



LUNDS  
UNIVERSITET

# Föreläsning 2: Test I, git, Design I

Programvaruutveckling - Metodik 2019 | Markus Borg



# Agenda F2

---

## Allmänt

- Formalia: Kursmål, kursombud
- Projektet
- Hemtentan

## Test Del 1

- Verifiering och validering
- Whitebox-testning
- Introduktion till JUnit

## Källkodshantering

- Introduktion till git

## Programvarudesign Del 1

- Objektorienterad design
- Refaktorisering



# ETSA02 Formella kursmål

---

## Kunskap och förståelse

- kunna definiera **grundläggande begrepp** inom utveckling av stora programvarusystem
- kunna beskriva de **vanligaste processerna** för utveckling av stora programvarusystem
- kunna förklara de viktigaste momenten i **kravhanteringsprocessen**
- kunna förklara hur **testning** går till
- kunna beskriva vad en **arkitekturdesign** är
- kunna beskriva de viktigaste stegen i **projektplanering och projektuppföljning**
- kunna beskriva hur organisationer planerar och genomför en **serie av projekt**

## Färdighet och förmåga

- kunna **utveckla projektplan, kravspecifikation och testplan** för ett mindre projekt
- kunna **granska** projektplan, kravspecifikation och testplan för ett mindre projekt.
- kunna skriftligen **formulera text i projektdokumentation**

## Värderingsförmåga och förhållningssätt

- förstå **komplexiteten** i uppgiften att utveckla ett programvarusystem.
- ha förståelse för **ingenjörrens yrkesroll**

# Kursombud

---

2 X C:

TBD

TBD

2 X I:

TBD

TBD



# Tidrapportering

- Tidrapportering per roll för tre utvecklingssprints
  - Projektledaren rapporterar veckovis för hela gruppen
  - Ange hela timmar i vita celler i **ETSA02 Time Reporting 2019**
  - Sammanställning presenteras på varje föreläsning

Tidrapportering Grupp 05								
Nedan följer ett exempel:	(Uppskattat arbete i timmar)	Vecka 14	Vecka 15	Vecka 16	Vecka 17	Vecka 18	Vecka 19	Summa per roll
<b>Project manager</b>								0
<b>Requirements engineer</b>								0
<b>Software architect</b>								0
<b>Quality engineer</b>								0
<b>Configuration manager</b>								0
<b>Sales engineer</b>								0
<b>Summa per vecka</b>		0	0	0	0	0	0	0
							0	Totalt projektarbetet
							480	Projektbudget
							480	Differens



LUND  
UNIVERSITET

# L2: Video pitch deadline fredag 23:59

---

- Tid att arbeta med detta på övning 2
- Flera grupper vill börja redan nu
- Förväntningar på videon
  - Meddela tydligt om ni erbjuder en ledare, en droid eller en vanlig robot
  - Maxlängd 2 min
  - Spara videon i ett standardiserat videoformat som kan spelas upp utan särskilda krav på programvara
  - Självklart inget olagligt eller stötande
- Möjligheter
  - Spela in med mobiltelefon
  - Screen capture
  - Animerad Powerpoint
  - Ritplatta, skådespelare, actionkamera, drönarfotografering, ...
  - Blanda tekniker



**LUNDS  
UNIVERSITET**

# Leverans på Google Drive

---

- Alla grupper har fått en projektkatalog på Google Drive
- Leverera marknadsföringsmaterial i “Display window”
  - Ladda upp video-pitchen
  - Ni får även ladda upp annat promotion-material (OBS! valfritt)
    - » Broschyror
    - » Skärmdumpar
    - » Dokument
    - » Försrag på laguppställningar för LU Rumble
    - » Osv.
- Den reguljära enheten kommer att skapa en delad katalog för robotmässan inför F3





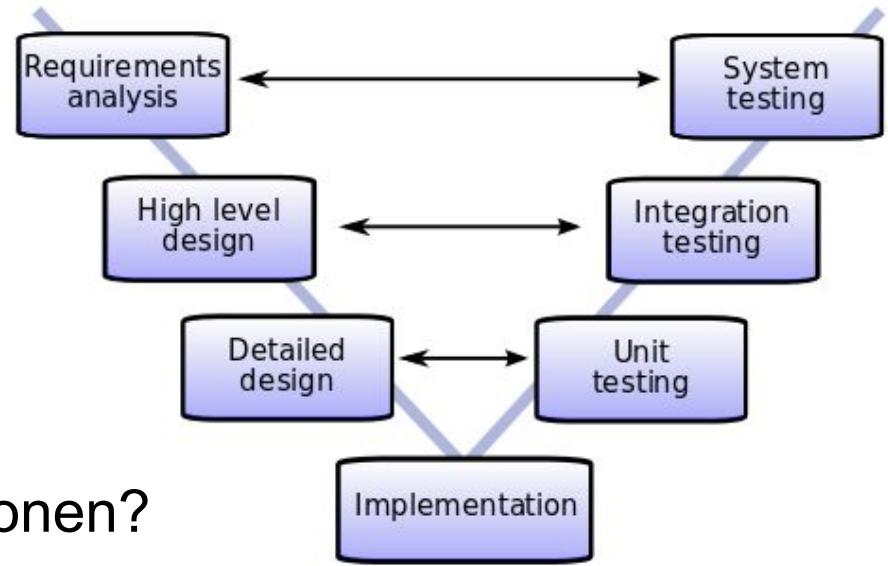
LUNDS  
UNIVERSITET

# Test del 1

Programvaruutveckling - Metodik | Markus Borg



# Verifiering & Validering



## Verifiering

Bygger vi produkten rätt?  
Uppfyller vi kravspecifikationen?

## Validering

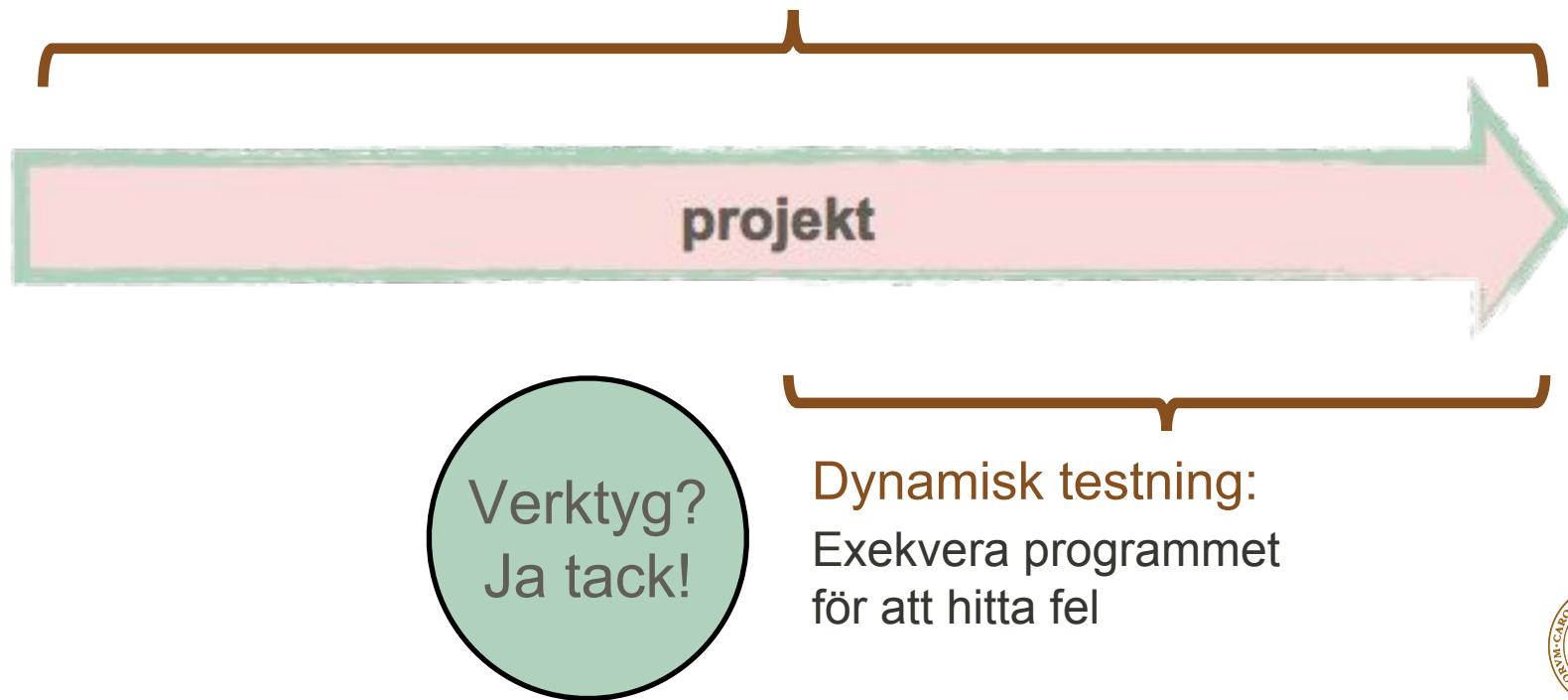
Bygger vi rätt produkt?  
Kommer beställaren att bli nöjd?  
Når beställaren sina affärsmål?



