gränsen mellan OOA och OOD är inte skarp



# Objektorientering, språk

- › Rena objektorienterade språk
  - › Smalltalk
  - › Scala
- › Språk främst utvecklade som objektorienterade
  - › C++
  - › Java, C#
  - › Python
- › I grunden procedurella språk där OO lagts till
  - › Visual Basic
  - › Ada 95
- › Andra språk med många OO-drag
  - › Common LISP
  - › Oberon

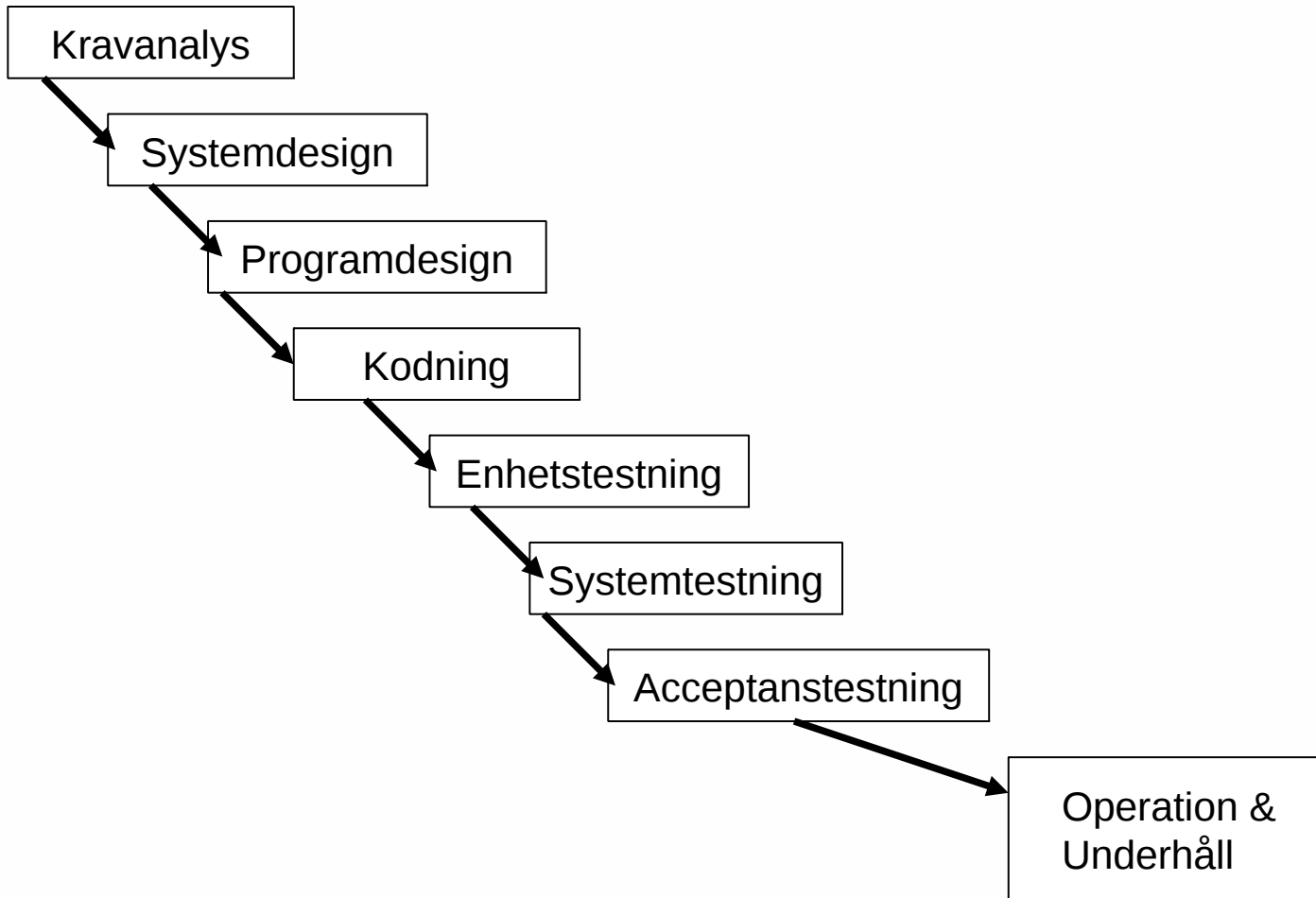
# Utvecklingsmetoder

- Vattenfallsmodellen
- V-modellen
- Prototyping
- Spiralmodellen
- Agil utveckling

