

IOPM 2015

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 hjälpmedel.
- **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 examinator 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



Från funktionell till imperativ programmering



```
module Main where

fib 0 = 0
fib 1 = 1
fib n = fib (n - 1) + fib (n - 2)

main = print (fib 15)
```

```
$ ghc fib.hs
$ ./fib
610
$ _
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int acc1, acc2, temp;
    acc1 = 0;
    acc2 = 1;

    const int n = 15;

    for (int i = 0; i < n; ++i)
    {
        temp = acc2;
        acc2 += acc1;
        acc1 = temp;
    }

    printf("fib(%d) = %d\n", n, acc1);

    return 0;
}
```

```
$ gcc -std=c11 -o fib fib.c
$ ./fib
fib(15) = 610
$ _
```



```
#include<stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello, world!");
    return 0;
}
```

hello.c

```
$ ./hello
Hello, world!
$ _
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        puts("Usage: ./greeter <name>");
    }
    else
    {
        printf("Hello, %s!\n", argv[1]);
    }
    return 0;
}
```

greeter.c

```
$ ./greeter
Usage: ./greeter <name>
$ ./greeter Tobias
Hello, Tobias
$ _
```



```
#include <stdio.h>

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        puts("Usage: ./mycat <name>");
    }
    else
    {
        FILE *f = fopen(argv[1], "r");
        int c = 0;
        while ((c = fgetc(f)) != EOF)
        {
            putchar(c);
        }
    }
    return 0;
}
```

mycat.c

```
$ ./mycat hello.c
#include<stdio.h>

int main(int argc, char *argv[])
{
    puts("Hello, world!");
    return 0;
}
$ _
```



```
#include <stdio.h>
#include <string.h>

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        puts("Usage: ./mygrep <pattern> <name>");
    }
    else
    {
        FILE *f = fopen(argv[2], "r");
        int str_len;
        size_t buf_siz;
        char *string;
        char *pattern = argv[1];

        while ((str_len = getline(&string, &buf_siz, f)) > 0)
        {
            if (strstr(string, pattern, str_len))
                printf("%s", string);
        }

        fclose(f);
    }
    return 0;
}
```

mygrep.c



F2: Introduktion till C

- Vi skall gå igenom C från grunden med start idag kl. 13:15

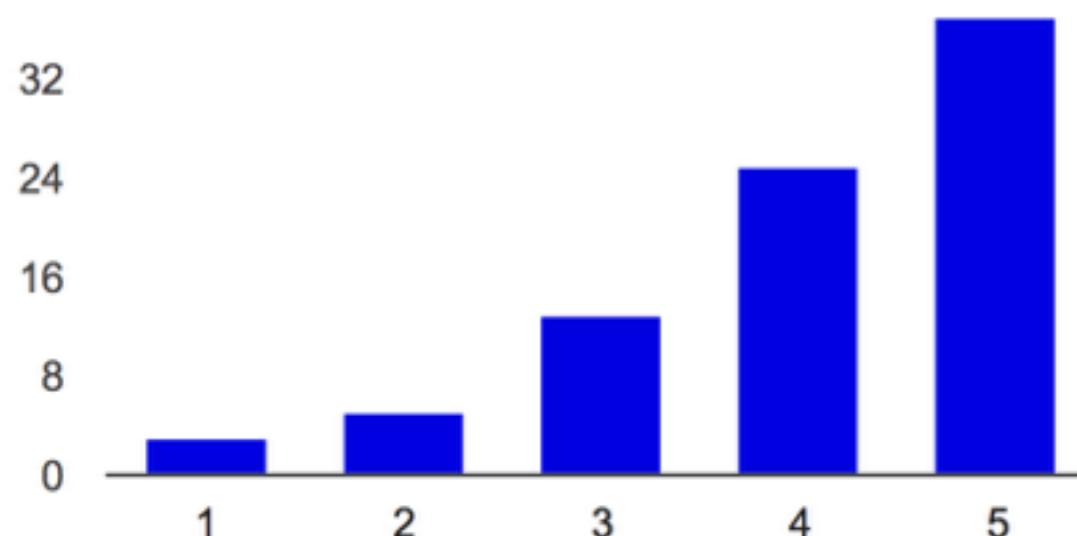


Resultatet av undersökningen i våras [83 svar!]

- Tar bara upp ett utsnitt
- **TACK**



Inspelade föreläsningar kan i stor utsträckning ersätta vanliga föreläsningar

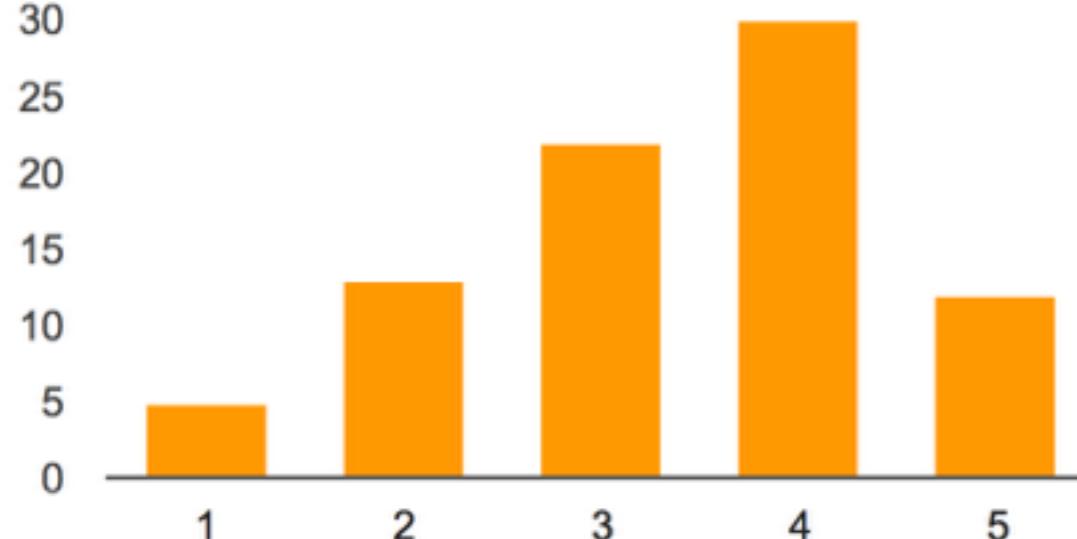


instämmer inte alls:	1	3	3.6%
	2	5	6%
	3	13	15.7%
	4	25	30.1%
instämmer fullständigt:	5	37	44.6%

Screencasts

Enstaka "flipping"