LUNDS  
UNIVERSITET

# Dynamisk och statisk testning

Statisk testning (Granskning):  
Hitta fel utan att exekvera programmet

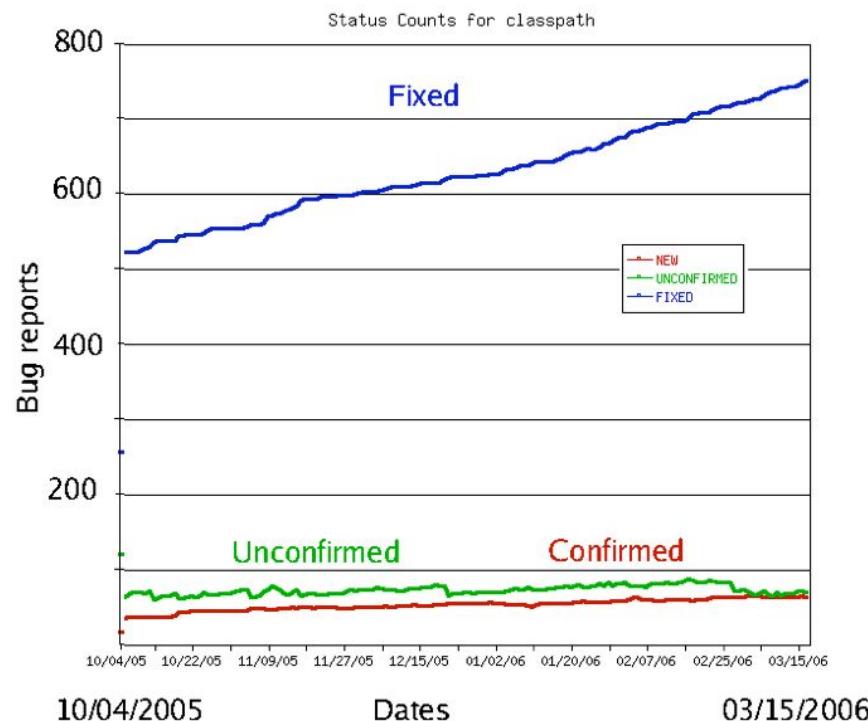


LUNDS  
UNIVERSITET

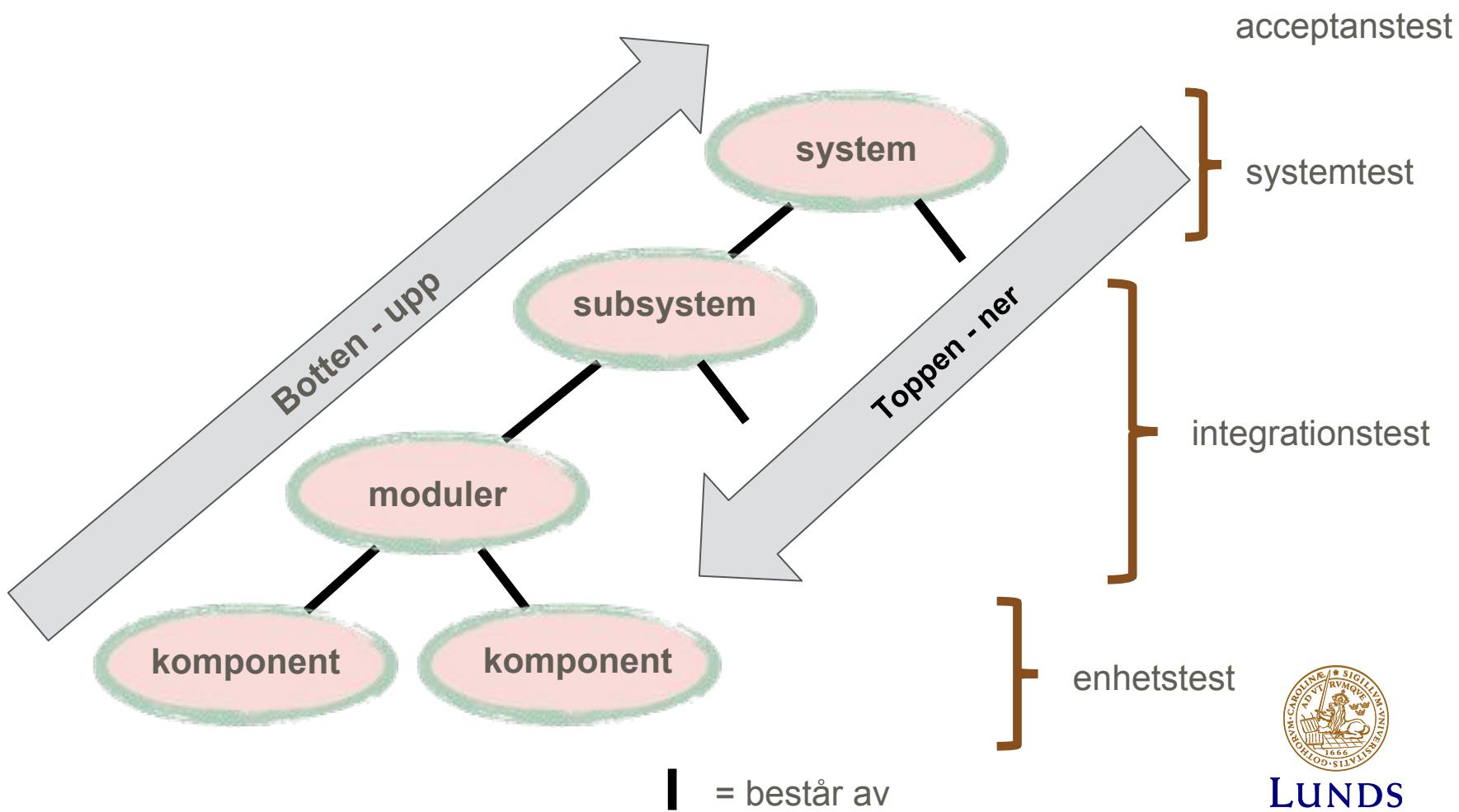
# Varför testar man?

---

- Hitta fel för att man vill förbättra produkten
- Visa att produkten är bra
- Verifiera att man uppfyller standard
- ...
- ... men oavsett vilket så kan man aldrig visa att det inte finns fel, bara att det finns fel...



# Testprocessen - flera delprocesser



LUNDS  
UNIVERSITET

# Testplanering - steg ett i processen

## Indata

- Projektplan
- Kravspec
- ...
- Design
- Manual
- Prototyper
- ...

## Testplanering

## Testplan

- Testprocess
- Testobjekt
- Testmiljö
- Kravspårning
- Tidplan
- Dokumentation av resultat
- Begränsningar
- Testfall



LUNDS  
UNIVERSITET

# Acceptanstestning

---

Det sista steget i programvarutestning.

Görs när: Innan sluttrelease på marknad eller till beställare.

Av vem: Beställaren eller tilltänkta användare.

Mål: Utvärderar om systemet uppfyller förväntningarna i den riktiga miljön.



I projekten kommer dock acceptans-testning göras efter gruppernas sluttrelease



LUNDS  
UNIVERSITET

# Systemtestning

---

Test av hela systemet. Domineras ofta av manuell testning.

Görs när: Inför releaser, milstolpar, certifiering - eventuellt under utveckling

Av vem: Testare. Eventuellt extern oberoende testorganisation.

Mål: Utvärderar om systemet uppfyller kravspecifikationen.



Air Force photo by Ron Mullan (2008)



LUNDS  
UNIVERSITET

# Systemtestestfall, två exempel från automat

---

**Test case:** Buy cola

**Pre-condition:** Machine in start state.

**Post-condition:** Machine in start state.  
A cola can received.

1. Press the cola selection button.
2. Insert a SEK 10 coin.
3. Insert a SEK 5 coin.
4. Receive a cola can.



**Test case:** Press more than one button

**Pre-condition:** Machine in start state.

**Post-condition:** Machine in start state.  
A water can is received.

1. Press the water selection button.
2. Press the beer selection button.
3. Insert a SEK 5 coin.
4. Receive a water can.



**LUNDS  
UNIVERSITET**

# Spårbarhet mellan krav och testfall

---

Alla krav ska testas av minst ett testfall

Alla testfall ska testa minst ett krav

Spårbarhetsmatris

	Test case												
Req	1	2	3	4	5	6	7	8	9	10	11	12	13
1													
2													
3													
4													
5													
6													
7													
8													
9													
10													
11													
12													
13													



LUNDS  
UNIVERSITET

# Testprotokoll

# Redogör för resultatet av test

- t.ex. en tabell med testfall, testresultat datum, testare, ...

9/9

0800 Anton started  
1000 " stopped - anton ✓ { 1.2700  
~~13' WC (032) MP - MC~~ ~~1.130476415~~  
 (033) PRO 2 2.130476415  
 const 2.130476415  
 Relays 6-2 in 033 failed special  
 in relay " 10.00 +  
 Relays changed  
 1100 Started Cosine Tape (Sine check)  
 1525 Started Multi Adder Test.

1545  Relay #70 (Moth) in relay

~~1600~~ First actual case of bug being  
 Anton started.  
 1700 closed down



LUNDS  
UNIVERSITET

# Integrationstestning

---

Testa att integrera komponenter.

Görs när: Så ofta som möjligt, gärna automatiskt.

Av vem: Utvecklare eller testare.

Mål: Säkerställa att komponenterna fungerar ihop.



@randal\_olson (2016)



LUNDS  
UNIVERSITET

# Enhetstestning

---

Automatisk tester av systemets minsta ingående delar. Oftast enskilda klasser.

Görs när: Kontinuerligt. Helst varje gång något ändras i källkoden.

Av vem: Testfall skapas av utvecklare och exekveras automatiskt.

Mål: Testa all produktionskod.



LUNDS  
UNIVERSITET

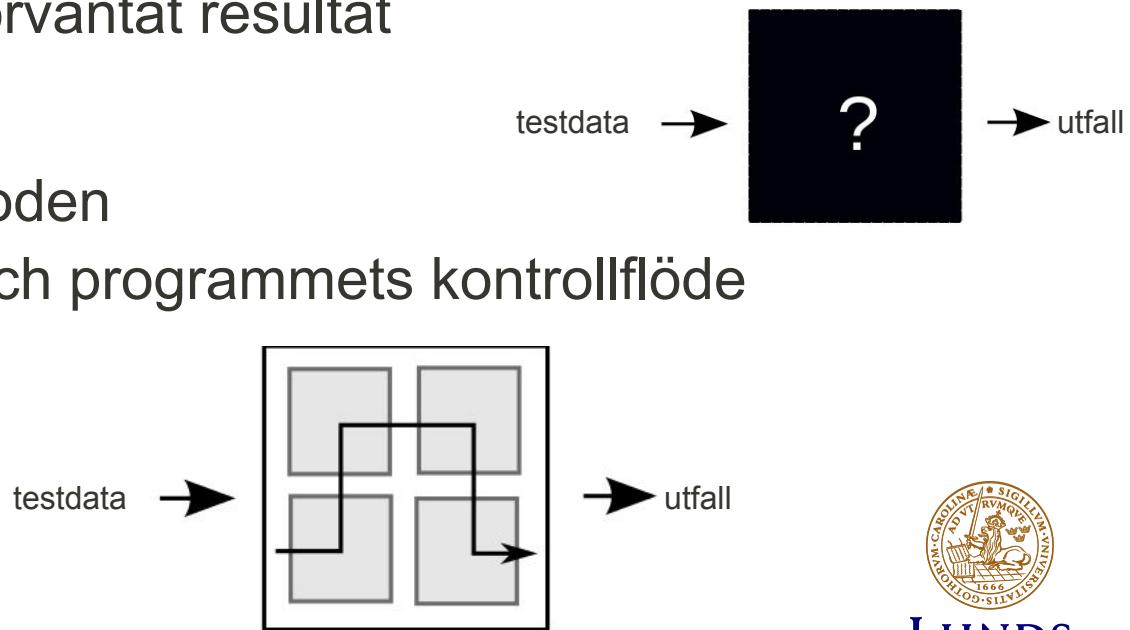
# Testtekniker: Black-box vs. White-box

## Black-box

- Programmet ses som en "svart låda". Kunskap om koden används ej när testfall skapas.
- Kravspecifikationen används för att ta fram testfall
- Jämför utfall med förväntat resultat

## White-box

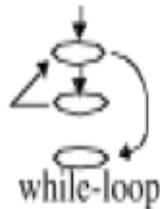
- Kräver tillgång till koden
- Testar både utfall och programmets kontrollflöde



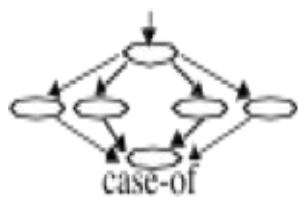
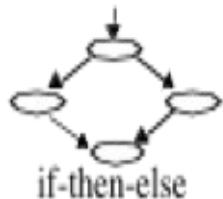
LUND  
UNIVERSITET

# White-box testning

---



- Tanken är att testfallen ska täcka all kod
  - Men vad menas med "all kod"?
- Betrakta kodens kontrollflödesgraf