# Varför behövs en utvecklingsmetod?

- Komplexitet och styrning
  - Att få alla i ett team att jobba åt samma håll
- Arbetsfördelning
- Kommunikation
  - Mellan utvecklare (under konstruktion)
  - Mot kund (innan, under och efter)
  - Till eftervärlden (testamentering)
  - Till omvärlden (manualer)

# Vattenfallsmetoden



# Vattenfallsmetoden

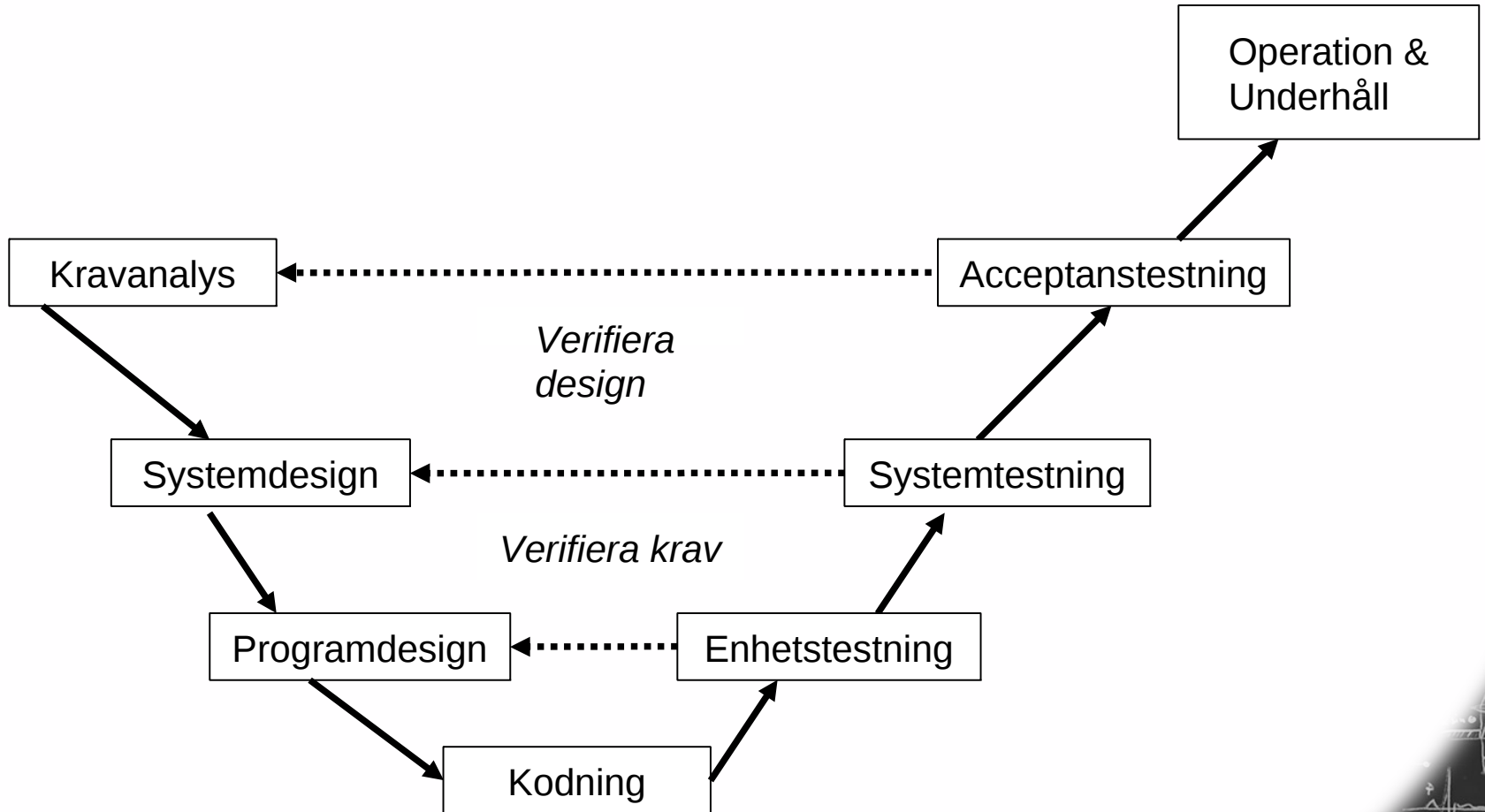
## Fördelar

- Enkel och lätt att förstå
- Passar in i projektstyrningspraktiker
- Fokusera på krav i början, billigt i slutet
- Bra för små projekt
- Bra för stabila projekt
- Fokus på dokument – mycket kunskap kan bevaras
- Används mycket
- Bra för fastpriskontrakt

## Nackdelar

- Krav kan förändras – dumt att låsa sig
- Tidig låsning vid lösningar – dyrt att göra om ifall man ändrar sig sent i projektet
- Feedback från senare faser skulle kunnat underlätta en del andra faser
- Svårt att göra tidsuppskattningar
- Ingen riskhantering
- Litet utrymme för problemlösning

# V-modellen



# Agil utveckling

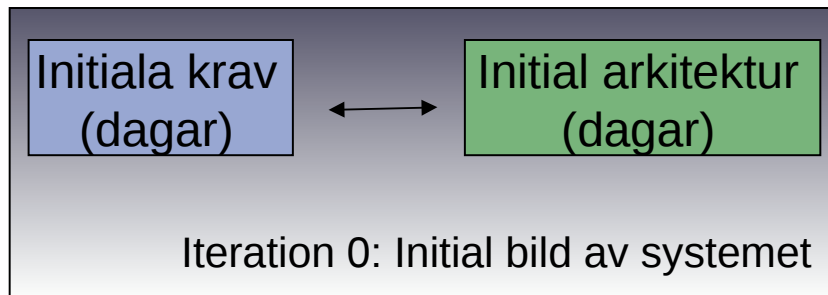
