TDDE45 Lecture 1: Software Design and Construction Avancerad programvarudesign

Adrian Pop and Martin Sjölund

Department of Computer and Information Science
Linköping University

2024-09-02

Part I

Organization

Staff 2024

- Martin Sjölund, Examiner, Lectures, Group A
- Sajad Khosravi, Group B
- Anders Märak Leffler, Group C
- Hanan Mohsen, course secretary
- Martin Sjölund, director of studies

Course homepage: `https://www.ida.liu.se/~TDDE45`.
Lisam will contain info about Zoom links and old lecture recordings.

Last Year's Course Evaluation (1/2)

Responses 26⇒27%⇒37%⇒19%⇒22% last year.
3.30⇒3.72⇒2.78⇒3.5⇒3.9/5.00 in overall impression of the course.

- "Some instructions for the labs are vague and need more clarification." (This is by design, one needs to be able to read the documentation and find the information)
- "Did not get feedback on labs in time" (We need help from you to report such things during the course instead of during the course evaluation).
- Previous years: "Seminars work really well in online mode thanks to the break up room of Zoom." Last year: "Would have been nice to have the seminars in person"
- "Really liked the structure of having labs on Monday, submission on Thursday and seminar Friday."
- "Would like hand-ins through Lisam instead of email." (But Lisam is terrible for teachers; it only works well with a fixed deadline)

Last Year's Course Evaluation (2/2)

- "I think that the lectures were streamed on Zoom allowed me to participate from home when I was sick" (we have old lectures saved, and sometimes replacing a lecture with Zoom or pre-recorded when the lecturer cannot attend; attending in person is usually best)
- "The course is too broad." The course is broad by design – most of your courses give you in-depth knowledge on some topic. This is more of an overview of a lot of things that you will encounter when working. Some of you will have encountered some of the topics before (such as debugging).

Note that since there is no mandatory exam and around 7 weeks to finish the course, 23 hours/week can be expected (57-hour weeks if taking 15 credits of courses with all content during the study period). You have time to finish up stray assignments after the study period and still finish before the deadline, so it shouldn't be quite that bad. There is a lot to do, but the vast majority of students pass the course.

Changes from last year

- Change in course assistant John Tinnerholm ⇒ Anders Märak Leffler
- Seminar 1 was renamed seminar 0 to make numbering more logical (after feedback from Evaliuate)
- Martin was on parental leave and did not make major changes. Would you like last-minute changes from 1 group doing seminars in person? Or hand-ins through Lisam?

Curriculum: Intended learning outcomes

After the course, students shall be able to:

- analyze the structure and organization in a larger program
- critically evaluate design principles and -patterns in relation to desired properties of software and interfaces
- design a program that handles internationalization and localization
- apply tools that help the user to refactor or generate source-code
- apply different forms of metaprogramming
- apply methods and tools to find different kinds of problems in source code both before and during execution of a program, for example parallel execution or memory leaks

Curriculum: Course content

- History of design patterns.
- Design patterns.
- Refactoring.
- Use of integrated development environments.
- Code generation.
- Design for testability.
- Design of domain-specific languages.
- Metaprogramming.
- Dependency injection.
- Design of frameworks and API:s.
- Advanced debugging of memory allocation in parallel programs.
- Memory profiling.

Curriculum: Teaching and working methods

Students use lecture material and study course literature as preparations for lab assignments in pairs and seminars in groups, where issues concerning the course literature are treated.

The lab assignments differ in scope and depth: some train students in basic skills and techniques with some guidance, whereas other assignments have students develop and analyze larger programs with less guidance.

The course is very varied, with oral presentations and discussions during seminars, reasoning about design patterns, implementing code, and so on. Excelling in all areas will be difficult and you are not required to do so.

Högskolelag (1992:1434)

- Utbildning på avancerad nivå skall innebära fördjupning av kunskaper, färdigheter och förmågor i förhållande till utbildning på grundnivå och skall, utöver vad som gäller för utbildning på grundnivå,
  - ytterligare utveckla studenternas förmåga att självständigt integrera och använda kunskaper,
  - utveckla studenternas förmåga att hantera komplexa företeelser, frågeställningar och situationer, och
  - utveckla studenternas förutsättningar för yrkesverksamhet som ställer stora krav på självständighet eller för forsknings- och utvecklingsarbete. Lag (2006:173).