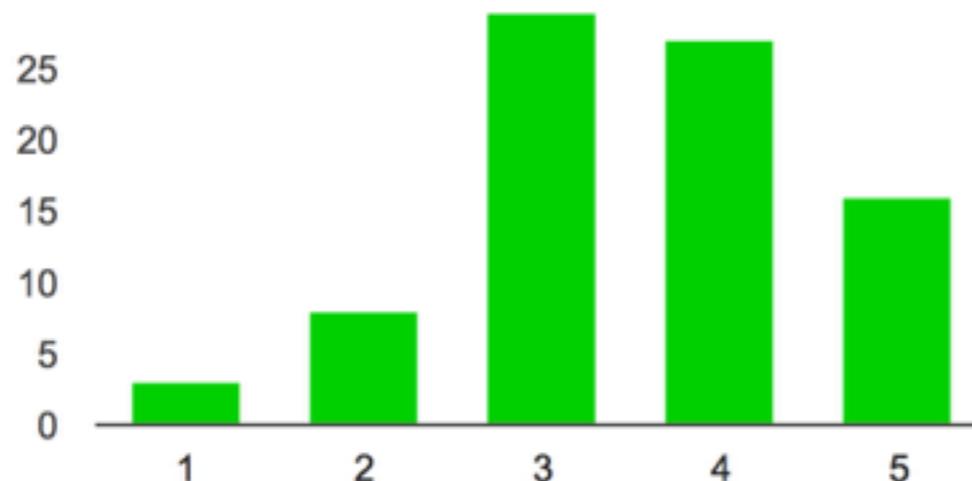
Vanliga föreläsningar är ett bra sätt att förmedla kunskap till mig som student



instämmer inte alls:	1	5	6.1%
	2	13	15.9%
	3	22	26.8%
	4	30	36.6%
instämmer fullständigt:	5	12	14.6%

Många FL kvar

Vanliga föreläsningar bör ersättas med moment där jag får arbeta aktivt med materialet, inte lyssna passivt



instämmer inte alls: 1 **3** 3.6%

 2 **8** 9.6%

 3 **29** 34.9%

 4 **27** 32.5%

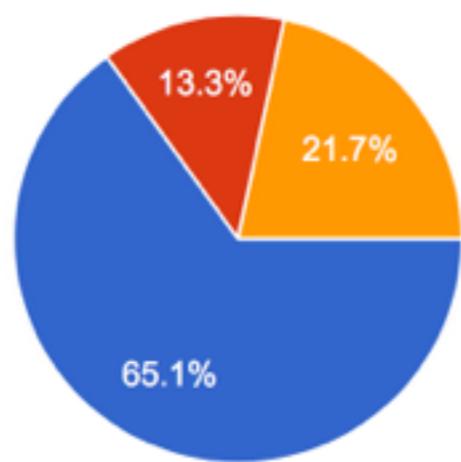
instämmer fullständigt: 5 **16** 19.3%

27 labbar

5 lektioner

Enstaka flippade FL

Live-kodning är ett bra redskap för att lära ut programmering



Ja **54** 65.1%

Nej **11** 13.3%

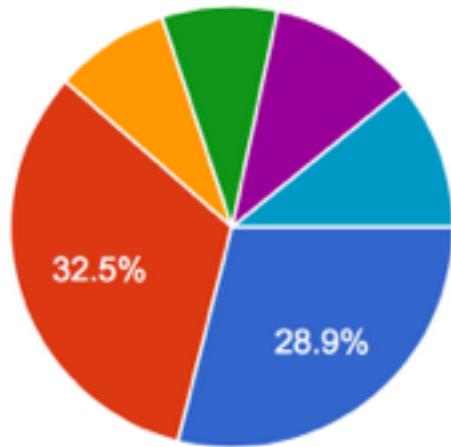
Vet inte/varken bra eller dåligt **18** 21.7%

Livekodning

26 föreläsningar

3 gästföreläsare

Vilken del av kurserna skall vi prioritera högst?

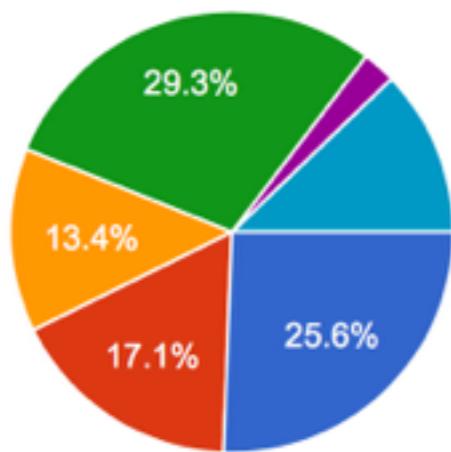


4 seniora assar

9 juniora assar

Föreläsningar	24	28.9%
Tillgång till handledning	27	32.5%
Tillgång till handledning online (t.ex. via epost)	7	8.4%
Återkoppling/feedback på inlämningar	7	8.4%
Annat (utveckla gärna i kommentarerna nedan)	9	10.8%
Vet inte/vill ej svara	9	10.8%

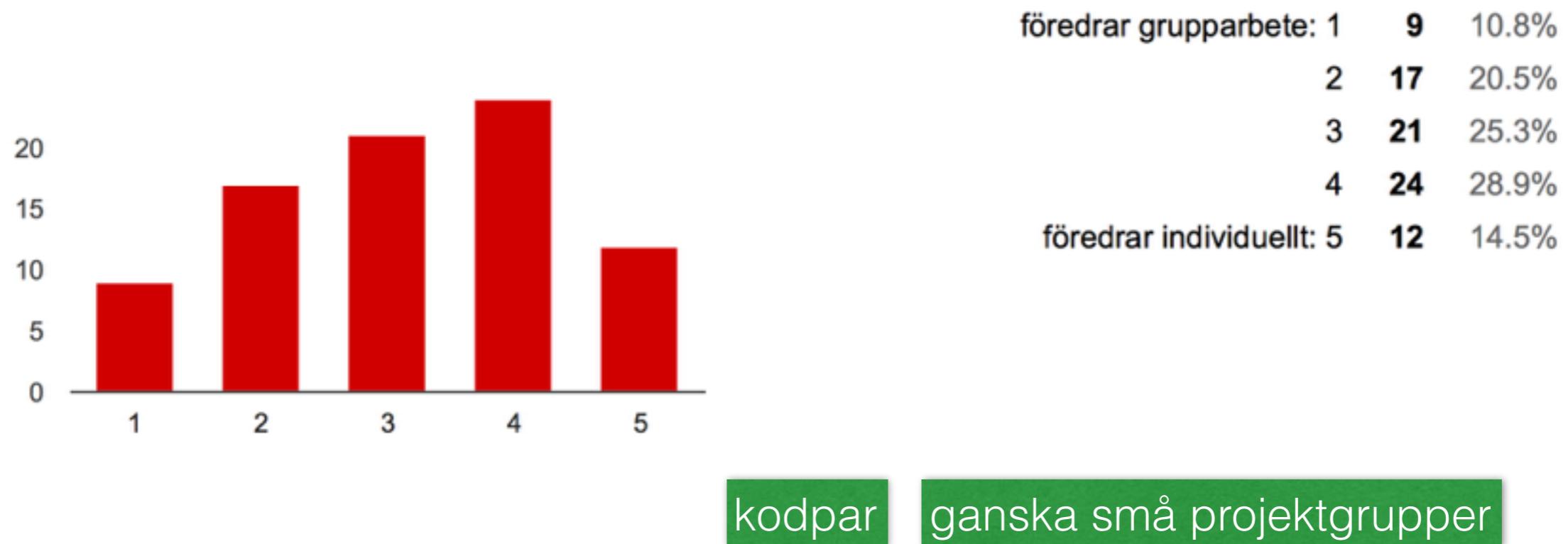
Vilken del av kurserna skall vi prioritera näst högst?



