

Föreläsning 3

Tobias Wrigstad

Resten av kursen...



Så här långt borde du ha...

1. Gått på två föreläsningar
2. Klarat 4/6 labbar
3. Ha tittat lite på kursens webbsida
4. Ha skaffat ett GitHub-konto och registrerat det hos oss
5. Ha loggat in på Piazza minst en gång

**Från och med nästa vecka: Labbar på
IOOPM avser tid att redovisa och få
hjälp med programmering.**

**Den mesta programmeringen sker
utanför schemalagd tid!**



Din uppgift

Att lära dig så mycket som möjligt om imperativ- och objektorienterad programmering, verktyg, programmeringsmetodik och process

Vår uppgift

Att hjälpa dig i din inlärningsprocess.

Att se till att det leder till att du blir godkänd på kursen.



IOPM 2016

Tobias Wrigstad

Kursansvarig

Elias Castegren

Huvudassistent



Imperativ och ObjektOrienterad ProgrammeringsMetodik



Imperativ och ObjektOrienterad ProgrammeringsMetodik

Del 1 & 2



Imperativ och ObjektOrienterad ProgrammeringsMetodik

Del 2



Imperativ och ObjektOrienterad ProgrammeringsMetodik

Del 1, 2 & 3



Efter godkänd kurs ska studenten kunna [1/2]

- förklara hur program **exekverar** och beskriva hur program **lagrar och hanterar information**.
- förklara och exemplifiera **kvalitativa** och **kvantitativa aspekter** av ett programs **design** och **implementation**.
- förklara skillnaden mellan **manuell och automatisk minneshantering** samt grundläggande relaterade begrepp (som stack, heap, statisk minnesarea) och använda **verktyg** för felsökning av minnesfel.
- förklara hur **större programuppgifter** kan lösas & resonera om olika lösningsalternativ.
- redogöra för hur **enkla parallellicerbara problem kan lösas** effektivt med relevanta hjälpmittel.
- **designa, koda, granska, testa, felsöka och dokumentera** egna program, med hjälp av lämpliga **verktyg**.
- **läsa, förstå och modifiera** icke-trivial **kod** som studenten själv inte har skrivit samt **integrera nyskriven kod med existerande**.



Efter godkänd kurs ska studenten kunna [2/2]

- beskriva – **oberoende av programkoden** – den uppgift ett program skall lösa och de förutsättningar som krävs för att det skall kunna arbeta. Motivera varför programmet under dessa förutsättningar är korrekt, **samt specificera och konstruera testfall och köra tester för att verifiera detta.**
- skriva lämplig **dokumentation för programmering** och **testhantering**.
- tillämpa specifika **utsnitt av kända utvecklingsprocesser** och -metodiker (ex. **agil utveckling, parprogrammering** och testdriven utveckling).
- beskriva olika former av **testning** och deras vikt i olika skeden av **utvecklingsprocessen**.
- bidra till ett **konstruktivt samarbete i programmeringsprojekt**.
- **presentera** och **diskutera** kursens innehåll **muntligt** och **skriftligt** med för utbildningsnivån lämplig färdighet.



Mål

Unlockable Achievements (aka kursmål/kunskapsmål)

Name	Short desc	Grade level	Assessment
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar	3	L
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt	3	L
A3	Demonstrera förståelse för designprincipen informationsgömning i ett C-program med hjälp av .c och .h-filer	4	L, G
B4	Använda arv, metodspecialisering och superanrop i ett program som drar nytta av subtypspolymorfism	3	L

↑

ID

↑

Ingress

↑

Nivå

3

4

5

↑

Hur målet kan redovisas

L – Labb

W – Essä

T – Tobias

P – Presentation

R – Rapport



Mål

Unlockable Achievement (kursmål/kunskapsmål)

Name	Short desc
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer bakom väldefinierade gränssnitt
A3	Demonstrera förståelse för designprincipen information hiding genom att implementera ett C-program med hjälp av .c och .h-filer
B4	Använda arv, metodspecialisering och superanrop för att lösa problem som drar nytta av subtypspolymorfism

(A. Abstraktion)

A1

Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och undvika upprepningar

Level Assessment

3 L

Abstraktion är en av de viktigaste programmeringsprinciperna. Vi vet att djupt, djupt nere under huven är allt bara ettor och nollor (redan detta är en abstraktion!), men ovanpå dessa har vi byggt lager på lager av abstraktioner som låter oss tala om program t.ex. i termer av strukturer och procedurer.

Proceduren `ritaEnCirkel(int radie, koordinat center)` utför beräkningar och tänder individuella pixlar på en skärm, men i och med att dessa rutiner kapslats in i en procedur med ett vettigt namn, där indata är uttryckt i termer av koordinater och radie har vi abstraherat bort dessa detaljer, och det blir möjligt att rita cirklar tills korna kommer hem utan att förstå hur själva implementationen ser ut.

Väl utförd abstraktion döljer detaljer och låter oss fokusera på färre koncept i taget.

Du bör ha en klar uppfattning om bland annat:

- Varför det är vettigt att identifiera liknande mönster i koden och extrahera dem och kapsla in dem i en enda procedur som kan anropas istället för upprepningarna?
- Abstraktioner kan "läcka". Vad betyder det och vad får det för konsekvenser?
- Vad är skillnaderna mellan "control abstraction" och "data abstraction"?
(Du kan läsa om dessa koncept på t.ex. [Wikipedia](#)).



Notera: målens namn/nummer kan ha ändrats

Redovisning

- **Din uppgift:** att övertyga examinatorn om att du uppfyller målet

Ge **exempel** från den kod (etc.) du har skrivit

Inga **core dumps**

Ha en **story** för hur allting hänger ihop

- Försök att alltid redovisa flera mål åt gången

Mindre arbete, mindre väntetid

Synergi!