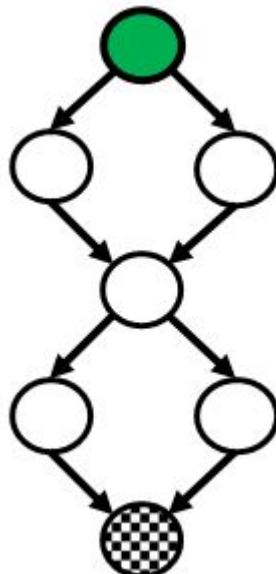


LUND  
UNIVERSITY

# Täcka rader (statement coverage)

---

- Exekvera alla rader minst en gång
  - Alla noder i kontrollflödesgrafen

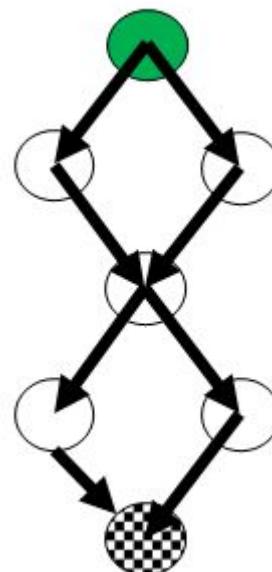


LUND  
UNIVERSITY

# Täcka grenar (branch coverage)

---

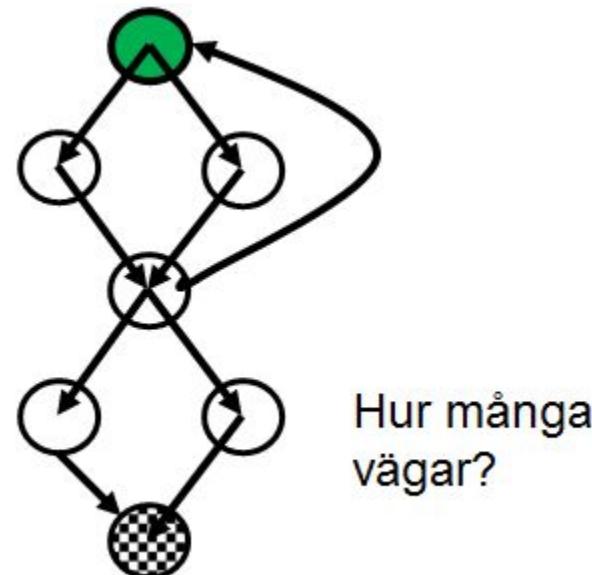
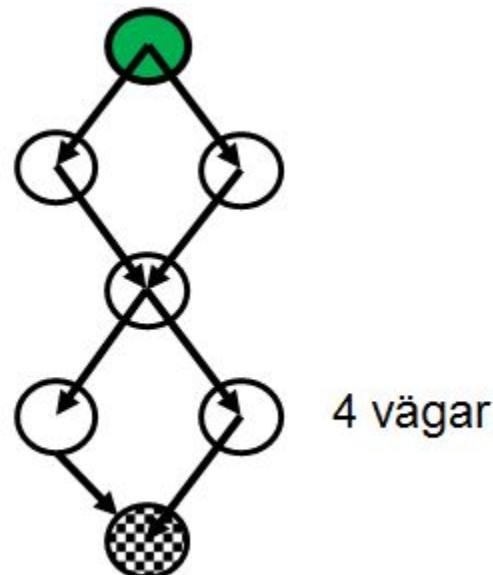
- Exekvera alla grenar minst en gång
  - Alla bågar i kontrollflödesgrafen
  - Innefattar även komplett radtäckning



LUND  
UNIVERSITY

# Täcka vägar (path coverage)

- Exekvera samtliga vägar från startnod till slutnod
  - Innefattar även komplett grentäckning
  - Antalet testfall exploderar med loopar!

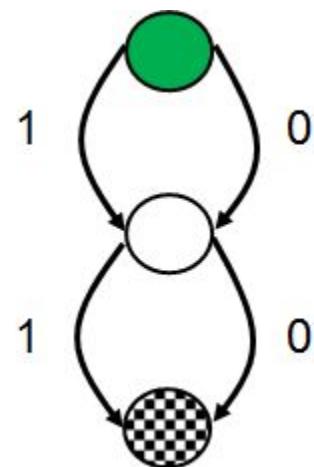


LUND  
UNIVERSITY

# Täcka enkla vägar (simple path coverage)

- Exekvera samtliga linjärt oberoende vägar från startnod till slutnod

- Innefattar komplett grentäckning
- Hanterar problematiken med loopar
- Ofta en rimlig kompromiss



Vägar:  
 $(1,1)$   
 $(0,0)$   
 $(1,0)$   
 $(0,1)$

Ej linjärt  
oberoende!



LUND  
UNIVERSITY

# McCabe's Cyclomatic Complexity (CC)

---

- Används för att beräkna antalet enkla vägar i en kontrollflödesgraf
- Antal linjärt oberoende vägar:

$$CC = \#bågar - \#noder + 2$$

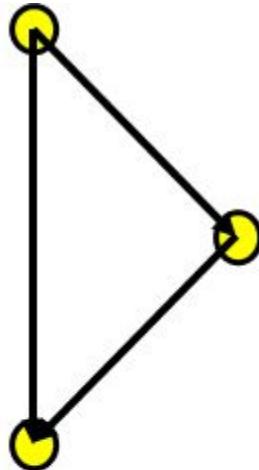
- Krav

- En enda komponent i grafen
- En unik startnod samt en unik slutnod



LUND  
UNIVERSITY

# Exempel ( $CC = \#\text{bågar} - \#\text{noder} + 2$ )

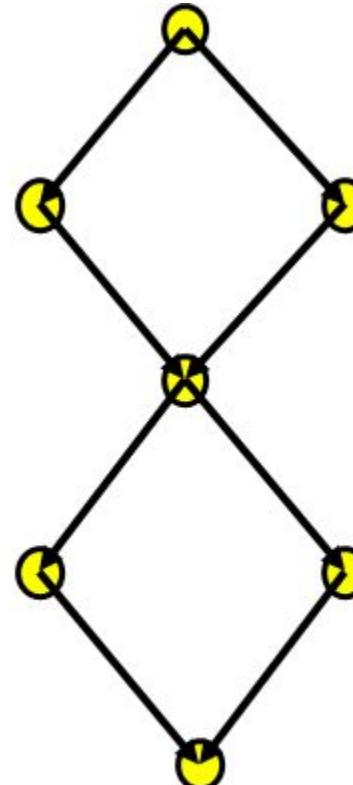


If:  
 $CC = 3-3+2 = 2$

2 linjärt oberoende vägar

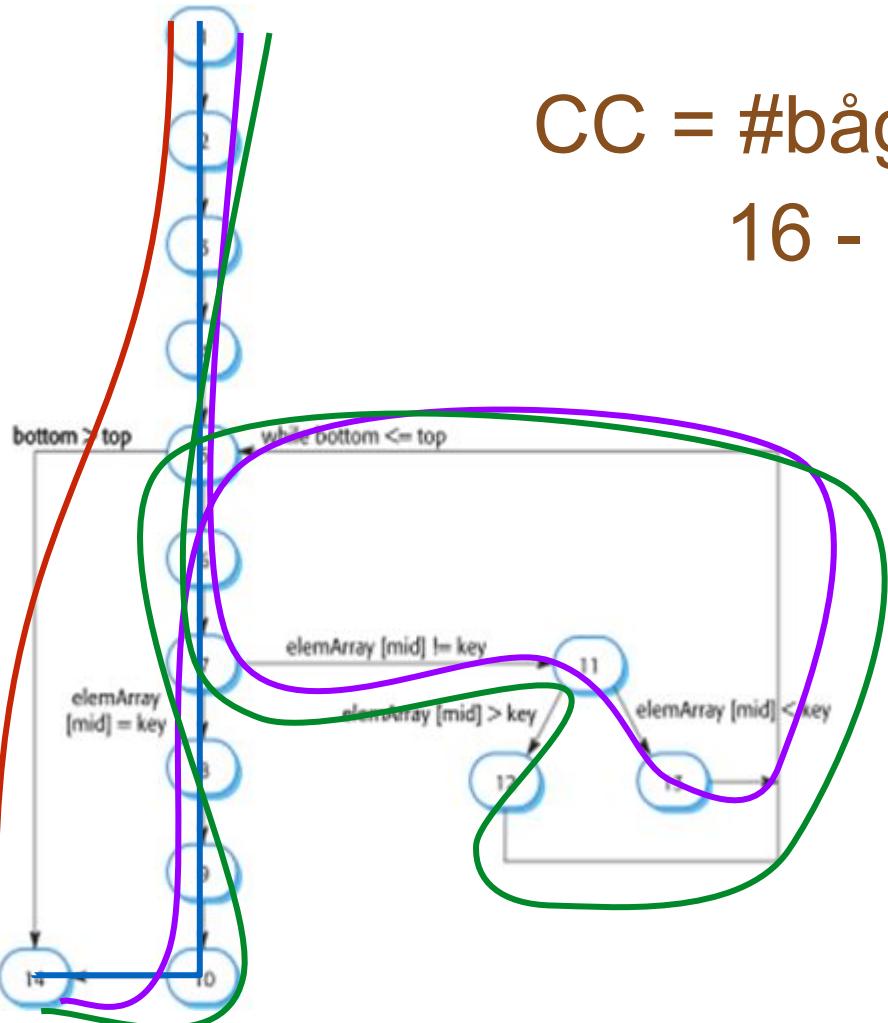
If-else:  
 $CC = 4-4+2 = 2$

$CC = 8-7+2 = 3$



3 linjärt  
oberoende  
vägar

# Ett något större exempel



$$\text{CC} = \#\text{bågar} - \#\text{noder} + 2 = \\ 16 - 14 + 2 = 4$$

CC används  
också som ett  
mått på  
komplexitet



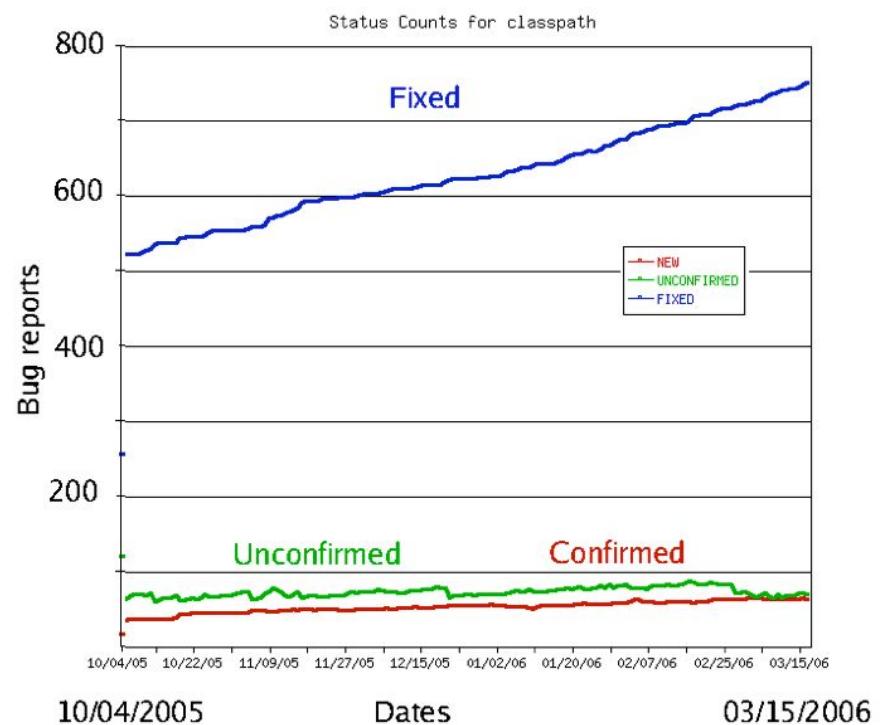
LUND  
UNIVERSITY

# Färdigtestat?

När vi gjort vad vi lovat i projekt- och testplan:

t ex visat att alla kraven på alla abstraktionsnivåer är uppfyllda:

- ...traverserat alla grenar i koden...
- ...med alla upptänkliga kombinationer av variabler...
- ...på alla plattformar...
- ...med >maxbelastning



LUND  
UNIVERSITET

# Sammanfattning - Test del 1

---

Testning kan påvisa fel, men inte bevisa att det inte finns fel

Testprocessen består av flera delprocesser: enhetstest, integrationstest, systemtest, acceptanstest

Testmetoder kan vara antingen "black box" eller "white box"

Dokumentgranskning är en testform där man kritiskt läser ett dokument på ett systematiskt sätt.

*Mer i kursen "ETSAN20 Programvarutestning"*



LUNDS  
UNIVERSITET

# Automatisk enhetstestning

---

## Fundamental idé

- Utvecklarna skriver testkod som exercerar produktionskod
  - tester konstrueras genom assertions, dvs. villkor som förväntas vara sanna
  - om någon assertion inte håller => test case **failed!**
- Organisera testfall i sviter
- Ramverk används för enkel exekvering och översikt

## Användningsområden

- Regressionstestning
- Stöd vid refaktorisering (Lab 2)
- Grunden i “test first development”

# JUnit

---

- Open source-ramverk för enhetstestning av Java  
<https://github.com/junit-team>
- Integrerat i Eclipse
- Exempel

```
public class MyFirstClassTest {  
  
    @Test  
    public void myFirstMethod(){  
        String str = "JUnit is working fine";  
        assertEquals("JUnit is working fine", str);  
  
    }  
}
```



LUNDS  
UNIVERSITET

# JUnit

---

The screenshot shows the Eclipse IDE interface with the JUnit perspective selected. The top bar includes tabs for 'Package Explorer' and 'JUnit'. Below the tabs is a toolbar with various icons for file operations like download, upload, and refresh. A message at the top states 'Finished after 0.059 seconds'. Below that, summary statistics are displayed: 'Runs: 6/6', 'Errors: 0', and 'Failures: 1'. A large red progress bar spans most of the available width. The main content area displays a tree view of test results. Under the package 'etsa02\_lab2.test', the class 'MovementSystemTest' is listed with its runner as 'JUnit 4' and a duration of '0.011 s'. Six individual test methods are shown, all of which have passed, indicated by green checkmarks and times ranging from '0.000 s' to '0.005 s'.