- Individer och interaktion istället för processer och verktyg
  - Ansikte-mot-ansikte-kommunikation istället för dokument
  - Lita på att utvecklarna organiserar sig och sköter sig
- Producera kod istället för dokument
  - Framgång mäts i hur väl programmet fungerar
- Samarbeta med kunden istället för kontrakt
- Förändring istället för planering
  - Inser att det inte går att förutse alla krav i början av projektet

# Agil utveckling, värden

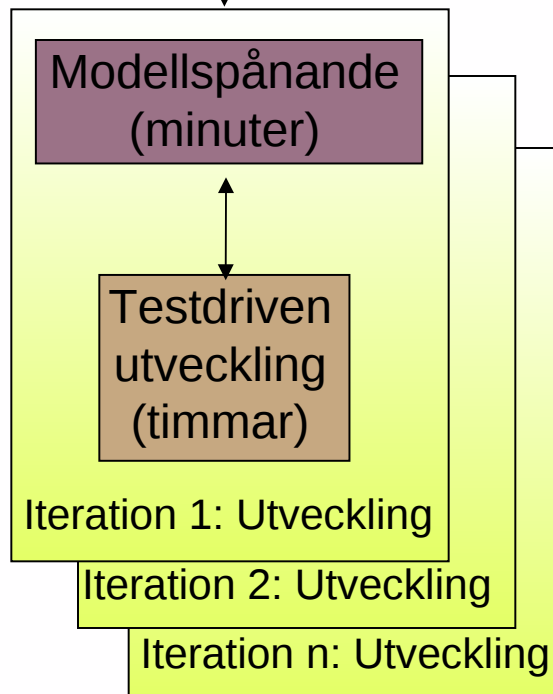
- Kommunikation
  - Alla inblandade i processen måste kommunicera. Kund, medarbetare, ägare.
- Enkelhet
  - Utveckla den enklast tänkbara lösningen som uppfyller alla dina behov.
- Återkoppling
  - “Optimism is an occupational hazard of programming, feedback is the treatment.” (Kent Beck) Begär och ge återkoppling hela tiden.
- Mod
  - Våga besluta dig och stå för dina åsikter.
- Ödmjukhet
  - Erkänn att du inte vet allt och att andra kan tillföra värdefulla synpunkter till ditt projekt.