- Demo av redovisningsystemet sker på föreläsning innan första redovisningstillfället



Var finns information om kursen?

The screenshot shows the IOOPM 2016 homepage with a yellow header and a city skyline background. It features a navigation bar with links like HOMEPAGE, DEADLINES, OM KURSEN, PROCESS, MÅL, UPPGIFTER, LÄNKAR, KONTAKT, and HJÄLP!. Below the header, there's a section titled 'KURSDELAR' with a box containing 'IMPERATIV OCH' and 'wrigstad.com/ioopm'. A yellow box highlights the URL 'wrigstad.com/ioopm'.

Hur lämnar jag in en uppgift?

The screenshot shows a GitHub repository page for 'IOOPM-UU / ioopm15'. It displays a list of commits, branches, and pull requests. A yellow box highlights the URL 'wrigstad.com/ioopm'.

GitHub

Hur interagerar jag med er?

The screenshot shows the Piazza platform interface. It includes a sidebar with 'Class at a Glance' statistics and a main feed of posts. A yellow box highlights the word 'Piazza'.

Piazza

Hur redovisar jag?

The screenshot shows the AU Portal with a section titled 'Unlockable Achievements (aka kursmål/kunskapsmål)'. It lists various achievements with descriptions, grades, and edit/delete buttons. A yellow box highlights the word 'AU Portal'.

AU Portal



Du börjar här:

The screenshot shows a web browser window with the URL wrigstad.com in the address bar. The main title is **IOOPM 2016** with the subtitle **IMPERATIV ... OCH SÅ VIDARE**. Below the title is a navigation menu with links: SCHEMA (highlighted in dark grey), DEADLINES, HUR FUNKAR IOOPM?, PROCESS, PROGRAMMERA, MÅL, KODPROV, LÄNKAR, KONTAKT, and HJÄLP!. The background features a blurred image of a city skyline at sunset. On the left side of the page, there is a sidebar with the heading **LÄNKAR** and links to [Inlämningsuppgifter](#) and [AU Portal](#). At the bottom of the sidebar is a button labeled [Visa en meny](#). To the right of the sidebar is a large central text block: **IMPERATIV OCH
OBJEKTORIENTERAD
PROGRAMMERINGS-METODIK**, with the website URL wrigstad.com/ioopm displayed below it in a yellow box.