The Swedish Higher Education Act (1992:1434)

- Second-cycle courses and study programmes shall involve the acquisition of specialist knowledge, competence and skills in relation to first-cycle courses and study programmes, and in addition to the requirements for first-cycle courses and study programmes shall:
  - further develop the ability of students to integrate and make autonomous use of their knowledge,
  - develop the students' ability to deal with complex phenomena, issues and situations, and
  - develop the students' potential for professional activities that demand considerable autonomy, or for research and development work.

So you are expected to figure out many things for yourselves. Some lab instructions are deliberately short, giving only general instructions. Over the years we added some clarifications, but not everywhere.

Themes

- Design patterns
- Testability
- Cross-platform software construction
- Domain-specific languages
- Meta-programming and debugging
- Reading design

Many topics, with roughly 1 week per theme.
There will not be enough time for in-depth knowledge in a particular topic.

General routine for a theme

- Wednesday: Lecture; general information on the subject as well as describing the lab and seminar attached to the theme.
- Wednesday-Friday: Unsupervised time suitable for the next lab. Note that this time overlaps with preparing for Friday's seminar.
- Monday: Supervised lab.
- Friday: Seminar. Group discussion of the findings. Requires preparation to be sent to the lab assistant ahead of time.
- Monday: Writing and submitting lab and reflection report.

Roughly 6 hours supervised, **12 hours unsupervised per week** to reach the expected hours per course credit.

Exception: Week 1

- Today: Introduction.
- Wednesday: Lecture (Design Patterns)
- Friday: Seminar (Design Patterns).
- Next week Monday, Friday: Lab, Seminar (Design Patterns; usual structure)

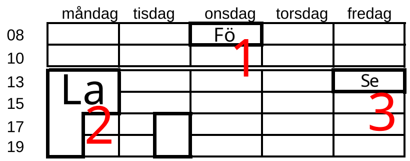
Time for lab hours

"More lab hours. 2h per lab is not enough, especially when you have to spend the weekend writing the report for the previous lab so you don't have time to prepare for the next (all lab sessions were on Mondays)."

The problem is that we can only have labs reliably on Mondays (getting 4 labs in the same 2-hour slot is impossible), and it has been very appreciated to have a fixed schedule every week.

Because of this, lectures were moved from Fridays as they were in the original edition of the course.

This gives you Wednesday-Monday to prepare for the supervised lab (but you have other work in parallel).



Groups

- 3 groups: Group A (Martin S.), Group B (Sajad K.), Group C (Anders M.L.)
- Labs are performed in pairs.
- Seminars are performed in groups of 6 students.
- Some labs have 3 different assignments (one for each pair).
- Sign up in WebReg (LAB1 before UPG1 to avoid concurrency problems; remember you need to be registered on the course to get credits). *NOTE! you need to choose the same group for LAB1 and UPG1!*
- Create your Gitlab groups according to the instructions on the course homepage. `https://www.ida.liu.se/~TDDE45/info/gitlab.en.shtml`

Examination

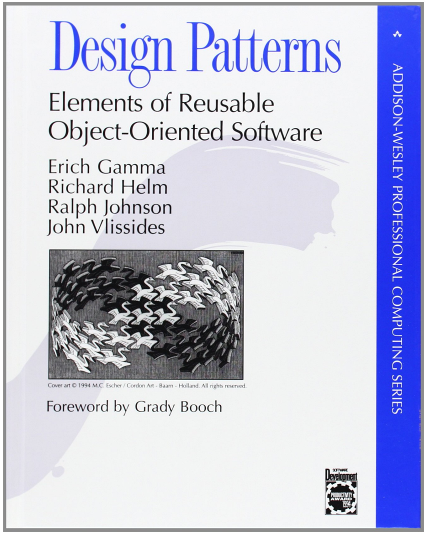
- Labs: Finish lab; write lab report after finishing the seminar (one report per pair).
- Seminars: Mandatory attendance with mandatory submission of preparation before the seminar.
- Computer exam: for higher grades, you will have needed to grasp more details of the topics/theory than was necessary to pass the labs. The computer exam may be skipped by completing the labs and seminars before the deadline (giving you a grade of 3).