- ✗ etsa02\_lab2.test.MovementSystemTest [Runner: JUnit 4] (0.011 s)
  - ✓ testTwoEnemies (0.001 s)
  - ✓ testOneEnemyAtBearing180 (0.000 s)
  - ✗ testOneEnemyAtBearingNegativeGreaterThan90 (0.005 s)
  - ✓ testOneEnemyAtBearing0 (0.000 s)
  - ✓ testOneEnemyAtBearingNegativeLessThan90 (0.001 s)
  - ✓ testOneEnemyAtBearingPositiveLessThan90 (0.004 s)



LUND  
UNIVERSITET

# Exempel på JUnit assertions

---

void **assertEquals**(boolean expected, boolean actual)

void **assertTrue**(boolean condition)

void **assertNotNull**(Object object)

void **assertSame**(object1, object2)

void **assertArrayEquals**(expectedArray, resultArray)

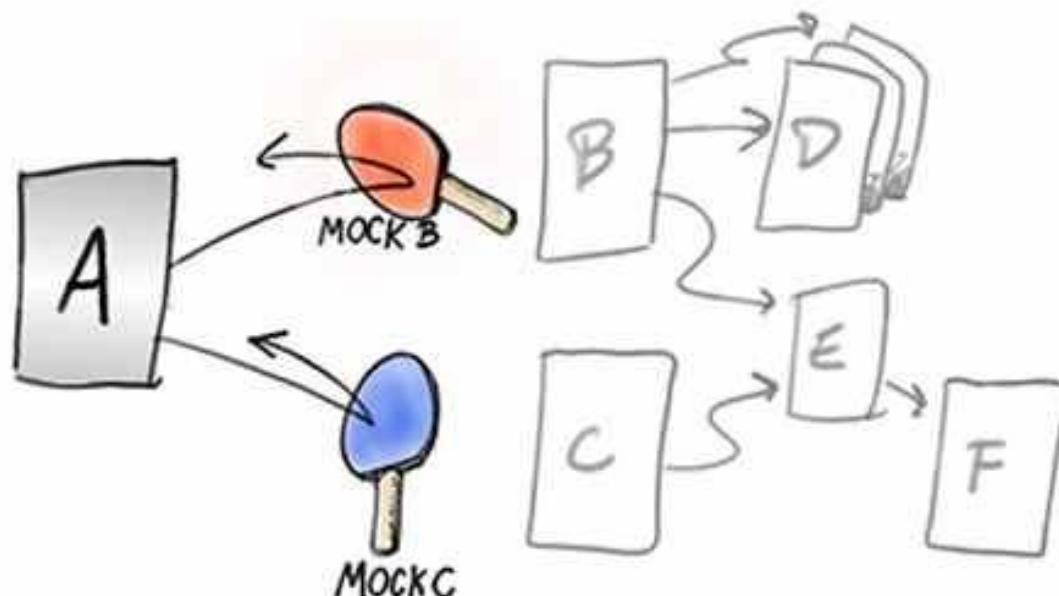


LUNDS  
UNIVERSITET

# Mockobjekt

---

- I praktiken fungerar vissa komponenter inte självständande
- Mockobjekt kan användas för att förenkla komponentens interaktion med övriga delar av systemet
  - Returnerar fejkad data för att möjliggöra enhetstest

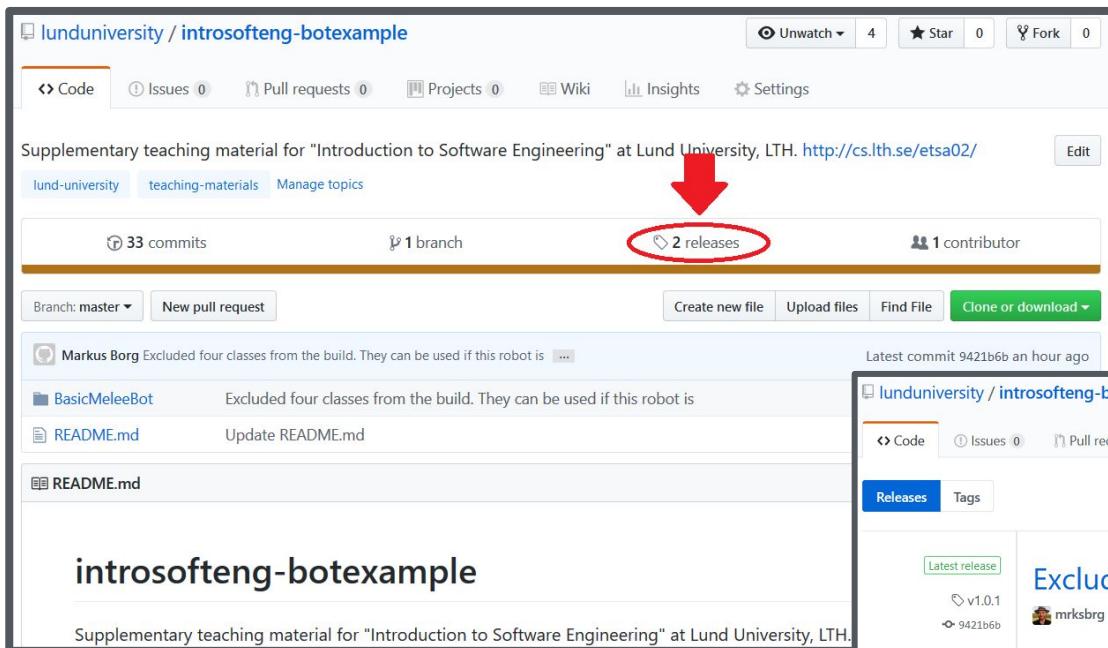


Jordan Schaenzle (2012)



LUNDS  
UNIVERSITET

# Ladda hem koden för laboration 2



The screenshot shows a GitHub repository page for `lunduniversity/introsofteng-botexample`. A red arrow points from the top of the page down to the '2 releases' button, which is highlighted with a red circle.

**Supplementary teaching material for "Introduction to Software Engineering" at Lund University, LTH. <http://cs.lth.se/etsa02/>**

**Branch: master** **New pull request** **Create new file** **Upload files** **Find File** **Clone or download**

**33 commits** **1 branch** **2 releases** **1 contributor**

**Latest commit** 9421b6b an hour ago

**Excluded four classes from the build. They can be used if this robot is ...**

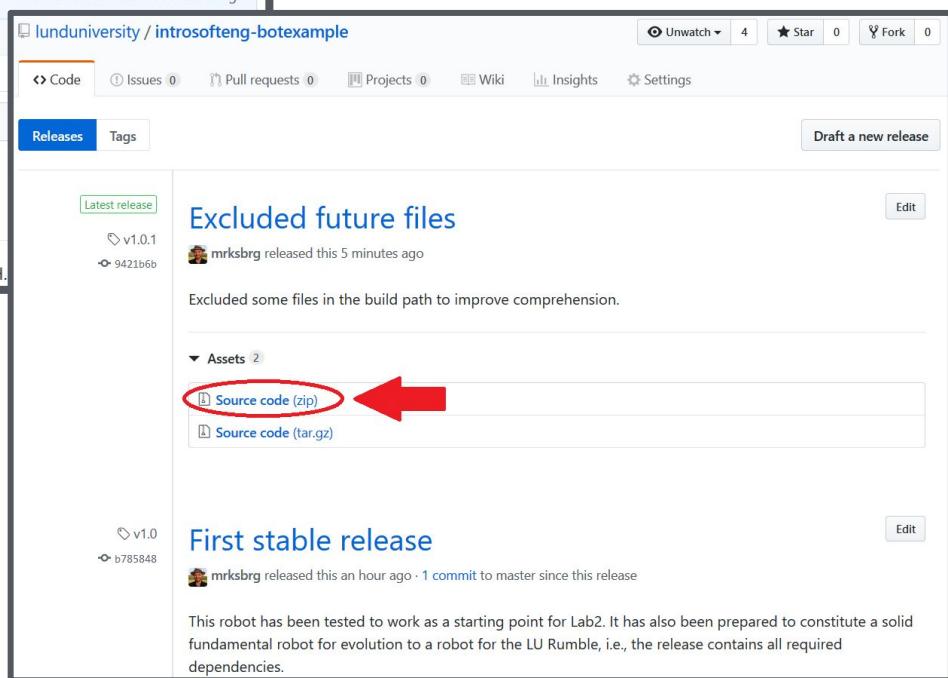
**BasicMeleeBot** Excluded four classes from the build. They can be used if this robot is ...

**README.md** Update README.md

**README.md**

## introsofteng-botexample

Supplementary teaching material for "Introduction to Software Engineering" at Lund University, LTH.



The screenshot shows the 'Releases' tab of the same GitHub repository. A red arrow points from the bottom of the left sidebar down to the 'Source code (zip)' link, which is highlighted with a red circle.

**Latest release** v1.0.1  
mrksbrg released this 5 minutes ago

Excluded some files in the build path to improve comprehension.

**Assets** 2

- Source code (zip)**
- Source code (tar.gz)**

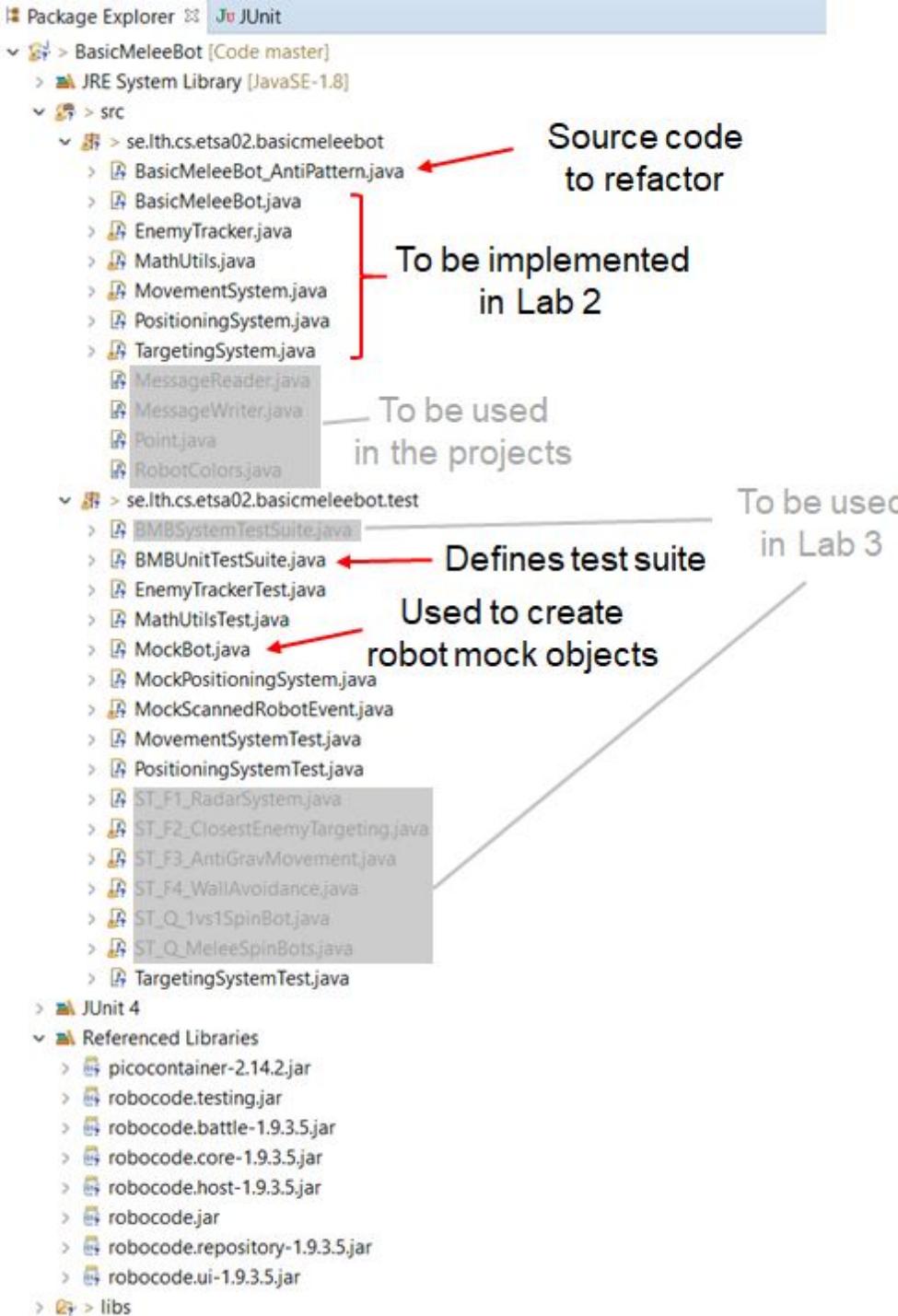
**v1.0** b785848  
mrksbrg released this an hour ago · 1 commit to master since this release

This robot has been tested to work as a starting point for Lab2. It has also been prepared to constitute a solid fundamental robot for evolution to a robot for the LU Rumble, i.e., the release contains all required dependencies.