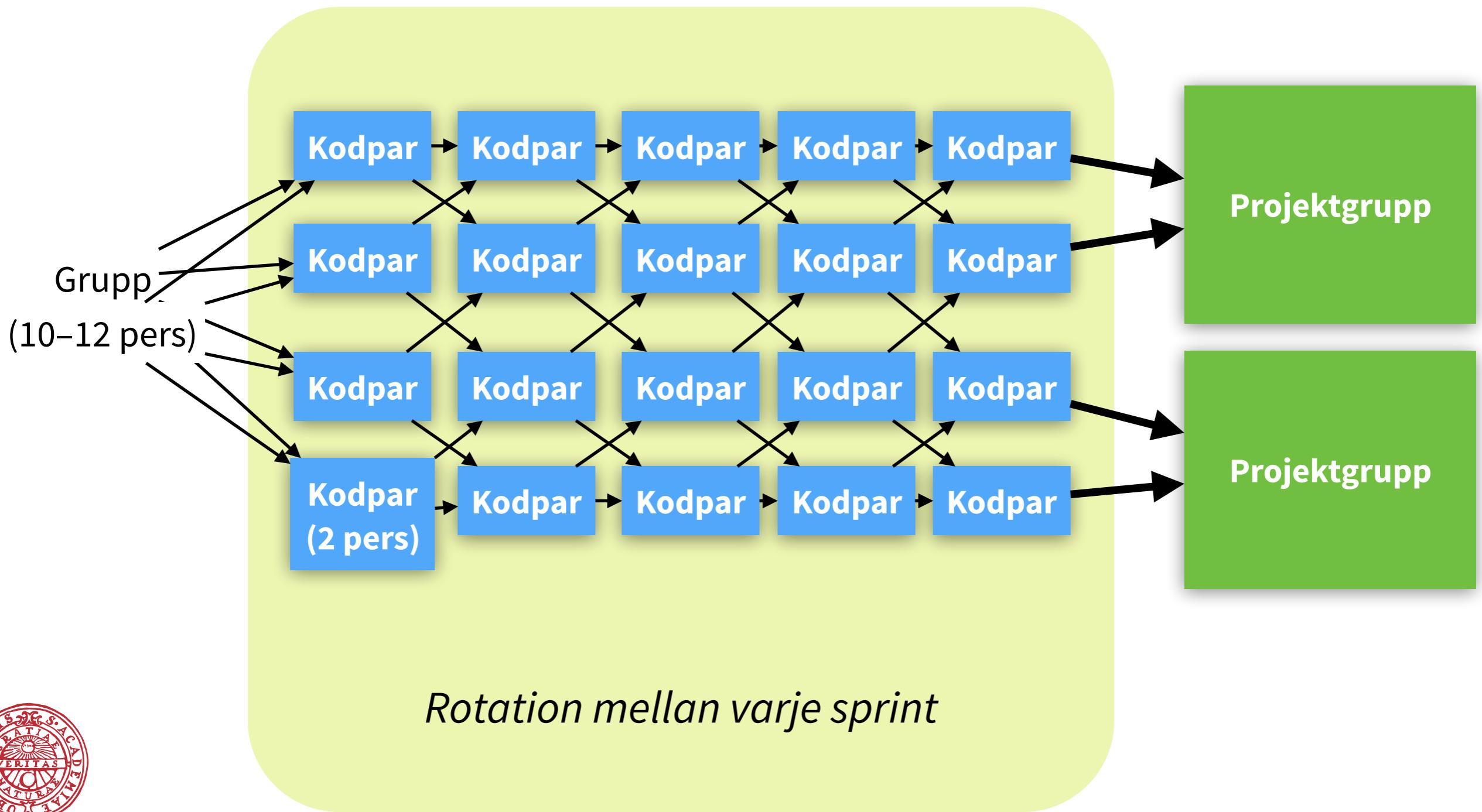


Översikt

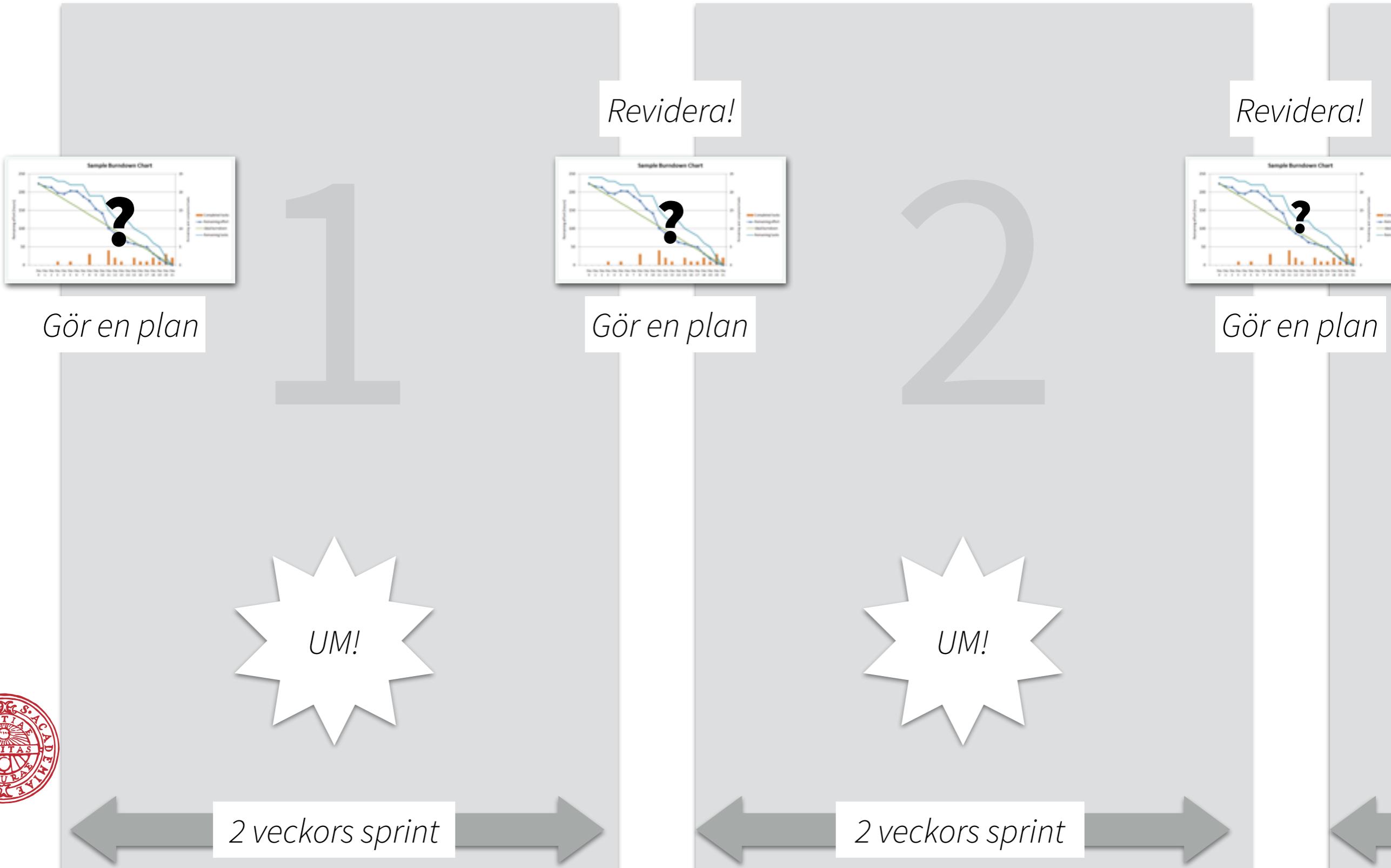
Fas 1 (Sep–Okt)					Fas 2 (Okt–Nov)				Fas 3 (Dec–Jan)		
Imperativ programmering i C					OO med Java				Projektet		
<i>Intro</i>	<i>Intro</i>	<i>Sprint 1</i>	<i>Sprint 2</i>	<i>Kodprov</i>	<i>Intro</i>	<i>Sprint 1</i>	<i>Sprint 2</i>	<i>Kodprov</i>	<i>Fria sprintar</i>		
		UM 1	UM 2			UM 4	UM 5				
		Lager	Lager Resepl.			Kassakö Twitterish	MUD Kalkylator				<i>UM = uppföljningsmöte</i>
14 föreläsningar 15 labbar				10 föreläsningar 10 labbar				1 föreläsning 4 labbar (initialt)			
<i>Deadlines</i>											
<i>Essä</i>											



Grupper och kodpar [rotationerna sköter ni själva]



Fas [kursen har 3]



Sprint [varje fas har minst 2]