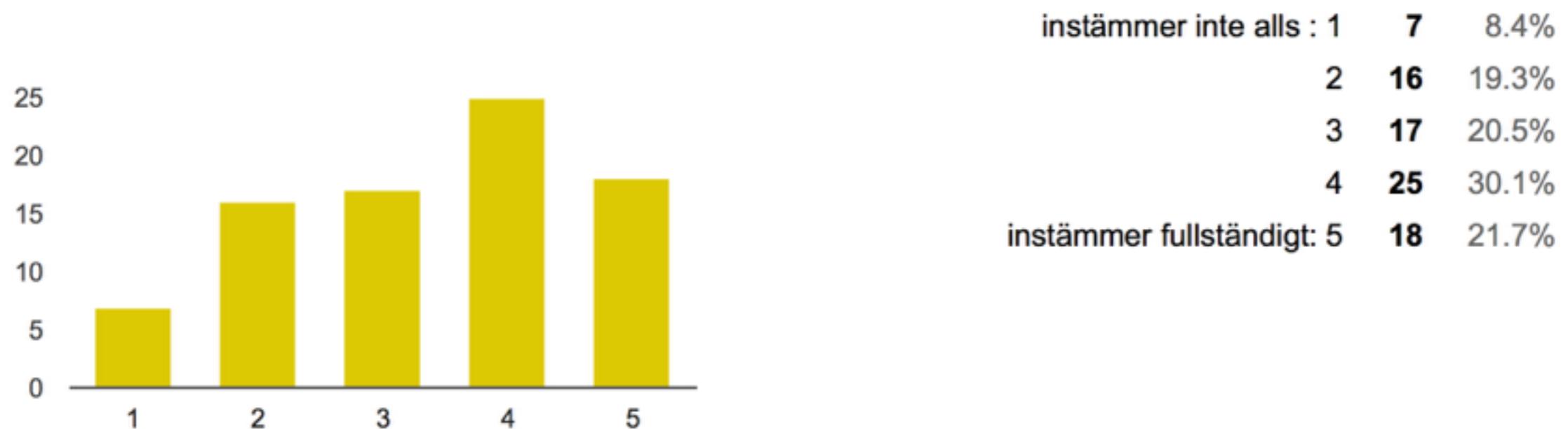
Föreläsningar	21	25.6%
Tillgång till handledning	14	17.1%
Tillgång till handledning online (t.ex. via epost)	11	13.4%
Återkoppling/feedback på inlämningar	24	29.3%
Annat (utveckla gärna i kommentarerna nedan)	2	2.4%
Vet inte/vill ej svara	10	12.2%

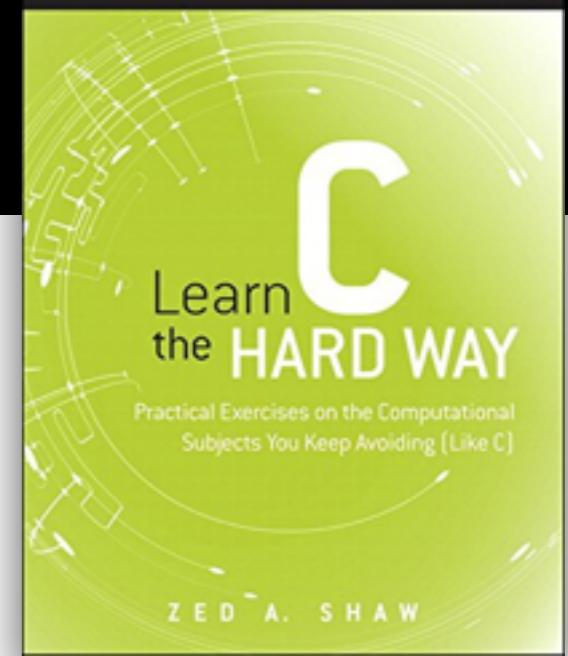
140 timmar HL/återkoppling

Jag föredrar att arbeta individuellt framför att arbeta i grupp



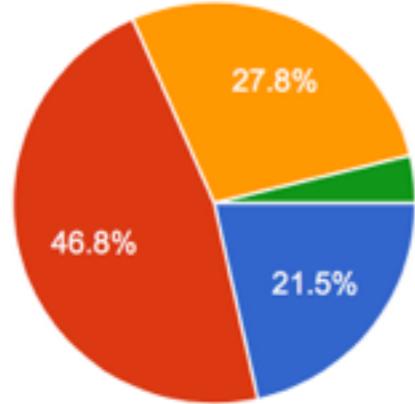
Jag lär mig mer programmering om jag får arbeta tillsammans med någon annan än om jag får arbeta själv





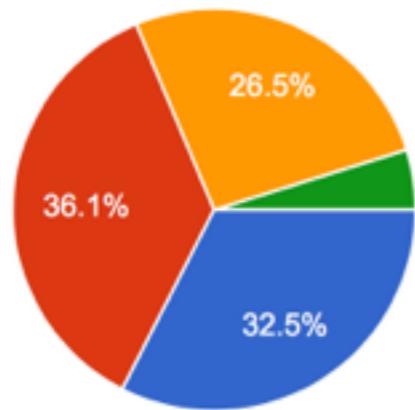
1 bok (C) á 250 kr

Vad tycker du är en rimlig summa att lägga på kursböcker på en kurs på 20 HP?



1 = 0 kr	17	21.5%
2 = upp till 500 kr	37	46.8%
3 = upp till 1000 kr	22	27.8%
4 = mer än 1000 kr	3	3.8%

I vilken utsträckning brukar du följa med i kurslitteraturen under kursens gång?



1 = läser oftast inte kurslitteraturen	27	32.5%
2 = sällan eller aldrig, läser mest inför tentan	30	36.1%
3 = följer oftast med om kapitelhänvisningar är givna	22	26.5%
4 = följar oftast med, även om kapitelhänvisningar inte är givna	4	4.8%

Anvisningar givna, praktisk bok

Supertydligt vad som gäller från dag 1 – med allt.

Supertydligt var information finns!

Supertydligt vem man skall fråga om vad – och hur!

Onlineföreläsningar!

Förutsätt att vi **inte** kan annat än vad som täcks av ÅK 1.

Många labbassar!

Många labbar!

Så lite väntetid som möjligt på labbar!

Förtroende för oss studenter!

Mer feedback!

Struktur.

Morgan Freeman som föreläsare!

Nytt för i år

- Genomgående tillämpning av metodiken SIMPLE
- Mer longitudinella uppgifter, alla börjar med samma uppgift
- Hjälp med mappning från uppgifter till mål (i sjunkande grad)
- Fler traditionella föreläsningar med grundläggande innehåll (C, Java)
- Lektioner (tröskla omsättning av teori i praktik)
- Tydligare koppling till lättrörliga processer även i hur kursen fungerar

Indelning i sprintar med planering, uppföljning

Tvång på att visualisera framsteg

- Uppföljningsmöten i smågrupper
- Kursråd
- Allt arbete sker på GitHub under hela kursen



Var finns information om kursen?