# Agil utveckling



- Grov programvision
- Arkitekturvision



- Kraven utvecklas under tiden
- Modellera sparsamt
- Involvera användarna
- Var specifik

- Utveckla körbara program

# Agila metoder

- eXtreme programming (XP)
- Scrum
- Crystal
  - Varje projekt behöver sina egna direktiv, konventioner och metoder
  - Projektets kvalitet beror av människorna i projektet
  - Högre produktivitet med bättre kommunikation och täta leveranser
- Adaptive Software development
  - Projekt organiserat kring att bygga komponenter som tillhandahåller olika egenskaper hos programmet
  - Fixa leveranstider

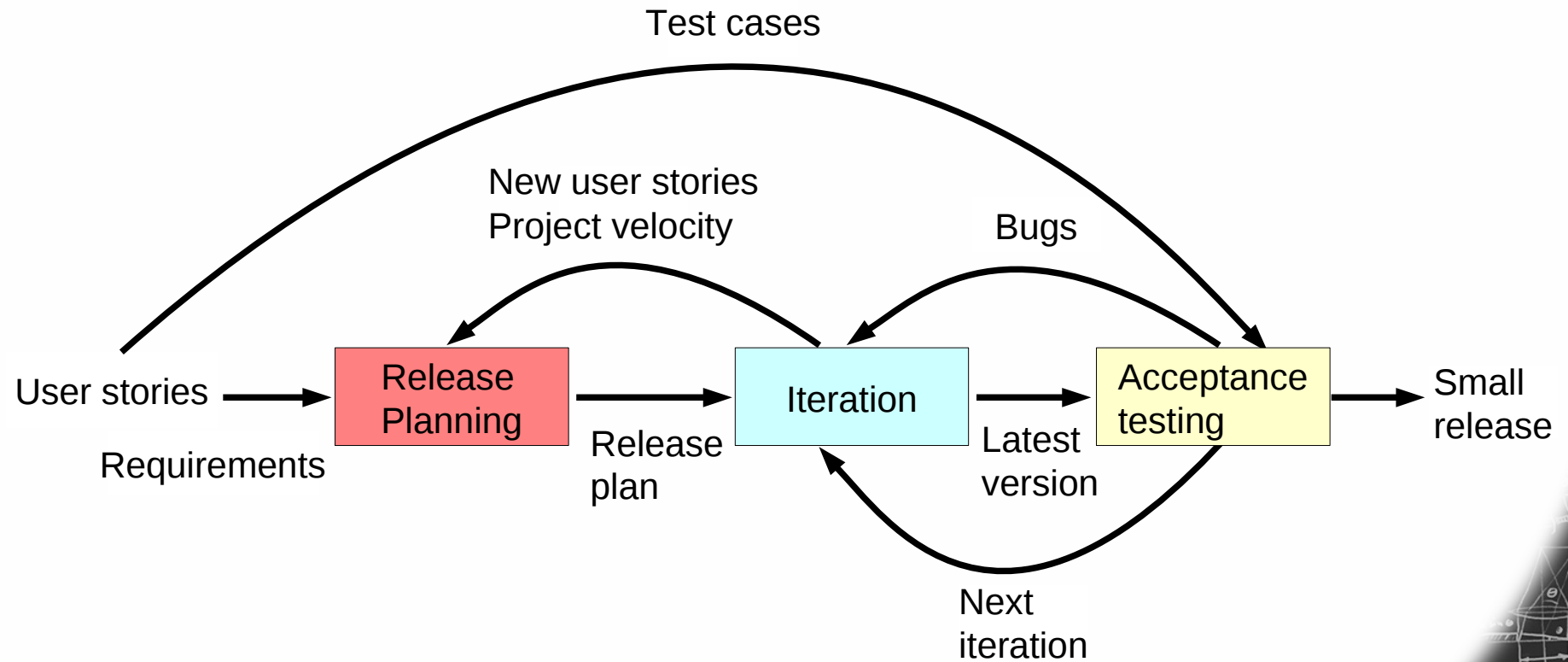
# Extreme programming (Kent Beck)