Välj uppgift



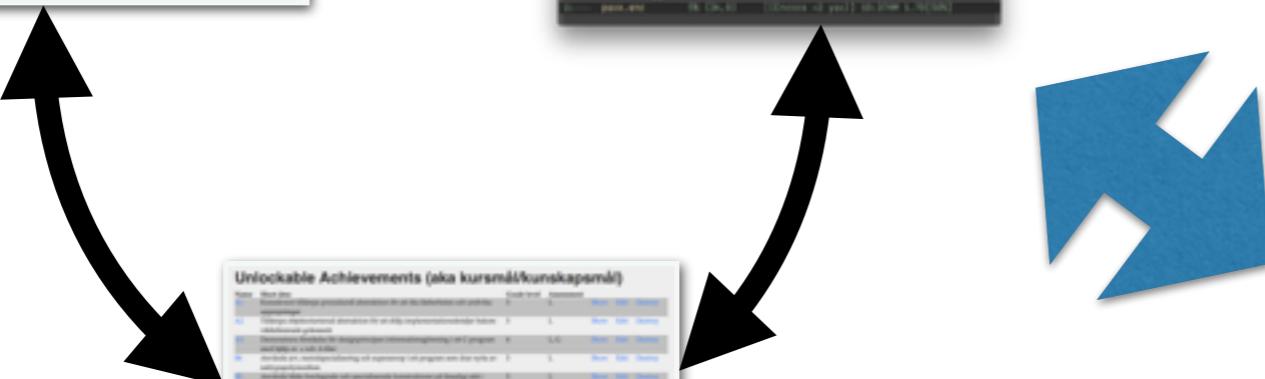
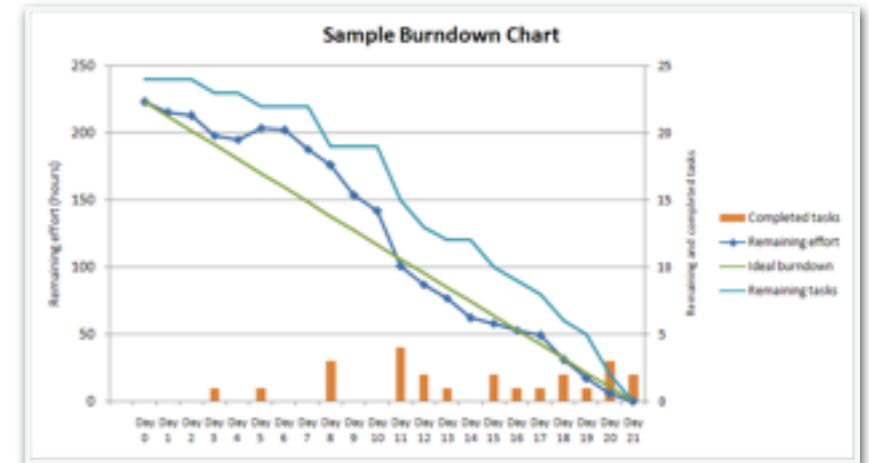
Implementera

```
embed void
    this->nodes = callc(k, sizeof(node_t));
end

def insert(vint) : bool {
    embed void
    node_t **n = (node_t**) &this->root;
    while (*n)
        {
            node_t *c = *n;
            if (c->value == v) { return false; }
            n = (c->value < v) ? &(c->left) : &(c->right);
        }
    ((node_t*)this->nodes)[this->size++] = (node_t) {
        .value = v;
        .n = &((node_t*)this->nodes)[this->size-1];
    };
    return true;
}

def reset() : void
{
    print("");
}
```

För bok över dina framsteg



A screenshot of an achievement list titled "Unlockable Achievements (aka kursmål/kunskapsmål)". It lists various milestones such as "Completed first assignment", "Completed final assignment", "Completed first project", "Completed final project", and many others related to programming and system design. Each achievement has a progress bar indicating completion status.

Välj mål



Redovisa



Uppföljningsmöte

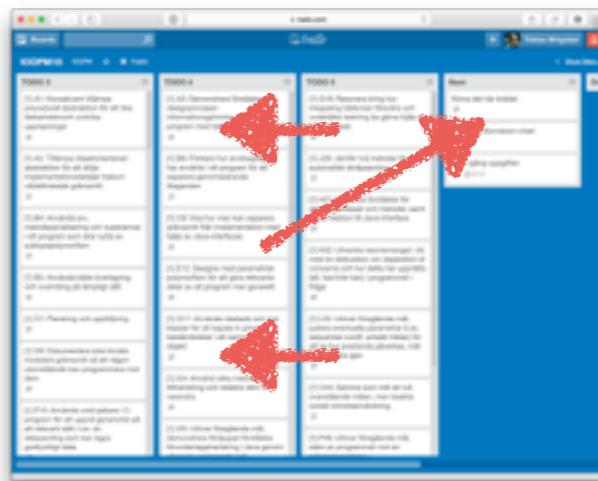
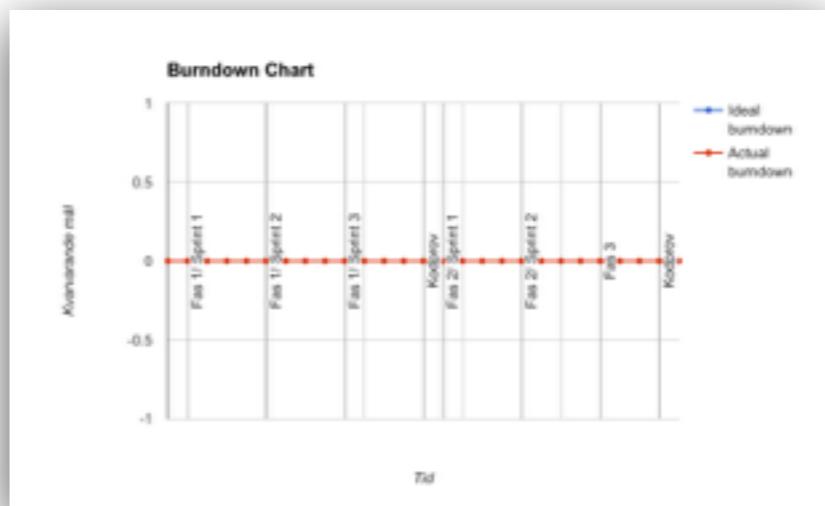
- I mitten av varje sprint

Diskussion med assistent
utifrån burndown chart

Tillsammans med tre
andra studenter

I början av kursen är det
fokus på planering

Vi går mot fokus på
uppföljning



Uppgifter | Imperativ och objektorienterad... Faser | Imperativ och objektorienterad... UpScale | Trello Creating cards by email - Trello Help +

Boards Trello + Tobias Wrigstad

UpScale Uppsala Programming Languages Org Visible

Doing

- Integration with new PonyRT
- Implement dependencies on tasks
- Basic data structures
- Module system
- Add a card...