The screenshot shows the IOOPM 2015 homepage with a yellow header. Below it, there's a section titled "IMPERATIV OCH SÄ VIDARE" with a sub-section "NYHETER". A yellow box highlights the URL "wrigstad.com/ioopm".

Hur interagerar jag med er?

The screenshot shows the Piazza platform interface. It displays a list of posts and a "Class at a Glance" summary. A yellow box highlights a message from a professor.

Professor Message:
Last Fall, we launched a new service called Piazza Careers. And through this service, we have connected students with other students studying same subjects, with recent alumnae in industry, and with potential employers. Many of them have successfully secured internships and jobs through our service! Just as Piazza was born out of my desire myself struggle to get help when I was stuck on a very difficult problem, I am excited to see what challenges our students will face in their careers. Piazza Careers is designed to help our students tackle challenges faced greater with what to do with why we live, helping enhance their experience in their chosen field of study. We have a vision for Piazza Careers, and like with Piazza Q&A you can't get there without your feedback. We are eager to make a real impact on students' lives, helping enhance their experience in their chosen field of study. It will require a lot of iteration and hard work. We'll inevitably make a few mistakes along the way and we ask for your support and forgiveness in those moments. The Piazza Careers service is essentially a third tier that students can choose to access when desired. Students can remove this additional tab at any time. Our foremost priority is to protect student privacy and ensure they remain anonymous from the world.

Hur lämnar jag in en uppgift?

The screenshot shows a GitHub repository page for "IOOPM-UU / ioopm15". It lists several commits and pull requests. A yellow box highlights the repository name "ioopm15".

Hur redovisar jag?

The screenshot shows the AU Portal interface with a section titled "Unlockable Achievements (aka kursmål/kunskapsmål)". It lists various achievements with descriptions and status. A yellow box highlights the section title.

Name	Description	Grade level	Assessment
A1	Konsekvent tillämpa procedurell abstraktion för att öka läsbarheten och smidiga uppreningar	3	L
A2	Tillämpa objektorienterad abstraktion för att dölja implementationsdetaljer beroende väldefinierade gränssnitt	3	L
A3	Demonstrera förmåga för designprincipens informationsglänsning i ett C-program, med hjälp av c och h-filer	4	L, G
A4	Använda arb, metodöverladdning och supersköring i ett program som drar nytta av syntypolymerism	3	L
A5	Använda både överskrider och specialiserade konstruktörer på lämpligt sätt i programmering	3	L
A6	Förklara hur användningsgrupp har använts i ett program till att separata gränssnittslagraden	4	G, T
A7	Förklara begreppet designmodeller samt visa hur man till ett designmodell har använt i ett icke-trivialt program	4	G
A8	Visa hur man kan separa gränssnitt från implementation med hjälp av Java-interfaces	4	L, G
A9	Dokumentera lika trivsamma modeller givetvis så att någon utomstående kan prata	3	L
A10	Använda viss pekare relevant sätt, t.ex. av	3	
A11	Använda parametriskt vid interaktion med Java	3	
A12	Designa med parametrar av ett program men så att det	3	
A13	Översätta mellan rekursiva och iterativa lösningar av samma problem samt diskutera till- och nackdelar med olika tekniker	3	L



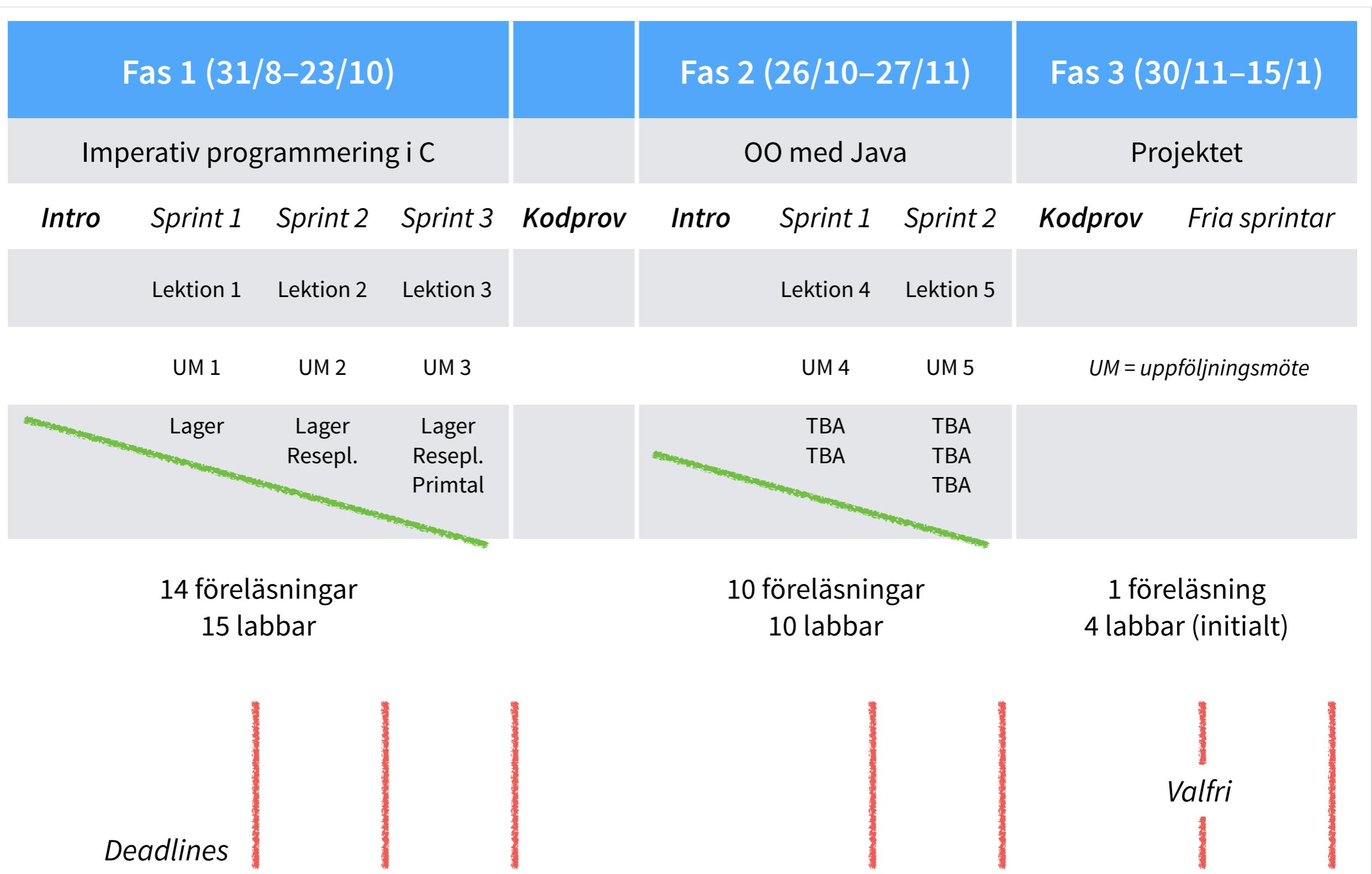
Du börjar här:

The screenshot shows a web browser window with the URL 127.0.0.1. The page has a yellow-to-white gradient background featuring a faint silhouette of a city skyline at the bottom. At the top center, the text "IOOPM 2015" is displayed in large, bold, black letters, with "IMPERATIV ... OCH SÄ VIDARE" in smaller letters below it. A navigation bar below the title contains links: HOMEPAGE (highlighted in dark grey), DEADLINES, OM KURSEN, PROCESS, MÅL, UPPGIFTER, LÄNKAR, KONTAKT, and HJÄLP!. On the left side, under "KURSDELAR", there is a section for "FAS 01" with the text "Introduktion till imperativ programmering i C. 31:a augusti till 23:e oktober." Below this, a "Visa en meny" link is visible. In the center, a large section titled "IMPERATIV OCH OBJEKTORIENTERAD PROGRAMMERINGS-METODIK" is prominently displayed. At the bottom of this central section, the text "Välkommen till sidorna för kursen IOOPM -- imperativ och objektorienterad" is shown. On the right side, a "NYHETER" section lists "[2015-08-22] Denna sida online" with a checkmark, and a "GAMLA NYHETER" button.

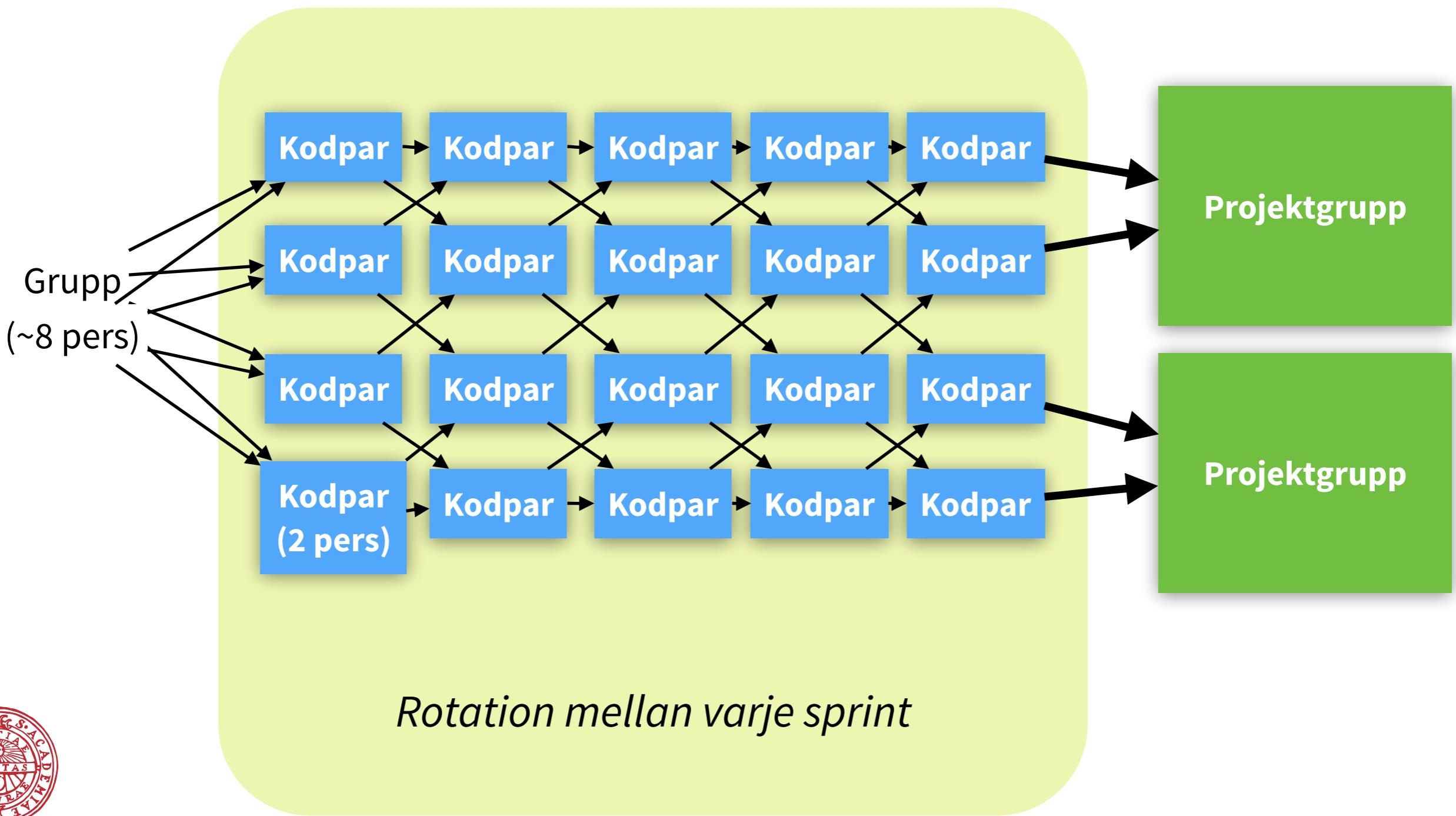
wrigstad.com/ioopm



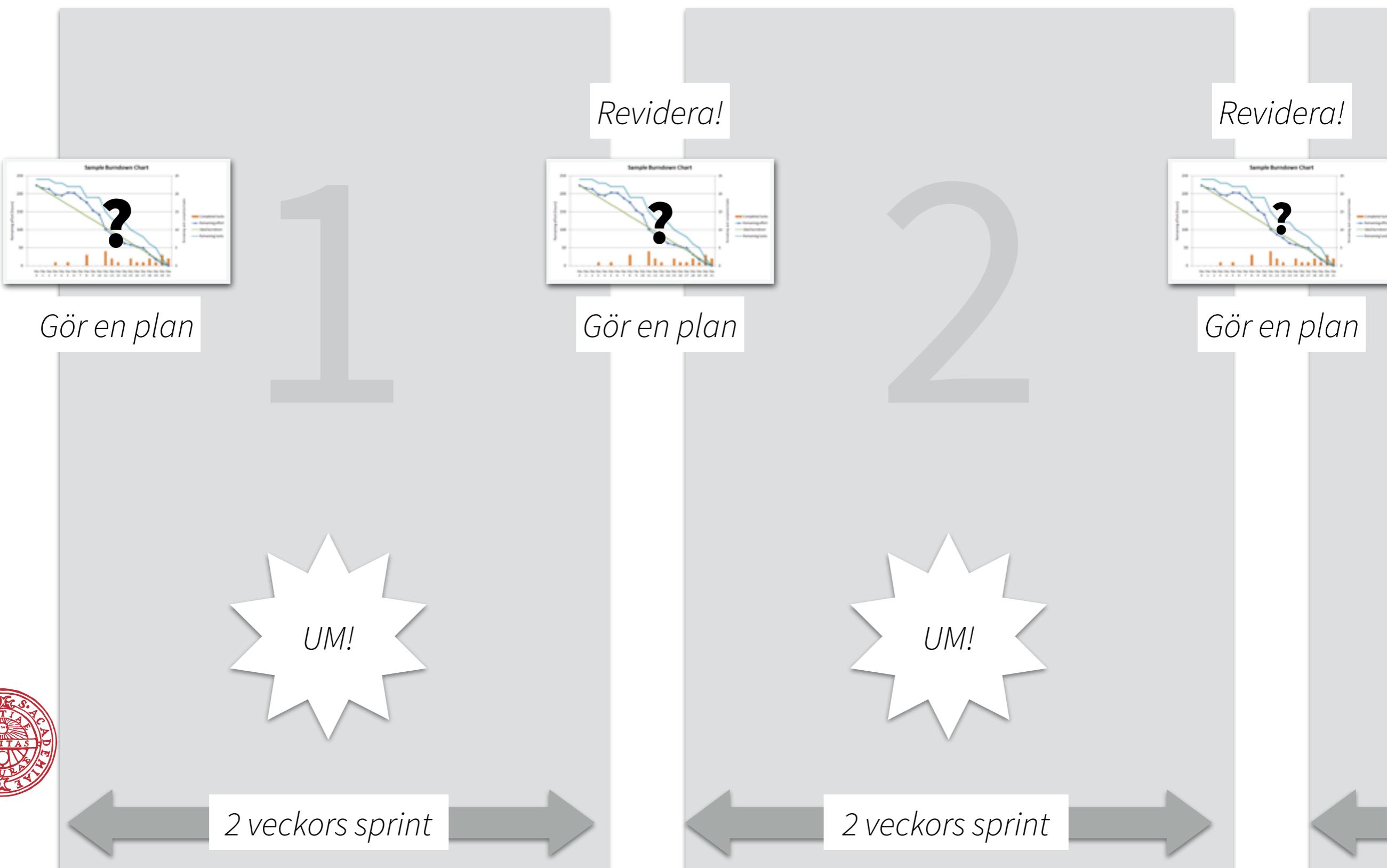
Översikt



Grupper och kodpar [rotationerna sköter ni själva]



Fas [kursen har 3]



Sprint [varje fas har minst 2]

Välj uppgift



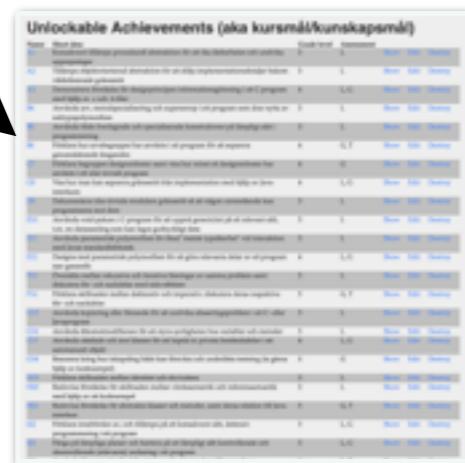
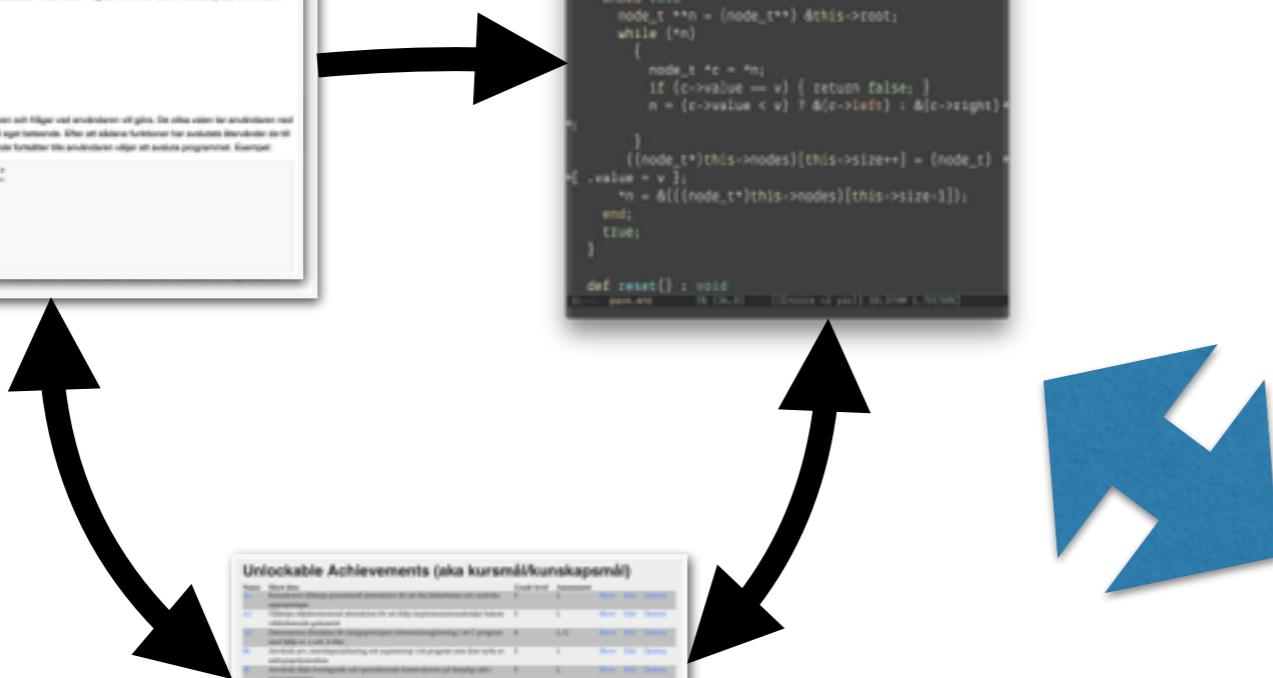
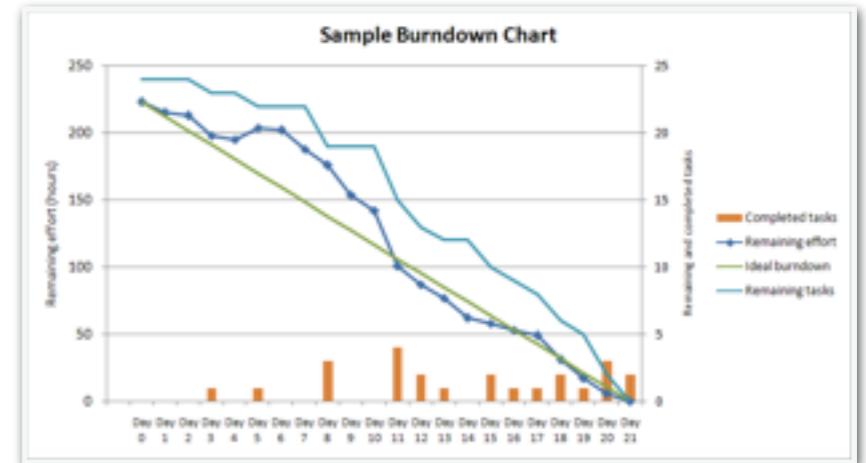
Implementera