- Extrem tonvikt på programmering  
...och testning
- Extrem valfrihet/flexibilitet
  - att kunna lägga ned (men ändå få något ut av det)
  - att kunna vända kurs
  - att kunna avvakta
  - att kunna växa snabbt
- Extremt korta iterationscykler

# XP, 12 aspekter

- Planering med kunden och fokus på kravens vikt
- Små inkrementella leveranser
- Metafor som gemensam vision av hur systemet skall fungera
- Enkel design som bara hanterar nuvarande behov
- Skriv testerna först
- Refactoring: omstrukturera krav, design och kod när det blir komplext och oöverskådligt
- Parprogrammering
- Gemensamt ägande, alla kan ändra i all kod
- Kontinuerlig integration och små förbättringar
- 40-timmarsvecka
- Kunden på plats
- Kodstandard som alla följer

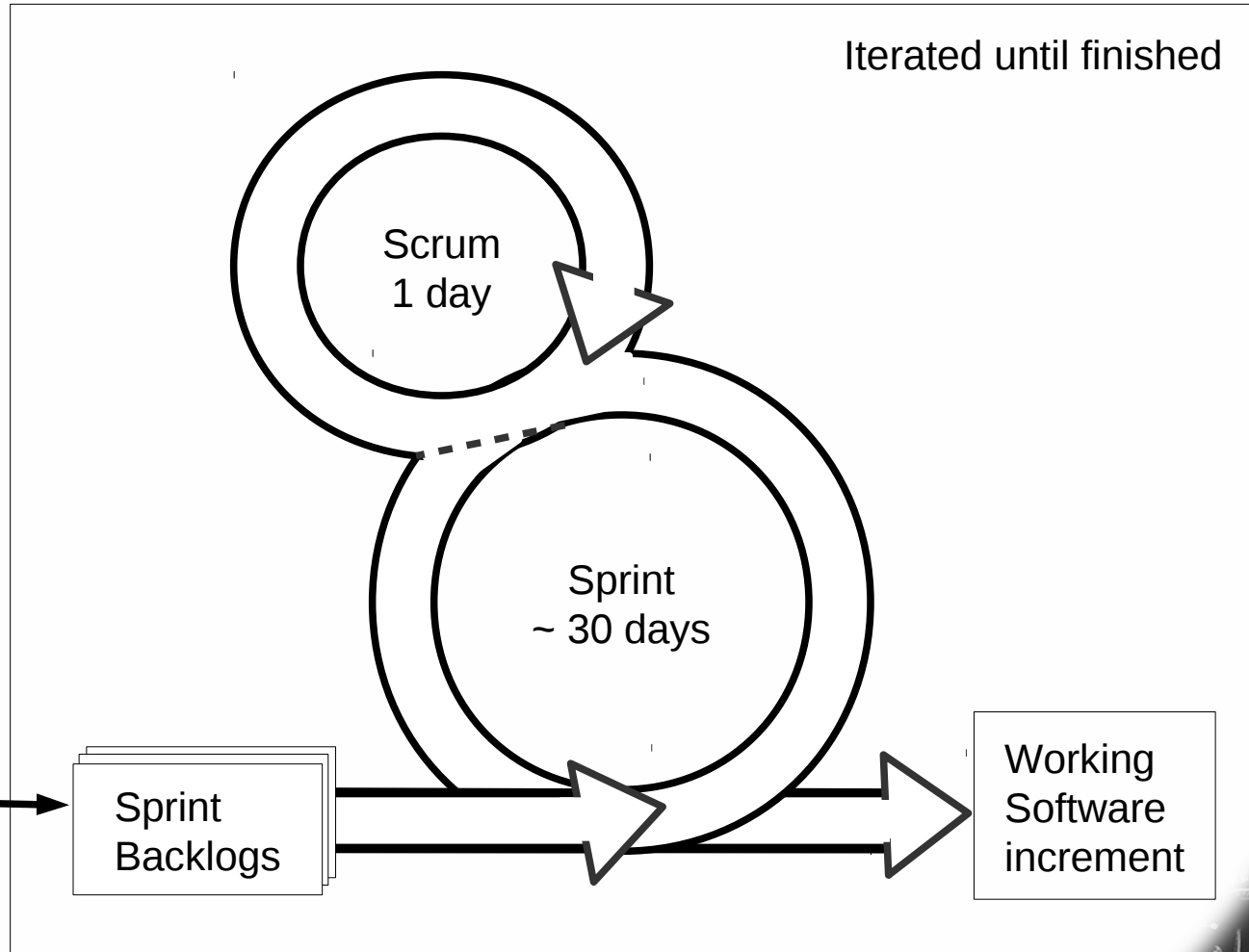
# XP



# Scrum

- Fokus både på management och systemutveckling
- Sprint
  - Iterationer om runt 7-30 dagar
  - Varje sprint ska producera och testa en körbar programversion
  - Mål prioriteras och väljs från projekt-backloggen, delvis baserat på erfarenheter från tidigare sprints
  - Planeringsmöte innan varje sprint
- Scrum
  - 1-dagarsiteration
  - Görs parallellt av många (självorganiserande) team
  - Scrum-möte varje morgon, c:a 15 min, stående

# Scrum



Planning  
High level design

# Designnivåer som struktur för en kravspecifikation, 1

- Vision
  - Vad är den bärande tanken bakom systemet?
- Mål
  - Vad är det mer konkreta målet med systemet?