Next

- Traits
25 2/4
- Parametric polymorphism
18 0/2
- Resolving deadlocks due to self-fulfilled futures
 0/3 AN
- Implement Par data structure.
10
- Finish the IMDB top 256 CS flicks
AN
- Block GC during suspension

Someday

- Backend TODOs
1 0/6
- GC TODOs
1 0/3
- Closure TODOs
 0/2
- Testing TODOs
 0/4
- AST TODOs
1 0/2 AN
- Alpha conversion for type variables
3
- Functional layer
1

Feature Requests

- Design high-level language for matrices.
1
- Maybe datatype
10
- Parallel loops
1
- Lazy Futures
8
- Annotation support
1
- Design (and implement) error handling for Encore
5
- Fields as sets
3 0/7

Add a card...



Uppgifter | Imperativ och objektorienterad... Faser | Imperativ och objektorienterad... Futures: improvements on UpScale... Creating cards by email - Trello Help +

Boards UpScale Doing Implement dependencies Basic data structures Module system Add a card...

Futures: improvements in list [Next](#) [Edit](#)

Members: AN +

Description [Edit](#)
Subsumes the following (archived) cards:

- [Await & Suspend](#)
- [Future chaining](#)
- [Futures](#)
- [Coroutines](#)
- [Suspendable/blocking actors \(was Futures etc.\)](#)

Checklist [Delete...](#)

0%

- Merge "children" and "responsibility" in the future struct
- Trace functions for futures and future type struct
- Remove stupid limitations (like 16 responsibilities max) on futures
- Add comprehensive testing for non-deterministic behaviour on futures, chaining, await and suspend

Add an item...

Activity

 Write a comment...

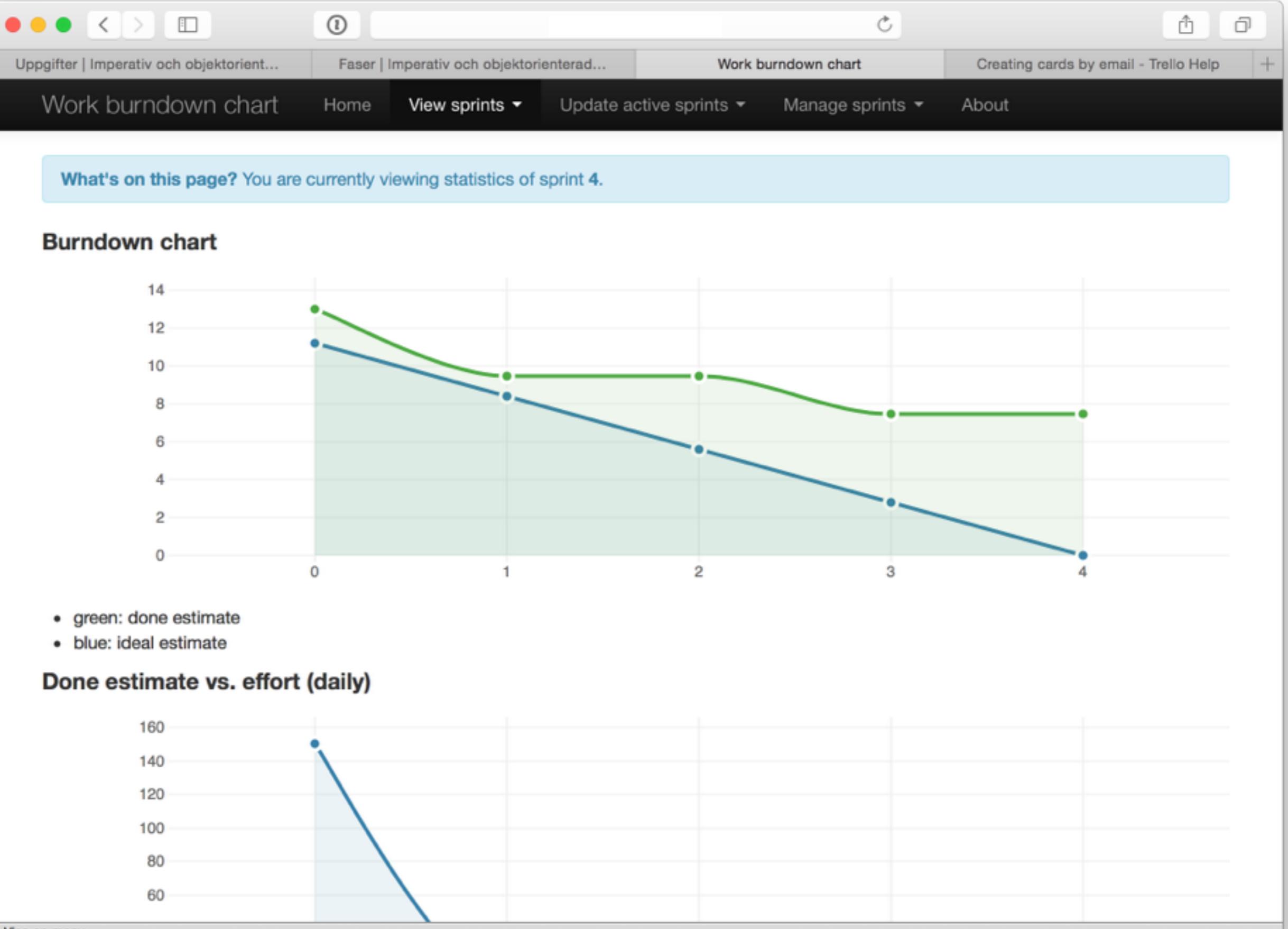
Add

- Members
- Labels
- Checklist
- Due Date
- Attachment

Actions

- Move
- Copy
- Subscribe
- Archive

[Share and more...](#)