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

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

def reset() : void
end
```

För bok över dina framsteg



Välj mål



Redovisa



Uppföljningsmöte

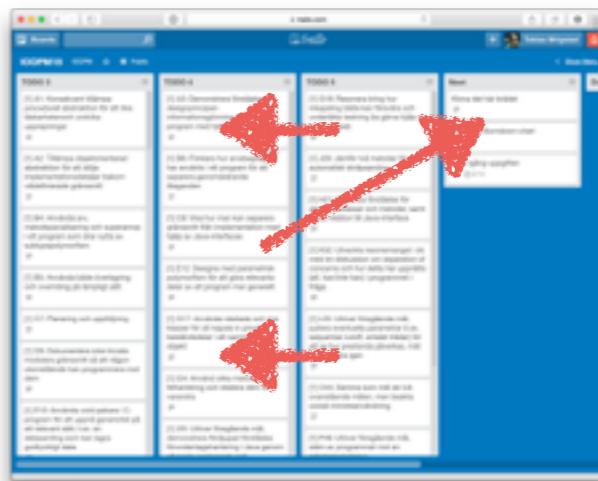
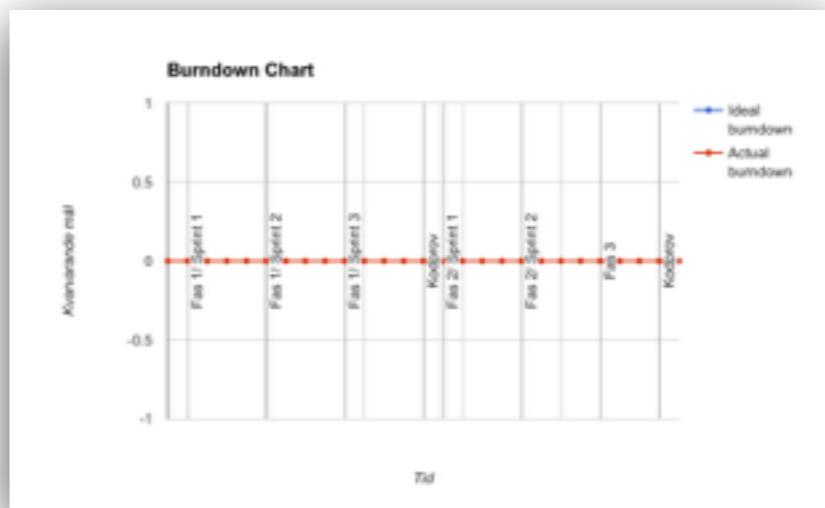
- I mitten av varje sprint

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 Show Menu

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
- 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
- Design (and implement) error handling for Encore
5
- Fields as sets
3 0/7



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

Boards

UpScale Up

Doing

- Integration with new system 2 comments
- Implement dependencies 1 comment
- Basic data structures 3 comments 4/5
- Module system 7/20

Add a card...

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

Members: AN, [+ Add Member](#)

Description [Edit](#)

Subsumes the following (archived) cards:

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

Checklist [Delete...](#) 0% [Move](#) [Copy](#) [Subscribe](#) [Archive](#)

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...

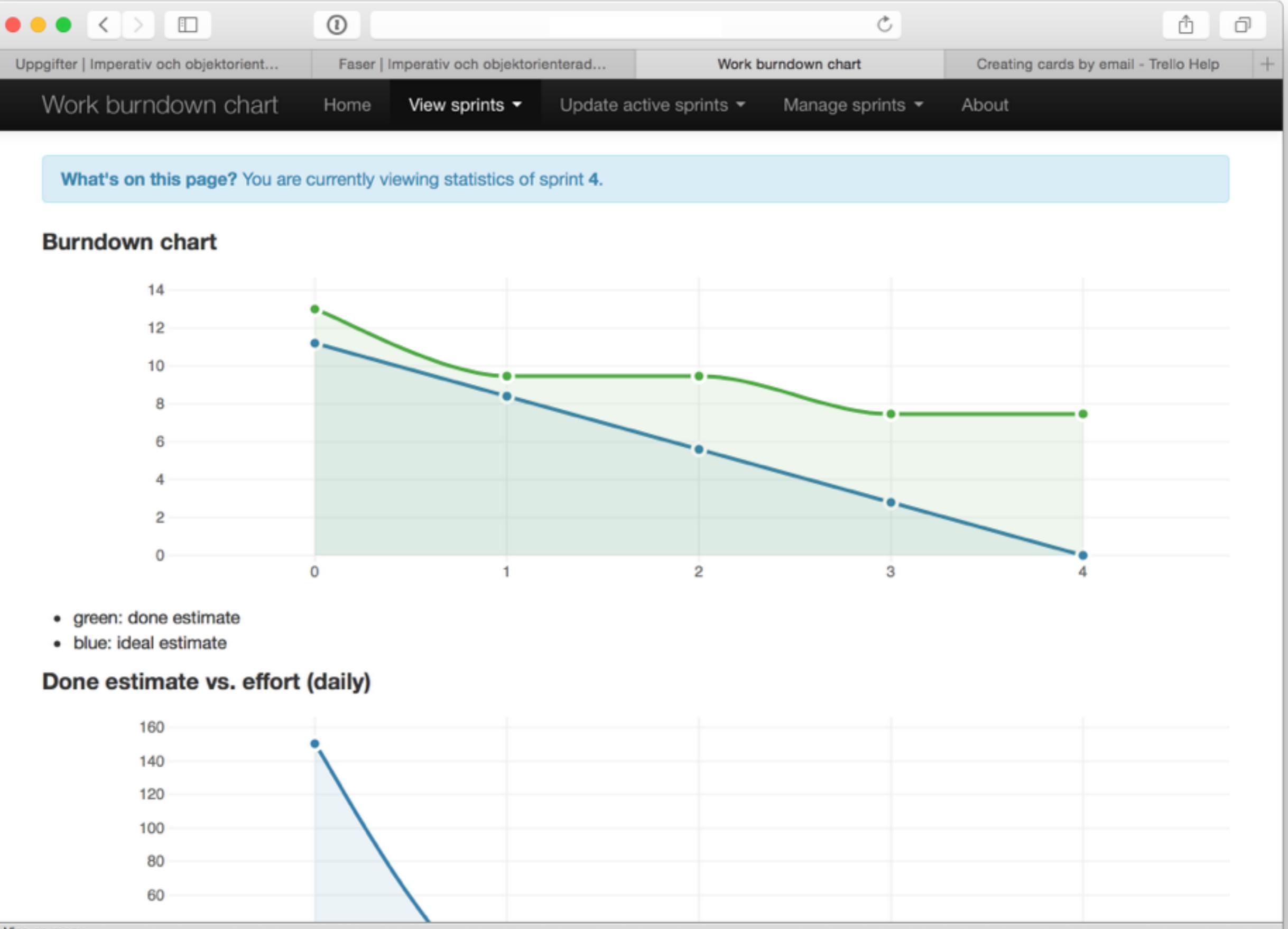
[Share and more...](#)

[Bias Wrigstad](#) [Show Menu](#)

Future Requests

- maybe datatype 1 comment
- parallel loops 10 comments
- Lazy Futures 8 comments
- annotation support 5 comments
- design (and implement) encoding for Encore 5 comments
- fields as sets 3 comments 0/7
- NQ (Language Integrated Query) 3 comments 0/7

[Create a card...](#)



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 (0, 7.2) and ending at (5, 0). The actual velocity starts at (0, 7.2), dips slightly to (1, 6.0), rises to (2, 6.2), dips again to (3, 6.0), rises slightly to (4, 6.2), and finally drops sharply to (5, 0).

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

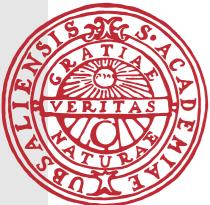
Visa en meny

Se "Länkar" på kurswebben

Högskolepoäng

	HP	Deadline	UPPDOK
Fas 1	5	21/10*	V44–45
Fas 2	5	3/12*	V49–50
Fas 3	5	18/12* 10/1*	V53–1
Kodprov	5	23/10 14/12	V3–4

*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: 6/12

- 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 känt 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 bero 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 orsa 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 vanliga 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 därför skräp och kommer att frigöra direkt sär referensräknande skräpsamlare är därför deterministisk², 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 MUTEX och kan därför användas tillsammans med destruktören, kod som körs automatiskt när objekten rörs.

En annan fördel med referensräkning är att arbetet för att ta reda på vad som är skräp är fördelat över hela programmet istället intet inte är fallet med spårande skräpsamlare. Detta är särdeles bra av snegletas med referensräkning olämnar det kräver en liten del extra arbete borta för att uppdatera objekters referensräknare³ men yeje gör en referens räknas till ändras då måste objekten lättas upp i minnet och dess referensräknare uppdateras. Det kan medföra stora mängder missar när objekten lättas upp i minnet, vilket påverkar prestandan negativt.

Det kanske stårta problemet med referensräknande skräpsamlare är att naiva implementeringar inte kan hantera cirkulära strukturer. Cirkulära strukturer innehåller objekt som direkt eller indirekt refererar till sig själv. Vissa implementeringar löser detta genom att använda svaga referenser som inte enbart inte klar objekts referensräknen. Det finns också mer komplexa algoritmer som kan hantera detta, men dessa kräver ofta stora mängder extra arbete.