- Målgrupp
  - Vilka ska använda systemet?
- Tjänster
  - Vad ska man kunna göra med systemet?
  - Vad erbjuder systemet för tjänster?
- Användbarhetsmål
  - Hur ska tjänsterna upplevas?



# Designnivåer som struktur för en kravspecifikation, 2

- Funktioner och innehåll
  - Går det att beskriva mer konkret vilka funktioner och vilket innehåll som ska finnas i systemet?
  - Interaktionsstruktur: Hur ser användargränssnittets struktur ut?
  - Interaktionstekniker: Hur interagerar man?
  - Presentationsstekniker: Vad är det man ser på bildytan?
  - Fysisk form: Vad har produkten för form? Är det ett specialdesignat föremål? Är det en programvara på en persondator?
- Säkerhetskrav.
- Eventuellt hårdvarukrav och prestandakrav.

# Andra krav

- ISO-standarder med mera som ska vara uppfyllda.
- Tidplan och leveransdatum
- Ekonomiska ramar.
- Ofta används kravspecifikationen som ett kontrakt för vad som ska levereras av utvecklaren.

# SMART kravspecification

- S**pecific: Tydlig och rakt på sak. Svara på frågorna:  
Vad ska du göra? Varför är det viktigt?
- M**easurable: Om du inte kan mäta målen, hur kan du då  
veta ifall du är uppfyllda eller inte?
- A**greed upon: Överenskommelser mellan alla inblandade  
(Kund, användare, et.c.)
- R**ealistic: Möjligt med de tillgängliga resurserna,  
kunskapen och tiden. Du måste vilja och  
kunna utföra det.
- T**imely: En tydlig tidsram för projektet.