Visa en meny

<https://github.com/devtyr/trello-burndown>

docs.google.com

tobias.wrigstad@gmail.com

Comments Share

Burndown chart

File Edit View Insert Format Data Tools Add-ons Help All changes saved in Drive

B C D E F G H I J K

1

2 Hur många mål siktar du på att ta denna sprint? 7

3

4

5 Hur många mål har du tagit?

	Lab 2	1
	Lab 3	0
	Lab 4	3
	Lab 5	3

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

Burndown Chart (Fas 1 / Sprint 1)

Ideal velocit

Actual velocit

Kvarvarande mål

Tid

The chart displays two lines: a blue line for 'Ideal velocit' and a red line for 'Actual velocit'. The y-axis represents 'Kvarvarande mål' (Remaining tasks) from 0 to 8. The x-axis represents 'Tid' (Time). The ideal velocity is a straight line starting at approximately 7.2 and ending at 0. The actual velocity starts at 7.2, dips slightly, rises to a peak of about 6.2, and then decreases more sharply than the ideal line, reaching 0 by the end of the sprint.

Burndown totalt Fas1/Sprint1 Fas1/Sprint2 Fas1/Sprint3 Fas2/Sprint1 Fas2/Sprint2 Fas3 Burndown

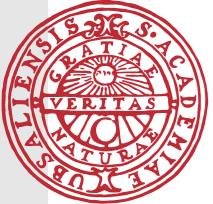
Visa en meny

Se "Länkar" på kurswebben

Högskolepoäng

	HP	Deadline
Fas 1	5	13/10*
Fas 2	5	8/12
Fas 3	5	10/1
Kodprov	2,5	17/10
	2,5	21/12

*It's complicated (se kurswebben för detaljer)



Övning i skriftlig färdighet

- På nivå 4 och 5 måste du redovisa ett mål som en essä
(För att nå nivå 3 räcker det med projektrapporten)
- Instruktioner finns på kurswebben

Omfattning: 7500 tecken

Deadline: 5/12

- Lämnas in via GitHub

ng till handledning online (t.ex. via epost)	11	13.4%
Återkoppling/feedback på inlämningar	24	29.3%
(utveckla gärna i kommentarerna nedan)	2	2.4%

Jämförelse mellan två skräpsamlingsalgoritmer

Mark and Sweep mot Reference counting

Uppsala universitet
January 16, 2015

Två skräpsamlingsmetoder

Något som blir vanligare och vanligare är användningen av skräpsamlare, även känd som garbage collectors. En skräpsamlares främsta uppgift är att lämna tillbaka minne du inte använder. Så den frigör minne som du använt och inte använder längre, på ett automatiskt sätt. Denna process brukar kallas för skräpsamling. Anledningen till användningen av skräpsamlare har blivit stort skulle kunna beröra på språk såsom Java, JavaScript, Python som alla använder sig av någon typ av skräpsamling. Varför beslutade sig folk för att använda skräpsamlare? Förmodligen för att det är svårt att hantera minnet manuellt. Om inte programmeraren har skickligheten som krävs och är på sin väkt hela tiden kan det uppstå minnesstöcker eller andra problem vid manuell hantering av minne. Med en skräpsamlare behöver inte programmeraren occupa sig över sådana saker och kan ägna mer tid till andra saker.

Jag är ganska övertygad att du någon gång kommer använda dig av någon skräpsamlare om du ångar dig åt programering. Hur mycket du tänker på det eller inte är en annan fråga. Jag kommer beskriva hur två metoder för skräpsamling går till och göra en jämförelse mellan dem.

I detta dokument ska jag beskriva två vändiga algoritmer för skräpsamling och deras skillnader. Metoderna jag kommer fokussera på är Mark and Sweep och Reference counting.

Om ett objekts referensräknare sätter till si finns det inte längre några referenser till det objekten, Objekten är därmed skräp och kommer att frigörs direkt när referensräknaren skräpsamlar. Detta är däremot deterministiskt², vilket innebär att vi vet exakt när ett objekt tas bort. Detta är en av de stora fördelarna med referensräkning. Detta innebär att referensräkning är kompatibel med RAFTLE och kan däremot användas tillsammans med destruktören, kod som körs automatiskt när objekten lämnas.

En annan fördel med referensräkning är att arbetet för att ta reda på vad som är skräp är fördelat över hela programmet (kördet), vilket inte är fallet med spårande skräpsamlare. Detta är särskilt bra av utgående med referensräkning eftersom det kräver en liten del extra arbete borta för att uppdatera objekters referensräknare.³ Men ju före en referens räknas eller ändras en enda objekts lämna upp i minnet och dess referensräknare uppdateras. Det kan medföra stora misstag eftersom objekten lämnas upp i codeminnet, vilket påverkar positionen negativt.

Det kanske största problemet med referensräkande skräpsamlare är att några implementeringar inte kan hantera cirkulära strukturer. Cirkulära strukturer innehåller objekt som direkt eller indirekt refererar till sig självt. Några implementeringar klarar detta genom att använda svaga referenser som inte enbart inte klar objekts referensräkningen. Det finns också mer komplexa algoritmer som kan hantera detta, men dessa kräver ofta stora misstag extra arbete.