2.2 Mark and sweep

Mark and sweep tillhör instegsin 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 det uppstår så kommer skräpsamlingens sätt vid olika trappsteg. Ofta startar detta när mängden ledigt minne tar slut eller faller under en viss nivå.

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

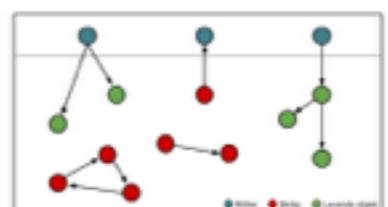


Figure 1: Objekter representerar objekt och pilarna referenser.

För att hitta alla levande objekt slår skräpsamlaren från ett antal rötter. Rötterna är slida referenser eller pelare som vi vet inte vi har tillträff till. I programmet slår den alla pelare som tillgå på staden. Skräpsamlaren går över rötterna och slår dessa referenser. Varje objekt som hittas markeras och för objekts medföra referenser slår uppgradera minnes process som för släckarna. Alltså vi slår referenser och alla objekt som hittas markeras och dessa referenser slår. Denna visualiseras i figuren.

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



Betyg

	3	4	5
Mål	36	18	9
Inluppar		5 (samma för alla)	
Projekt		6 (samma för alla)	
Varav utanför labb	3	2	3



Fas 3: Projektarbete

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

Rekommenderad start: 4/12

Klart: 18/12 eller 10/1

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

3

Uppgiften

Uppgiften går ut på att utveckla ett bibliotek, för enkeltets skull kallat "gc", för minneshantering i form av en kompakterande skräpsamlares. Med funktionen `b_init` kan en användare reservera en egen "heap" – ett konsekutivt minnesblock¹ i vilket man sedan kan allokerar minne med hjälp av biblioteksfunktioner. Detta minne ska sedan hanteras automatiskt – när minnet tas slut ska skräpsamlingen automatiskt triggas, och alla objekt i detta minne som inte är nödvändigt via någon rot i systemet tas bort².

Av pedagogiska skäl beskriver vi först skräpsamling med hjälp av mark-sweep, som vi öfver skulle använda innan vi går in på den kompakterande skräpsamlaren som använder en liknande algoritm.

3.1 Skräpsamling med mark-sweep

Skräpsamling med mark-sweep vandrar genom (traverserar) den graf som heapen utgör för att identifiera objekt som säkert kan deallokas utan att programmet kraschsar. 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". Ansas är objektet skräp som kan tas bort.

Vid skräpsamling 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 skräp initialt.

Steg 2 Sök igenom stacken efter 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.

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

¹T.ex. med hjälp av `malloc` i
`stdlib.h`, eller `new` i `new.h`.

²Vi gör en förenklning och utgår från
att programmet är enkeltredade
och att endast en heap skapas per
program.

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 (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

23/10 och 14/12

- Anmälan annonseras i Piazza



Simple [minimal sammanfattnings]

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 prova 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**

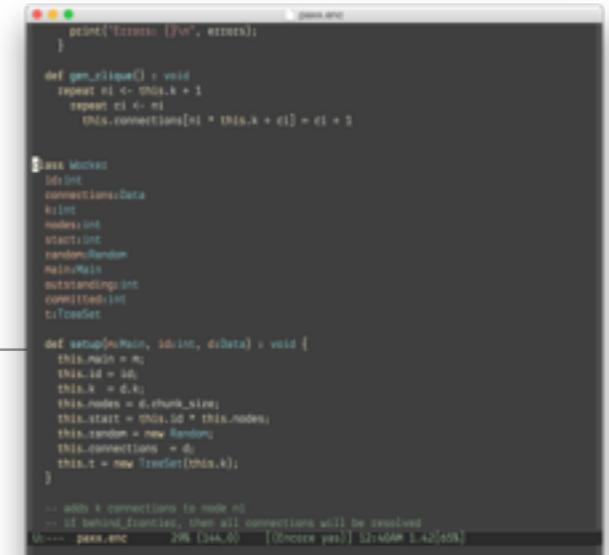
Gedit

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 connectionsData
#int nodes
#int start
#int random
#int maxNbrs
#int outstanding
#int committed
#int toTreeSet

def setup(#inMain, #inData, #inData) : void {
  this.main = #inMain;
  this.id = #inData;
  this.k = #inData;
  this.nodes = #inData.chunk_size;
  this.start = #inData.00 * this.nodes;
  this.random = new Random();
  this.connections = #inData;
  this.t = new TreeSet(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
```