# Design och testning

- Designen måste ta hänsyn till testningen
  - Bra design uppmuntrar till testning av individuella moduler snarare än hela systemet
- Enkla designer är ofta enkla att testa
  - Tydliga abstraktionsbarriärer
  - Väldefinierad modulfunktionalitet
- Objektorienterad design
  - Design som respekterar abstraktionsbarriärer
  - Moduler behandlas som enheter som tillhandahåller en väldefinierad mängd funktioner, inte som moduler som är implementerade på ett visst sätt

# Testning

- Enhetstestning
  - Testar enskilda metoder, klasser, etc
- Komponenttestning
  - Testar hela klasser eller små moduler
- Integrationstestning
  - Testar kombinationer av flera klasser eller subsystem
- Regressionstestning
  - Repetition av tidigare tester, för att se om de fortfarande fungerar när ny funktionalitet lagts till
- Systemtestning
  - Testar det fullständiga systemet
- Acceptanstestning
  - Testar om kraven uppfyllts i programmet

# Enhetstestning

- Att vara säker på att varje modul fungerar korrekt innan den integreras med andra moduler
- Mycket lättare att vara säker på att en liten del av systemet fungerar rätt än att avlusa hela systemet på en gång
- Bra för att hitta udda fel som bara uppträder för specifika indata

# Automatisera testning

- "Trist att testa"
- Skapa en testsvit som ständigt växer och som automatiskt körs varje gång ett program byggs och som bara genererar pass/fail
- Förhindrar att projektmedlemmer checkar in kod som inte fungerar med resten
- Det finns många alternativ för Unit-testning för C++
  - CUTE rekommenderas under kursen
  - <http://cute-test.com/>

# Skapa testfall

- Black-box
  - Tittar bara på specifikationen
  - Försök glömma koden
  - Be någon annan skriva testfallen
- White-box (Glass-box)
  - Tar hänsyn till koden
  - Skriv testfall som knäcker koden
  - Skrivs av programmeraren eller någon som kan koden



# Black-box-testfall

- Ekvivalenstestning
  - Dela upp indata i ekvivalensklasser och testa en korrekt och en felaktig från varje klass
    - Täckning: Varje indata hör till en ekvivalensklass
    - Disjunkta: Inget indata hör till mer än en klass
    - Representativitet: Alla indata i en klass ger samma fel
- Gränsfall
  - Testa gränserna för varje ekvivalensklass
- Orsak-effekt
  - Utgå från förväntat resultat för att bestämma indata
  - Antar att en kombination av värden orsakar fel
- Felgissning
  - Välj testfall utifrån domänkunskap

# Exempel

`getNumDaysInMonth(month, year)`

## ➤ Ekvivalenstestning

- ger 6 ekvivalensklasser:
- (31 dagar, 30 dagar, februari) x (skottår, icke-skottår)

```
assertEqual( (getNumDaysInMonth(1, 1200), 31)
```

```
assertEqual( (getNumDaysInMonth(7, 1300), 31)
```

```
assertEqual( (getNumDaysInMonth(4, 1996), 30)
```

```
assertEqual( (getNumDaysInMonth(9, 2001), 30)
```

```
assertEqual( (getNumDaysInMonth(2, 2000), 29)
```

```
assertEqual( (getNumDaysInMonth(2, 2100), 28)
```

## ➤ Gränsfall

```
getNumDaysInMonth(0, 1974)
```

```
getNumDaysInMonth(4, -1)
```

# White-box testing

- Varje sats i koden exekveras minst en gång
- Olika typer av satser:
  - Enskilda satser
  - Villkorssatser: varje utfall exekveras en gång
  - Loopar:
    - Hoppa över loopen
    - Exakt en gång i loopen
    - Mer än en gång
  - Varje väg igenom koden exekveras minst en gång

# Testdriven utveckling

- Skriv testerna först
- Undviker fel på grund av feltolkad kravspec
  - Klargör kraven på systemet
- Avslöjar brister i kravspecifikationen
  - Bättre att upptäcka när man skriver tester istället för kod
- Underlättar testning av koden när den utvecklas
  - Säkerställer att man inte testas mot koden (black-boxtestning)

# Kursens början!

- Projekt
  - Forma projektgrupper + anmäl i webreg (idag!)
  - Bestäm vad ni vill göra (idag!)
  - Börja skriva kravspecifikation (deadline 2013-11-18)
- Labbar
  - Onsdag 2013-11-13, Eclipse
  - Torsdag 2013-11-14, make
  - Material + uppgifter finns på kurshemsidan
- Föreläsning 2
  - Fredag 2013-11-15
  - SDL (grafikbibliotek)