2.2 Mark and sweep

Mark and sweep tillhör instängen av tracing eller spårande skräpsamlare. Denna typ av skräpsamling angriper problemet från Mark-and-Sweep. Istället för att hålla reda på och frigöra skräp direkt när upptäckt är kommunikationen via vissa trappsteg. Ofta startar detta när maningen levtid minne tar slut eller faller under en viss nivå.

Mark and sweep har två steg som här kallas för "mark" och "sweep". Första stegen mark, gör ut på att hitta och markera alla levande objekt. Själva markeringen sätter genomsatt en flagga som sparar tillsammans med varje objekts sätta. Itan "mark" steget påbörjas tilldelas samtliga flaggar.

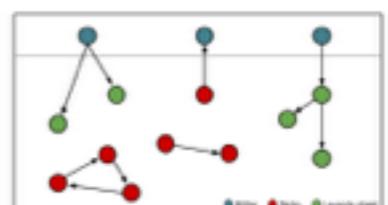


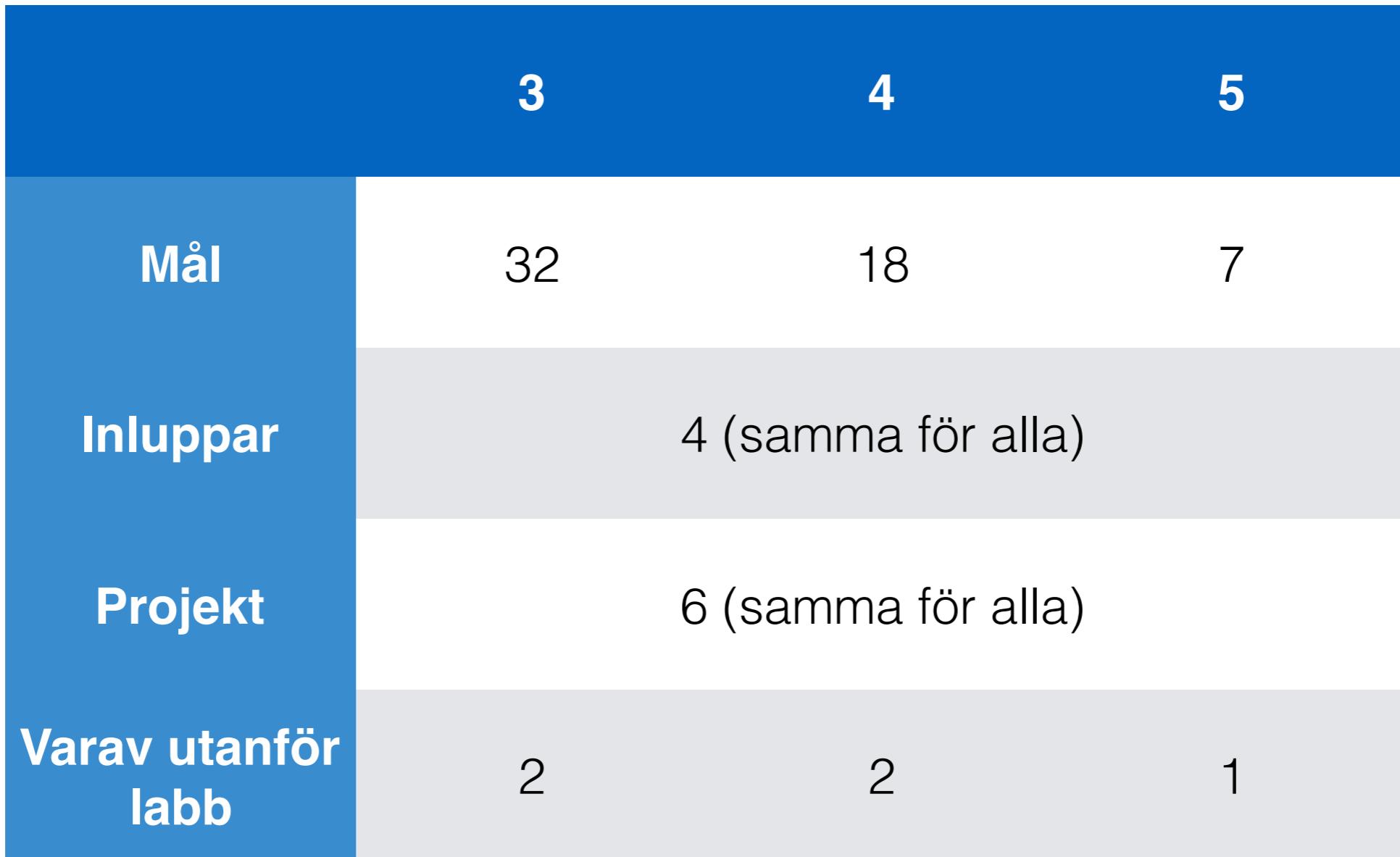
Figure 1: Objekter representerar objekt och pilarna referenser.

För att hitta alla levande objekt är unga skräpsamlare från ett antal rötter. Rötterna är alda referenser eller pelare som vi vet inte vi har tillträffat till. I programmet är just dessa varia alda pelare som taggs på staden. Skräpsamlaren går över rötterna och färgar dessa referenser. Varje objekt som hittas markeras och för objekts senliga referenser är upprepat numera process som för rötterna. Alltså vi färgar referenser och alla objekt som hittas markeras och dessa referenser färgas. Denna visualiseras i figuren.

²Detta är inte helt sant, då i faktiskt program kan uppdateringen av referensräkningen kan hända till och med.



Betyg



Fas 3: Projektarbete

- Arbeta 4–6 personer (vi tar fram grupperna under november)
- Uppgiften TBA (blir variant av tidigare år)

Rekommenderad start: 1/12

Klart: 10/1

- Lämnas in via GitHub
- Presentation och verkstad med annan grupp
- Grupperna lägger själva upp sprintar
- 1–2 KLOC, plus tester

3

Uppgiften

OBSERVERA
Denna del av specificeringen är ett
livslångt dokument som kan kom-
ma att uppdateras och förändras
under projektets gång.