# Getting started with Lab2 and JUnit

The screenshot shows two open windows in the Eclipse IDE:

- Import Projects from File System or Archive**: This window displays the contents of a zip archive. A red box labeled "1)" highlights the folder icon in the top right corner of the file list. Another red box labeled "2)" highlights the checkbox for importing the expanded project as an Eclipse project.
- Run Configurations**: This window is used for creating, managing, and running JUnit tests. It shows a configuration for a test class named "BasicMeleeBot" under the "JUnit" category. The configuration includes fields for "Project" (BasicMeleeBot), "Test class" (se.lth.cs.etsa02.basicmeleebot.test.BMBUnitTestSuite), and "Test method" ((all methods)). A red box highlights the green play button icon in the top right corner of this window.



Finished after 0.036 seconds

Runs: 44/44

Errors: 7

Failures: 2

## se.lth.cs.etsa02.basicmeleebot.test.BMBUnitTestSuite [Runner: JUnit 4] (0.007 s)

se.lth.cs.etsa02.basicmeleebot.test.MathUtilsTest (0.000 s)

## se.lth.cs.etsa02.basicmeleebot.test.PositioningSystemTest (0.003 s)

testCheckCloseToWallRight (0.000 s)

testGetProjectionOnWallLeft (0.000 s)

testGetProjectionOnWallBottom (0.000 s)

testCheckCloseToWallLeft (0.000 s)

testGetProjectionOnWallTop (0.000 s)

testCheckCloseToWallBottom (0.002 s)

testGetProjectionOnWallRight (0.000 s)

testCheckCloseToWallNot (0.001 s)

testCheckCloseToWallTop (0.000 s)

## se.lth.cs.etsa02.basicmeleebot.test.EnemyTrackerTest (0.001 s)

testGetEnemyCount (0.000 s)

testAddEnemies (0.000 s)

testRemoveEnemies (0.000 s)

testRemoveEnemiesNotExists (0.000 s)

## se.lth.cs.etsa02.basicmeleebot.test.TargetingSystemTest (0.001 s)

testOneEnemyAtBearing180 (0.000 s)

testOneEnemyAtBearingNeg (0.000 s)

testOneEnemyAtBearingPos (0.000 s)

testOneEnemyAtBearing0 (0.001 s)

## se.lth.cs.etsa02.basicmeleebot.test.MovementSystemTest (0.002 s)

testTwoEnemies (0.000 s)

testOneEnemyAtBearing180 (0.001 s)

testOneEnemyAtBearingPositiveGreaterThan90 (0.000 s)

testOneEnemyAtBearingNegativeGreaterThan90 (0.000 s)

testOneEnemyAtBearing0 (0.001 s)

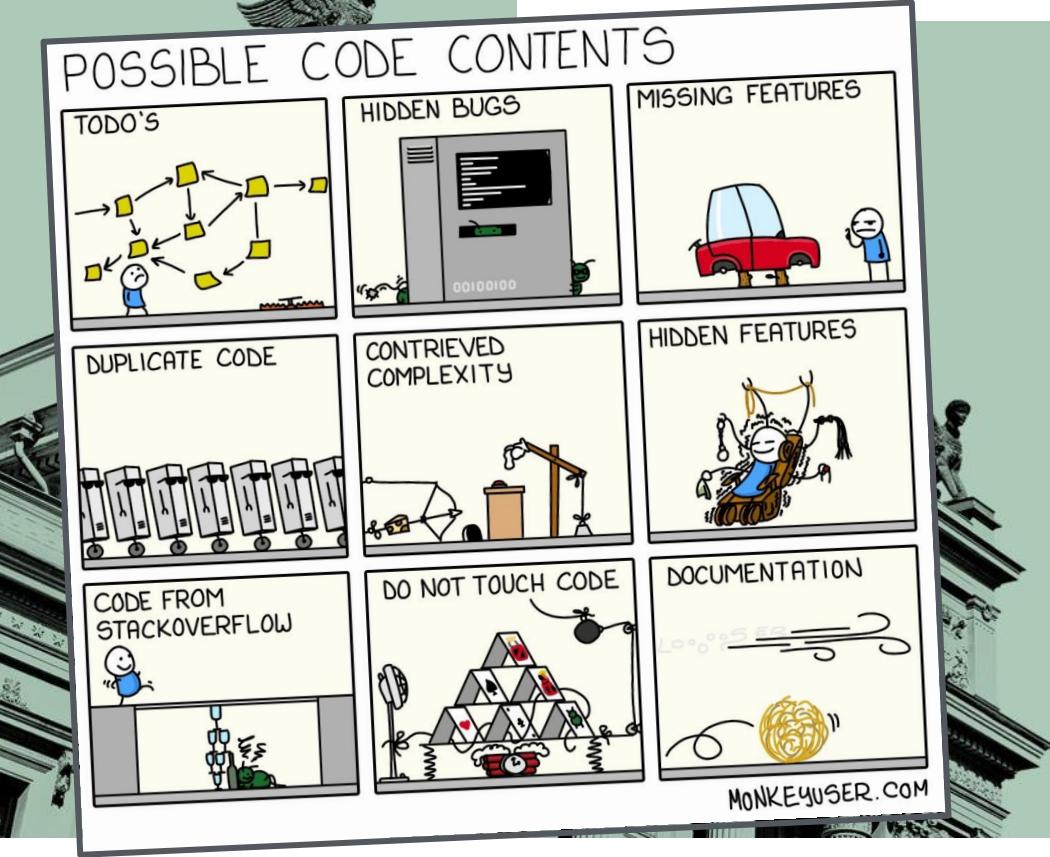
testOneEnemyAtBearingNegativeLessThan90 (0.000 s)

testOneEnemyAtBearingPositiveLessThan90 (0.000 s)



# git

Programvaruutveckling - Metodik | Markus Borg (och Patrik Persson)



# Samarbete kring källkod

---

## Versionshantering kritiskt

- Vilken version av filerna är de senaste?
- Vem gjorde vilken ändring?
- Varför gjordes ändringarna?
- I vilken ordning gjordes ändringarna?

## Vad man inte skall göra

- Skicka enskilda filer mellan enskilda eller alla i gruppen
  - E-post, Facebook Messenger, Skype, etc.
- Delade filytor som inte är skräddarsydda för källkod
  - Dropbox, Google Drive, Box, OneDrive, etc.



LUNDS  
UNIVERSITET

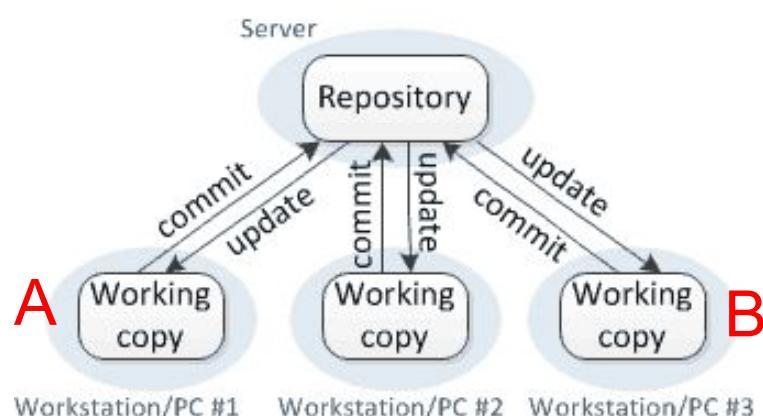
# Versionshanteringssystem

## Centraliserad versionshantering

- Varje utvecklare har en kopia av koden
- Det finns bara ett repo

## För att B ska få ändringar från A

- A gör commit
- B gör update



Prof. M. Ernst, Univ. of Washington

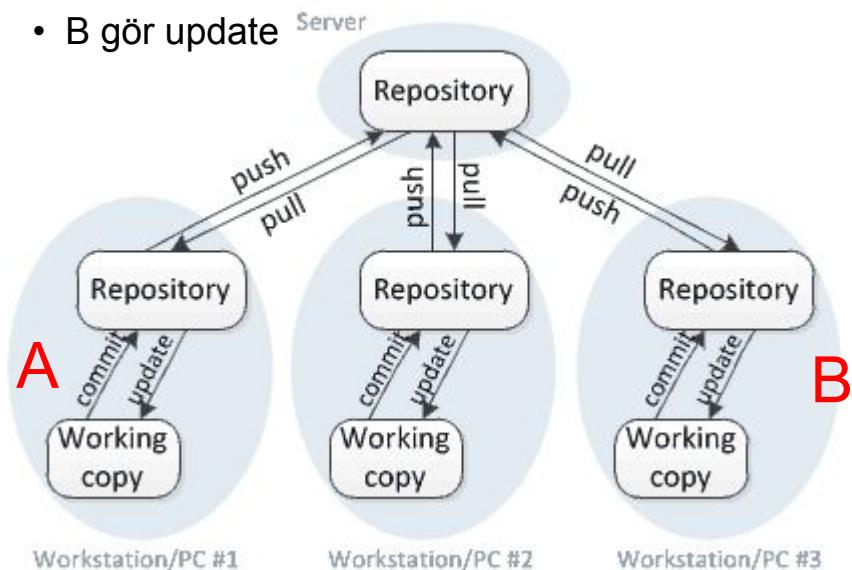
<https://homes.cs.washington.edu/~mernst/advice/version-control.html>