¹T.ex. med hjälp av metod i
mstlib.h, eller snap i sys/mem.h.

²Vi giv en förskräck och utgår från
att programmen är enkeltredade
och att endast en heap skapas per
program.

3.1 Skriptsamling med mark-sweep

Skriptsamling med mark-sweep vandrar genom (traverserar) den graf
som heapen utgör för att identifiera objekt som säkert kan destrukteras
utan att programmet kraschs. Vi går igenom algoritmen steg-för-steg
nedan.

Vi kan tänka om att varje objekt innehåller en extra bit³, den s.k.
mark-biten. När denna bit är satt (1) anses objektet vara "vid liv".
Annars är objektet skrål som kan tas bort.

Vid skriptsamling sker följande (logiskt sett):

Steg 1 Iterera över samtliga objekt på heapen och sätt mark-biten till
0. Detta innebär att alla objekt anses vara skrål initialt.

Steg 2 Sök igenom stacken eller pekare till objekt på heapen⁴, och med
utgångspunkt från dessa objekt, traversera heapen och markera
alla objekt som påträffats genom att mark-biten sätts till 1.

Steg 3 Iterera över lista över samtliga objekt på heapen och frigör alla
objekt vars mark-bit fortfarande är 0.

Steg 2 kallas för "mark-fasen" och steg 3 för "sweep-fasen", härav
algoritmens namn, mark-sweep.

³Tekniskt kan det också vara en bit
som man har en över. Dådand kan man
packa in bitar i annat data – vi skall
se exempel på det senare i denna text!

⁴Dessa pekare kallas vi också för
"vötter".



Kodprovet (2x2,5 HP)

- Två frågor – en C, en Java
 - Individuellt prov i datorsal, 3 timmar – **ingen tillgång till Internet**
- Syftet: att tvinga alla att sitta i framsätet vid parprogrammering

Examinerar inte kursmål!

- Går att ta i steg (klara en fråga på varje prov)
- Delresultat giltiga i ett år
- Två provtillfällen under kursen

17/10 och 21/12

- Anmälan annonseras i Piazza



Simple [minimal sammanfattning]

1. Läs specifikationen och leta specifikt efter **verb** (funktioner/beteende), eller **substantiv** (data/objekt/strukturer) — gör en work breakdown structure
2. Skriv kod för att pröva om du tänkt rätt (vad är rätt – hur man kollar det?)
3. Ha alltid ett fungerande program
4. Kompilera efter varje förändring
5. Kör programmet hela tiden för att hitta fel (eller ännu bättre – kör testen!)
6. Dela upp alla problem i delproblem, gå till 7. först när något är enkelt
7. Dela upp alla delproblem i mindre steg – gör de enklaste först
8. **Fuska** (cheat) varje gång du riskerar att fastna
9. **Skarva** (dodge) för att förenkla specifikationer och skapa fler enklare delsteg
10. Växla mellan att: tänka, koda och ibland refaktorera (speciellt **fusk** och **skarvar**)



Simple

- Om du inte redan är en programmerare måste du använda SIMPLE under kursen

- Finns detaljerad beskrivning på kurswebben

Använder inlupp 1 som löpande exempel

En hel del gratiskod för inlupp 1 finns där

- Lektion 1 handlar om att tillämpa Simple på inlupp 1

Det kan vara svårt att ta in allt direkt, så börja med det som verkar enkelt

Försök inte göra rätt, utan det som känns rätt — gå tillbaka till texten när det behövs



Att använda en texteditor

- Under kursen kommer vi att använda **Emacs**

Under fas 2 är de tillåtet att använda IDE:er,
men inte rekommenderat

- Emacs kan också betyda vim men **inte**

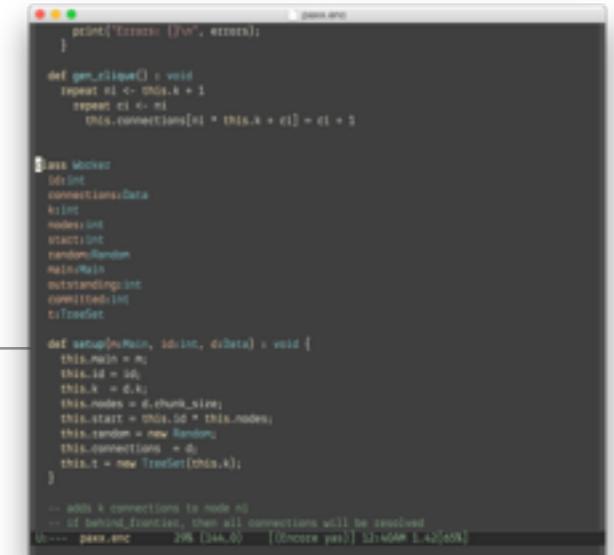
Gedit

Nano

Notepad++

Sublime

...



```
point("Error: (%s", errors);

}

def gen_ring() : void
repeat #i <- this.k + 1
repeat #j <- #i
  this.connections[#i * this.k + #j] = #j + 1

}

class Model
#int connections;
#int k;
#int nodes;
#int start;
#random Random;
#int max;
#int outstanding;
#int committed;
#TrieSet t;

def setup(#in max, #in start, #in data) : void {
  this.max = max;
  this.id = start;
  this.k = data;
  this.nodes = data.chunk_size;
  this.start = max / data * this.nodes;
  this.random = new Random();
  this.connections = max;
  this.t = new TrieSet(this.k);
}

-- adds k connections to node n0
-- if behind frontier, then all connections will be resolved
Unit-- pass.emc 296 (344.00) (Emacs pass) 51464m 1.42/63k
```