## Decentraliserad versionshantering

- Varje utvecklare har en klon av repot **och** en kopia av koden

## För att B ska få ändringar från A

- A gör commit
- A gör push
- B gör pull
- B gör update



# Varför så komplicerat?

---

## Fördelar med decentraliserad versionshantering

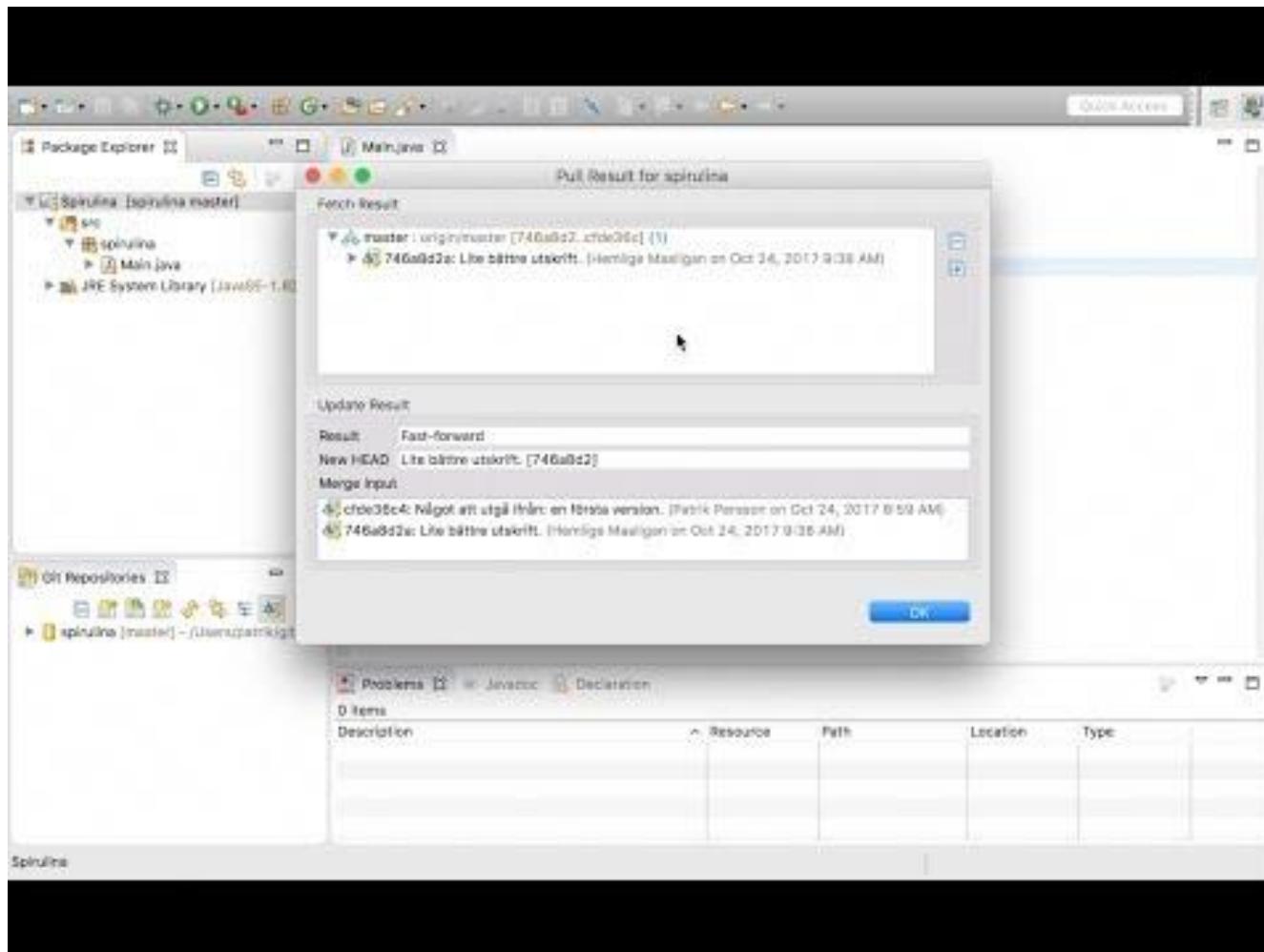
- Det finns inga låsningar. Allt åt alla, hela tiden.
  - All kod. All metadata. All ändringshistorik.
- Allt förutom pull och push går väldigt snabbt.
- Alla kan jobba offline.
- Mellansteget (commit innan push) kan vara en fördel.
- Möjliggör kraftfulla features (undvik det i ETSA02...)

## För projekten

- Lagra källkoden i **privat** repository på valfri tjänst
  - GitHub, GitLab, Bitbucket, etc.

*Mer i kursen "EDAN10 Konfigurationshantering"*

# Patrik Perssons instruktionsvideo



<https://www.youtube.com/watch?v=Ght0cOKQKjg&index=4&list=UUUVvQijppBXiVKuFJRPBrcqgt=55s>



LUNDS  
UNIVERSITET



LUNDS  
UNIVERSITET

# Programvarudesign I

Programvaruutveckling - Metodik | Markus Borg



# Objektorienterad design

---

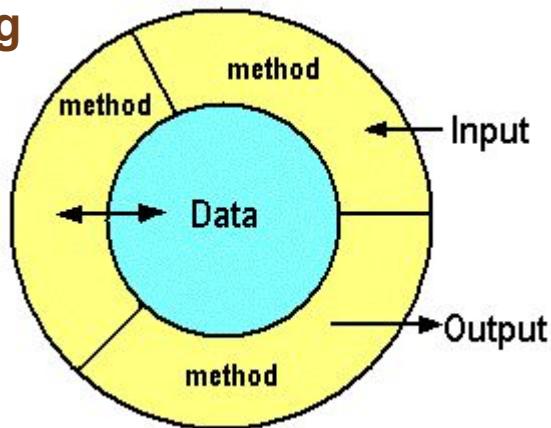
- Implementationsnära design
- Beskrivning av hur komponenter implementeras på klassnivå
- Klasser beskriver meningsfulla entiteter i problemdomänen
  - Substantiv i beskrivningen blir klasser
  - Operationer implementeras i metoder



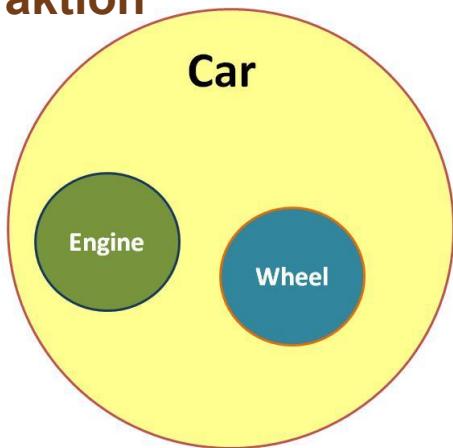
LUNDS  
UNIVERSITET

# Objektorientering - Fundamentalta koncept

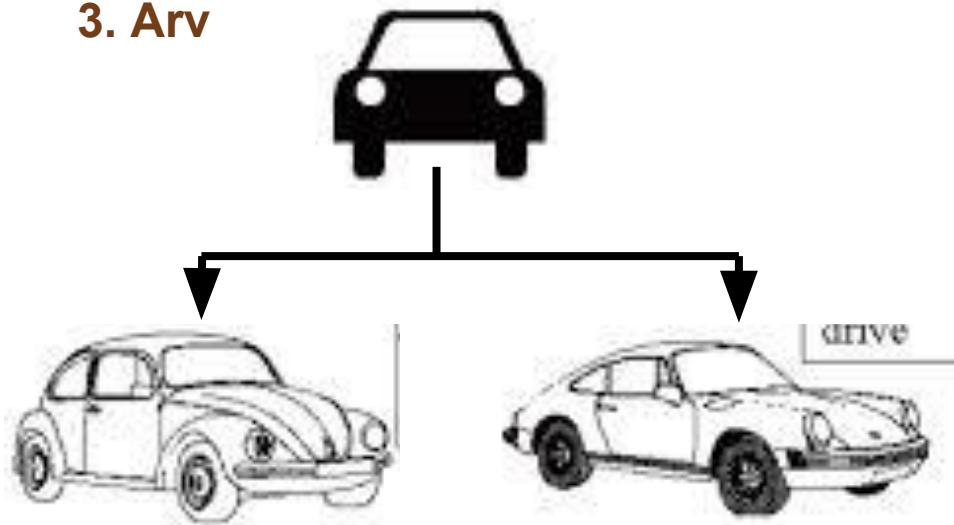
## 1. Inkapsling



## 2. Abstraktion



## 3. Arv



## 4. Polymorfism



# Unified Modeling Language - UML

---

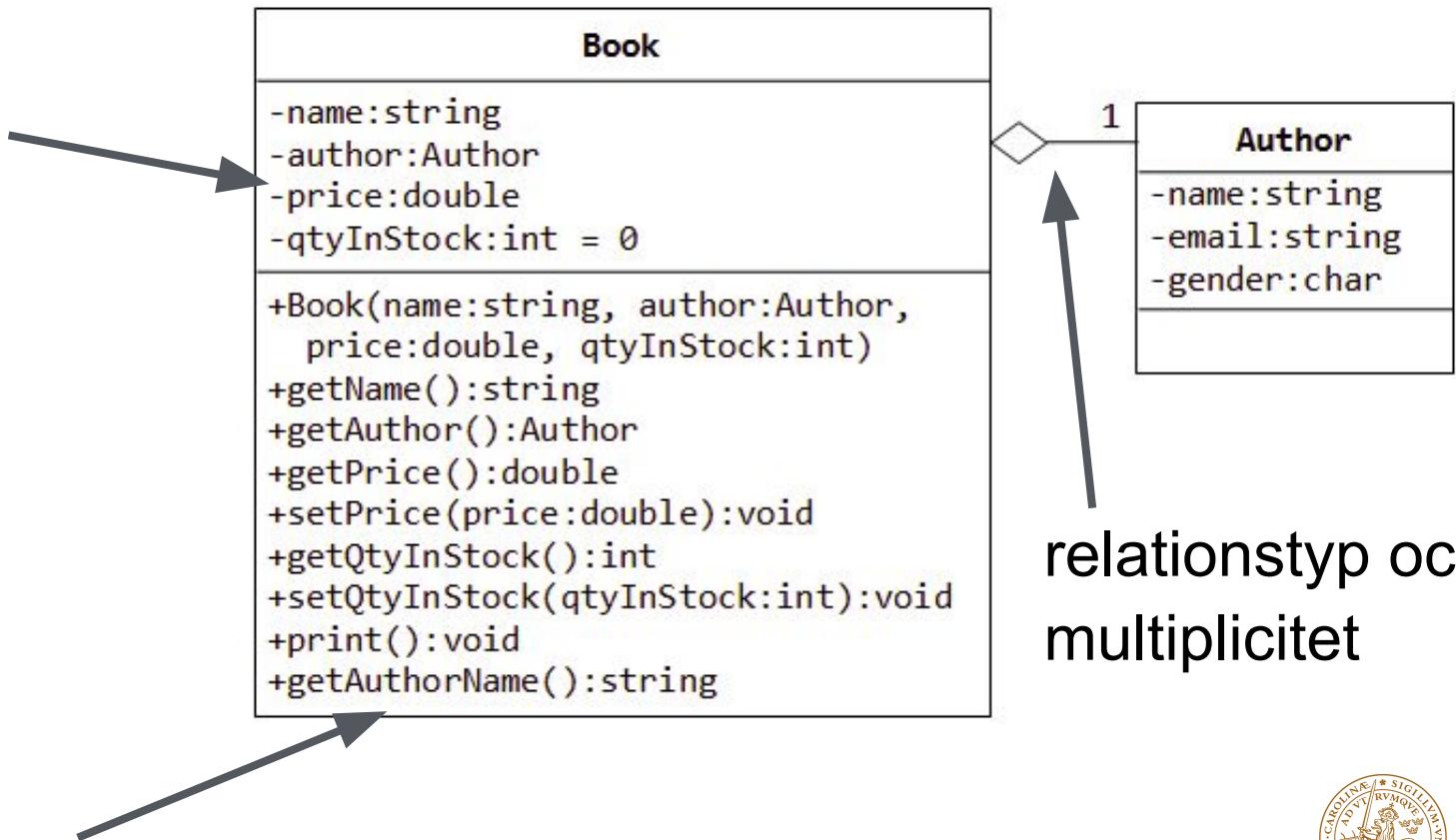
- Generellt språk för att modellera programvarusystem
  - Standardiserat av ISO
  - Tre skapare, varav en svensk: Ivar Jacobson
- Statisk modell av systemet
  - Klassdiagram
- Dynamisk modell av systemet
  - Sekvensdiagram

*Mer i kursen "EDAF60 Objektorienterad  
modellering och design"*



# Klassdiagram

+ public  
- private



parametrar och returtyper

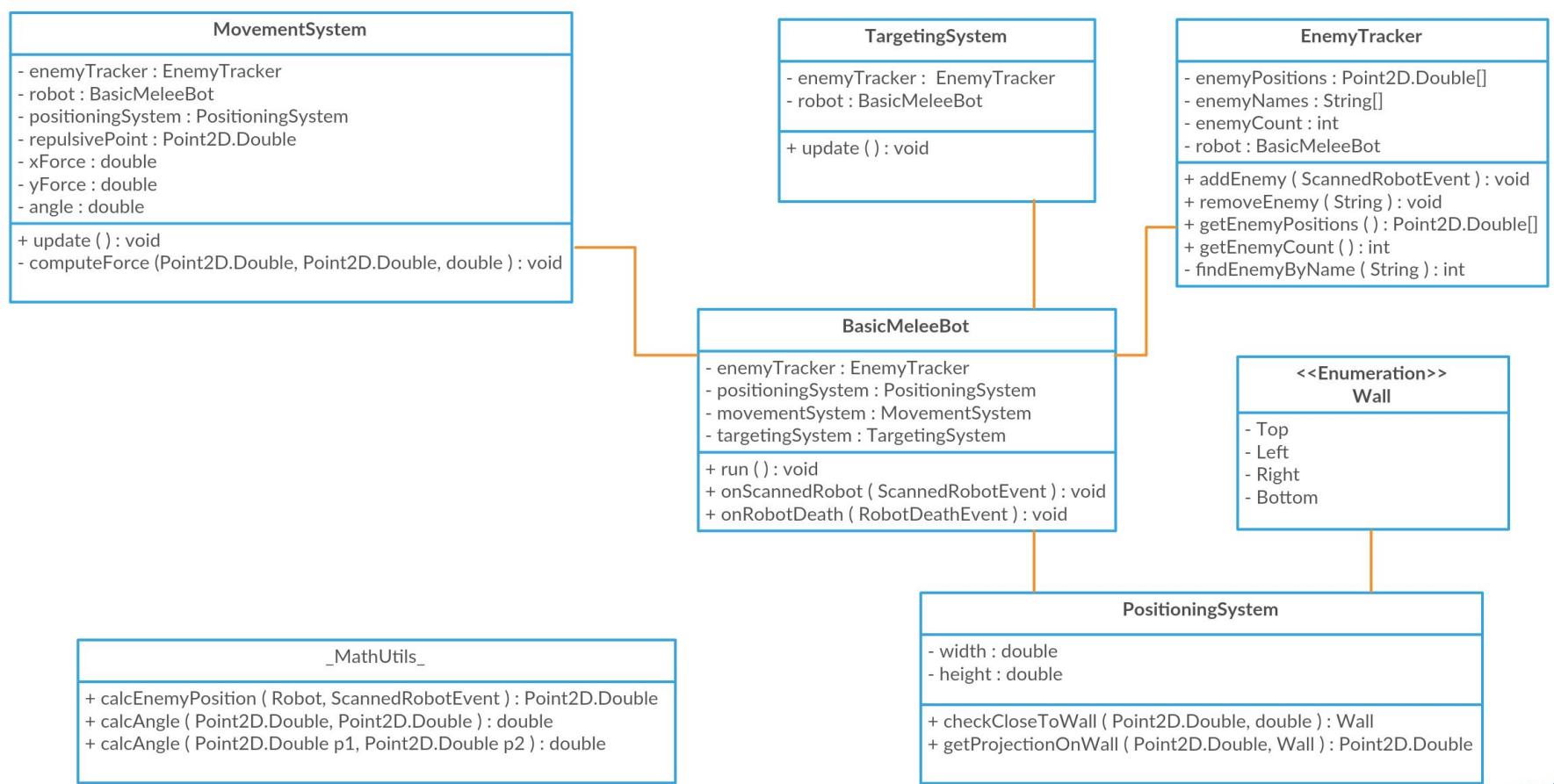
relationstyp och  
multiplicitet



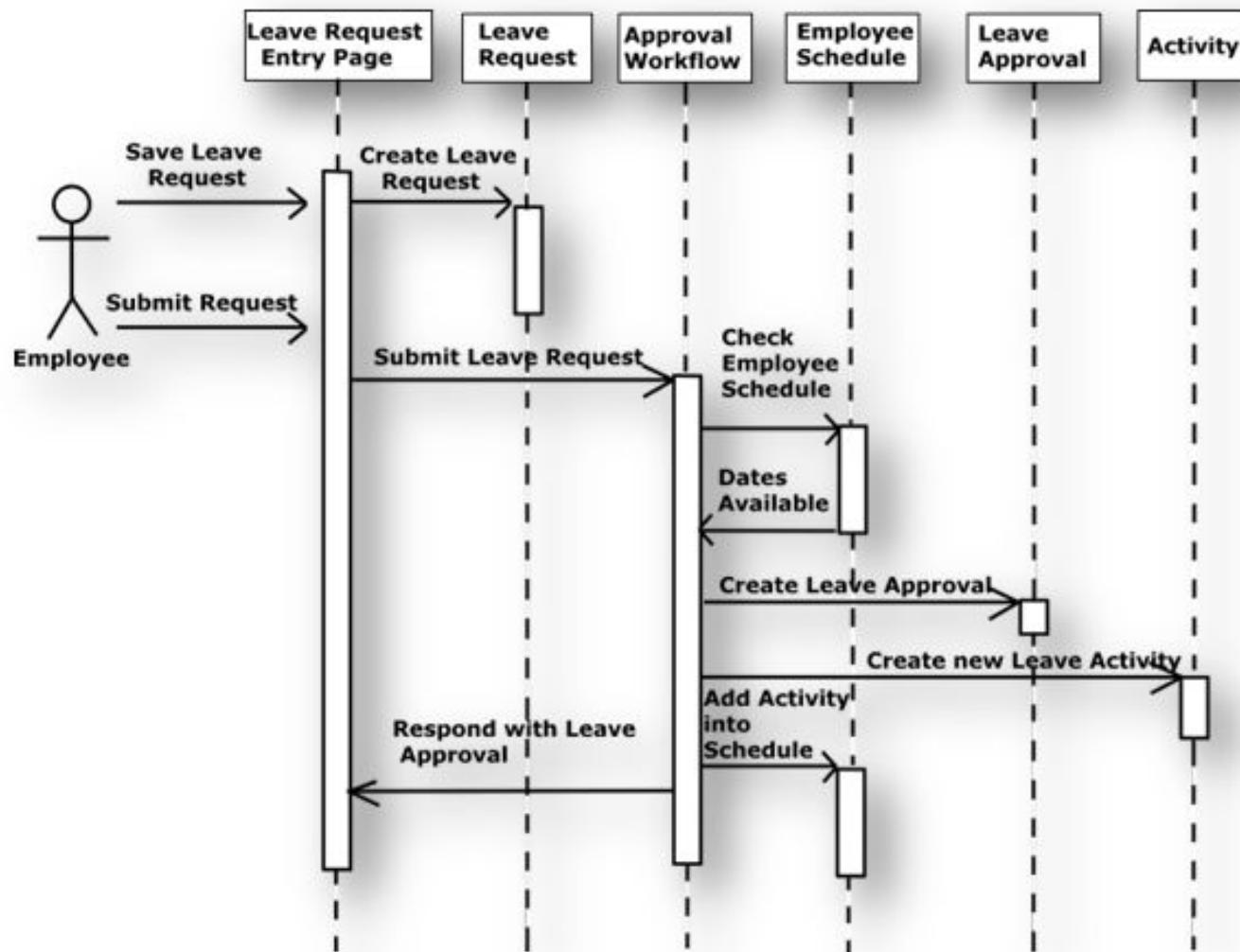
LUNDS  
UNIVERSITET

# Lab 2 - refaktorisera enligt klassdiagram

Package etsa02\_lab2



# Sekvensdiagram

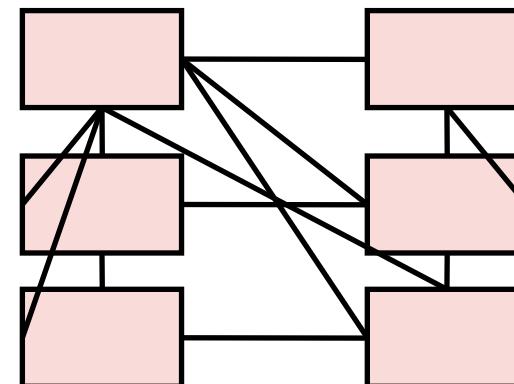
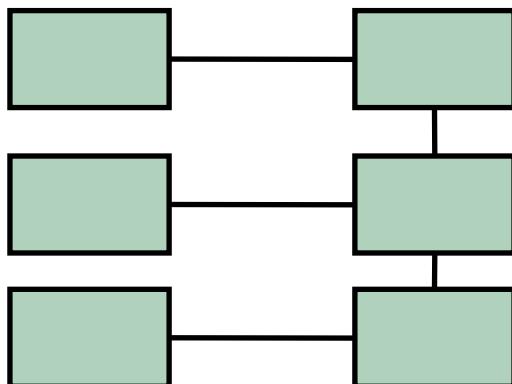


LUNDS  
UNIVERSITET

(c-sharpcorner)

# Designmål: Låg koppling (coupling)

- I hur stor utsträckning är klasser i programmet kopplade till varandra?
- Låg koppling underlättar underhåll och evolution



LUNDS  
UNIVERSITET

# Designmål: Hög samhörighet (cohesion)

---

Hur väl innehållet i en klass "hänger samman"

Exempel:

- **Logisk** – t.ex. en klass som sköter all utmatning av data
- **Temporal** – t.ex. en klass som sköter all "uppstart" eller "avslut"
- **Procedurbaserad** – aktiviteter som utförs efter varandra slås ihop
- **Kommunikationsbaserad** – delar som behandlar samma data slås ihop
- **Sekventiell** – kedja av aktiviteter som hänger ihop i sekvens
- **Funktionell** – innehållet står för en enstaka funktion, t ex "sortera vektor"



LUNDS  
UNIVERSITET

# Designmönster

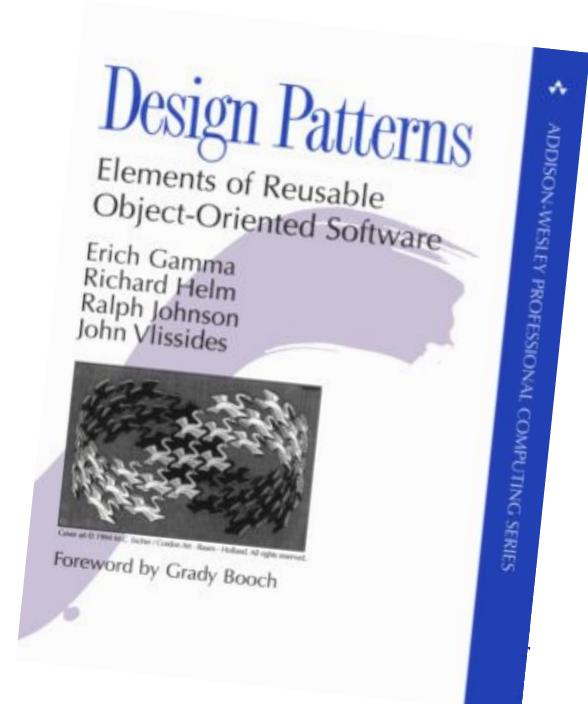
---

Beprövade lösningar på återkommande problem

- Ursprungligen från arkitektur
- Inom programvara används objektorienterade koncept

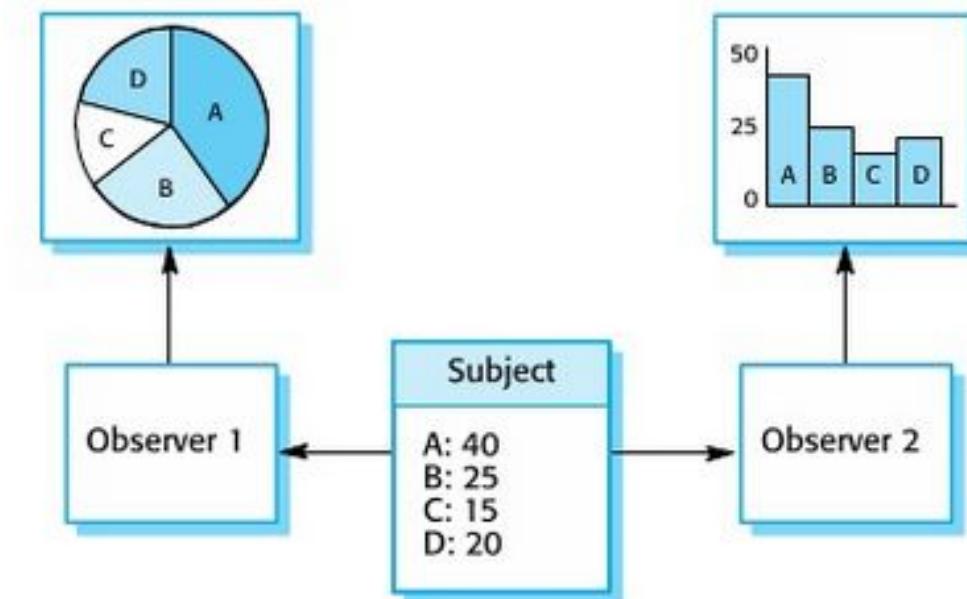
Några exempel:

- **Singleton** garanterar ett unikt objekt av en klass
- **Iterator** traverserar en samling objekt
- **Visitor** gör en operation på samtliga objekt i en samling



# Exempelmönster: Observer

- Separera ett objekts tillstånd från presentation
- Möjliggör flera olika vyer



LUNDS  
UNIVERSITET



# Tentafrågor - exempel

Programvaruutveckling - Metodik | Markus Borg



# Tentafråga VT 2012

## U1 Test hela vägen (15 p)

Beskriv de olika nivåerna av dynamisk testning under ett mjukvaruutvecklingsprojekt från enhetstestning till acceptanstest. För varje nivå bör beskrivningen åtminstone svara på följande frågor: Vem? Vad? När? Hur? Hur länge? Varför? Använd åtminstone följande begrepp i rätt sammanhang:

Acceptanstest

Kravtäckningsmatris

Testprotokoll

Black-box-test

Parvis testning

Validering

Branch coverage

Regressionstest

Verifiering

Enhetstest

Stopkriterium

White-box-test

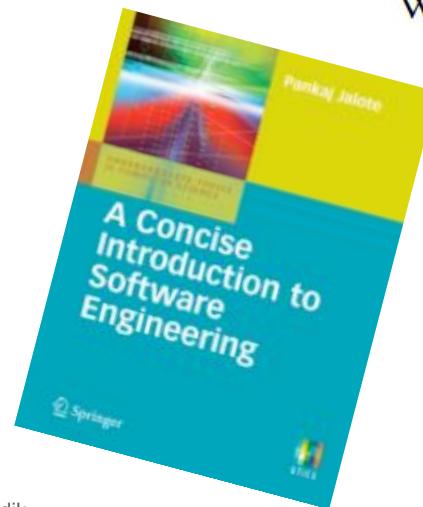
Gränsvärdestestning

Stresstest

Integrationstest

Systemtest

#	Datum	Storlek	Ändring	Editor	Kommentar	Ätgärd
42	2010-03-27 07:29:30	1237	<input type="radio"/> till fängslande	JonesWibomart	Visa	
41	2010-03-27 07:26:26	1238	<input type="radio"/> till fängslande	JonesWibomart	Visa	
40	2010-03-26 20:59:48	1237	<input type="radio"/> till fängslande	JonesWibomart	Visa	
39	2010-03-26 07:44:16	1238	<input type="radio"/> till fängslande	JonesWibomart	Visa	
38	2010-03-26 07:44:04	1273	<input type="radio"/> till fängslande	JonesWibomart	Visa	
37	2010-03-26 07:45:57	1968	<input type="radio"/> att återhämtas	JonesWibomart	Visa	



LUND  
UNIVERSITET

# Tentafråga VT 2012

Förslag

## U1 Test hela vägen (15 p)

- Förstå skillnad på **verifiering/validering** samt **statisk/dynamisk** testning

### Enhetstestning

- Av utvecklarna medan källkod skrivs, whitebox på klassnivå
- Testsvit används som regressionstest

### Integrationstest

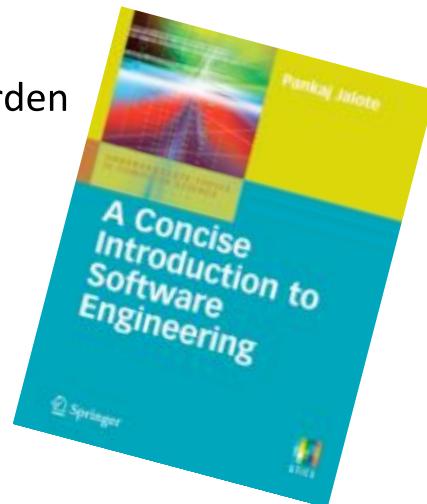
- Av utvecklarna efter hand som klasser blir klara, whitebox med kodtäckning, stoppkriterium?

### Systemtest

- Av testarna, testar helheten, parvis testning, gränsvärden
- blackbox baserat på kraven mot kravtäckningsmatris
- Stressstest?

### Acceptanstest

- Av testarna med kundrepresentant, blackbox



LUNDS  
UNIVERSITET



# Tentafråga VT 2012

## U3 Samma sak på olika sätt (15p)

Ett *användarfall* (*use-case*), ett *acceptanstestfall* och ett *scenario* i en *användarmanual* kan mycket väl beskriva i princip samma tänkta händelseförlopp.

- Börja med att formulera ett eget *användarfall* på kanske 5-10 punkter för vilket system som helst som inte har att göra med cykelgarageprojektet eller de andra exemplen som används i kursen. Systemet kan finnas på riktigt eller vara påhittat.

Formulera därefter ett (1) *acceptanstestfall* och en (1) *manuellttext* som relaterar till användarfallet.

Inled med en kort beskrivning av ditt tänkta system så man förstår användarfallets sammanhang. Presentera sedan gärna användarfallet, testfallet och manualtexten i en tabell med tre kolumner så att de blir lättare att jämföra med varandra.

- Jämför de tre olika beskrivningarna och diskutera varför de skiljer sig åt.



LUND  
UNIVERSITET

# Tentafråga VT 2012

Förslag

## U3 Samma sak på olika sätt (15p)

System för att administrera **badmintonlävlingar**. Exempel: **registrera match**

### Användningsfall

1. Skanna QR-kod på rapport
2. Mata in set-siffror
3. Mata in start-/sluttid
4. Mata in antal använda bollar
5. Klicka på 'Bekräfta'

### Testfall

1. Skanna QR-kod på rapport TEST\_A
2. Mata in set-siffrorna "15-21, 21-17, 21-8"
3. Mata in starttid "10:25" och sluttid "11:30"
4. Mata in antal använda bollar: 12
5. Klicka på 'Bekräfta'
6. Verifiera att "Match 22" presenteras i lista 2

### Scenario i manual

1. Använd QR skannern för att registrera din match
2. Fyll i matchens resultat, uppdelat per set
3. Fyll i klockslag då matchen började resp. slutade
4. Mata in hur många fjäderbollar som användes
5. Klicka på 'Bekräfta' för att lagra matchresultatet



LUNDS  
UNIVERSITET



# Tentafråga VT 2012

---

## U4 Spårbarhet (10p)

Raja and Kamran genomförde 2008 en systematisk kartläggningsstudie som en del av ett examensarbete i software engineering vid Blekinge Tekniska Högskola. En av författarnas forskningsfrågor rörde vilken definition av spårbarhet som det oftast refereras till i vetenskapliga artiklar. Studenterna fann att den definition som vanligast används är:

*"(Requirements) traceability refers to the ability to describe and follow the life of a requirement, in both a forward and backward direction (i.e. from its origin, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)" (Gotel and Finkelstein, 1994)*

Frågor:

- I cykelgarageprojektet underhölls spårbarhet mellan systemkrav och systemtester. Ge exempel på aktiviteter i ett programvaruutvecklingsprojekt som kan underlätta att denna typ av spårbarhet finns tillgänglig.
- Spårbarhet kan även existera mellan andra typer av utvecklingsartefakter, t.ex. krav-källkod, källkod-testfall, källkod-manual och defektrapport-krav. Att bestämma vilken typ av spårbarhet som ska underhållas innebär ett avvägande. Redogör för nackdelar med en hög nivå av spårbarhet. Föreslå även exempel på typer av projekt där spårbarhet trots detta brukar anses nödvändigt.

# Tentafråga VT 2012

Förslag

## U4 Spårbarhet (10p)

Underlättas av spårbarhet:

- Buggrättning
- Ändringshantering
- Vidareutveckling
- Regressionstestning
- Generell systemförståelse
- ...

Nackdelar med spårbarhet. När nödvändigt?

- dyrt att upprätta, särskilt i legacy-system
- krävs mycket arbete att underhålla vid förändringar, svårt för agil utveckling
- lätt att peka ut fel kopplingar, t.ex. för många
- ingår i standarder för säkerhetskritisk utveckling



LUNDS  
UNIVERSITET

# Tentafråga VT 2012

---

## U5 Statisk testning (10 p)

Beskriv hur processen för dokumentgranskning bör gå till. Ange t. ex. inblandade roller, delaktiviteter och delresultat. Diskutera kostnader i form av arbetstid och ledtid, d.v.s. hur lång tid man kan tänkas behöva mellan de olika aktiviteterna för att det ska vara realistiskt att genomföra hela granskningsprocessen.

Ge exempel på typer av fel och problem som är lättare att identifiera vid dokumentgranskning än vid dynamisk testning. Motivera dina svar.

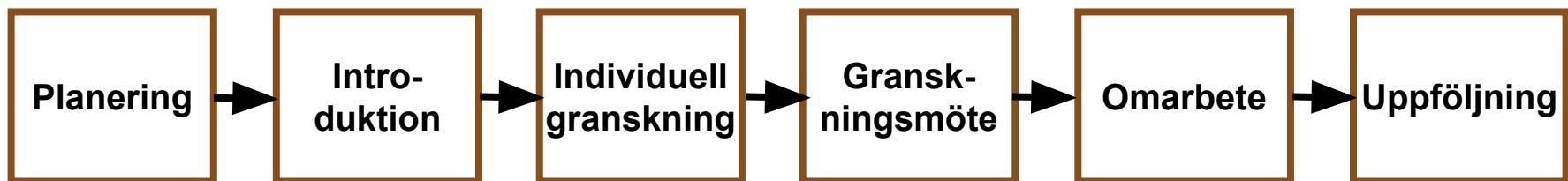


LUNDS  
UNIVERSITET

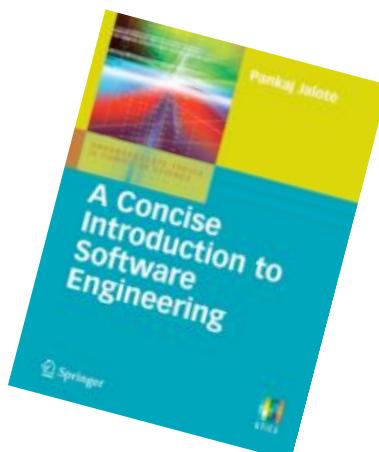
# Tentafråga VT 2012

Förslag

## U5 Statisk testning (10 p)



- Roller vid mötet: ordförande, sekreterare, författare, granskare
- Rimligt med inbjudan 1-2 veckor innan mötet
- Individuell granskning: 5-15 sidor/h beroende på komplexitet
- Granskningsmöte: max 2 h
- Uppföljningsmöte en vecka senare



# Kursombud

---

2 X C:

TBD

TBD

2 X I:

TBD

TBD



# Att göra inför Lab 2

---

- Läs labinstruktionen, inkl. läsanvisningarna
- Ladda hem källkoden som ska användas
  - finns på GitHub
- Prova gärna att köra JUnit
  - annars löser vi det i datorsalen!



LUND  
UNIVERSITET