Literature

Classic textbook; a bit dated but still relevant for people doing especially Java programming. Used for the first seminar and lab (first theme). Recommend to buy it used or borrow it.

Design Patterns: Elements of Reusable Object-Oriented Software
by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (the authors and/or the book are often referred to as the Gang of Four or GoF)

Hardcover: 395 pages
Publisher: Addison-Wesley Professional; 1st edition (January 15, 1995)
Language: English
ISBN: 0201633612


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Alternative Literature – Head First Design Patterns

More practical book with lots of figures and focuses on the design patterns rather than the story behind them.

Softcover: 638 pages
Publisher: O'Reilly
Language: English
ISBN: 978-0-596-00712-6, 978-1-492-07800-5 (2nd edition; new and I do not have a copy)

O'REILLY
Second Edition
Head First
Design Patterns
Building Extensible & Maintainable Object-Oriented Software
Eric Freeman & Elisabeth Robson
with Kathy Sierra & Bert Bates
A Brain-Friendly Guide

Rules for examination of computer lab assignments at IDA

- You are expected to do lab assignments in group or individually, as instructed for a course. However, examination is always based on individual performance.
- **It is not allowed** to hand in solutions copied from other students, or from elsewhere, even if you make changes to the solutions. If there is suspicion of such, or any other form of cheating, teachers are obliged to report it to the University Disciplinary Board.
- **Be prepared to answer** questions about details in specific code and its connection to theory. You may also be asked to explain why you have chosen a specific solution. This applies to all group members.
- **If you foresee problems** meeting a deadline, contact your teacher. You can then get some help and maybe the deadline can be set to a later date. It is always better to discuss problems, instead of, e.g., to cheat.
- **Any kind of academic dishonesty**, such as cheating, e.g. plagiarism or use of unauthorized assistance, and failure to comply with university examination rules, may result in the filing of a complaint to the University Disciplinary Board. The potential penalties include suspension, warning.

Policy for handing in computer lab assignments at IDA

- **For all IDA courses** having computer lab assignments there will be one deadline during or at the end of the course. If you fail to make the deadline, you must retake the, possibly new, lab course the next time the course is given.
- **If a course deviates** from this policy, information will be given on the course web pages.

Equal opportunities / Lika villkor

- Please note that equal opportunities mean special treatment of select groups is not allowed. If one student is given special treatment, all students need to get the same opportunity. This means for example that it is infeasible to give additional exams for exchange students (who often move home before the second re-exam period, and so on). The same goes for late submissions of labs.
- Ideally, there would be a mixture of exchange students and Swedish students in the labs (1/3 of students taking the course are exchange students).

Exceptions to posted deadlines

- Reports are corrected when it is the last course remaining in the curriculum, or required to advance to the next grade (submit a transcript).
- Reports are also corrected when required to get credit for the course at a different university (submit proof of deadline).
- It is possible to miss a single seminar with a good excuse (such as medical procedures); a make-up exercise is then performed. Notify your assistant in advance to make preparations.

Reporting in Ladok is performed outside of the regular course schedule as time permits.

Computer Labs

Note that lab instructions have little detail in them as you are supposed to read up on documentation to figure things out yourself rather than follow instructions. During the labs, you will use many different programming languages, tools and libraries.

- Java
- C
- C++
- Julia
- More, especially in domain-specific languages lab

You are allowed to use your own computer, Windows computers, etc. But if you do so, you will not get as good support from your lab assistant. Recommended is using the IDA labs or thinlinc.edu.liu.se.

WebReg

Time to find pairs and register in WebReg. You can access it from the course page: `https://www.ida.liu.se/~TDDE45/`

Note that you need to sign up to the same group (same assistant) for both labs (LAB1) and seminars (UPG1)!

Part II

Content Overview

Why design? Software qualities

- **Maintainability**
- Efficiency
- Reliability
- Security
- **Size** (estimate size for cost estimation)

Consortium for IT Software Quality

Maintainability

- How easily can I make changes to accommodate new demands?
- How easily can I test this method?
- How easy is it for someone else to understand this?
- How easy is it to isolate faults and locate bugs with this design?

Design Principles

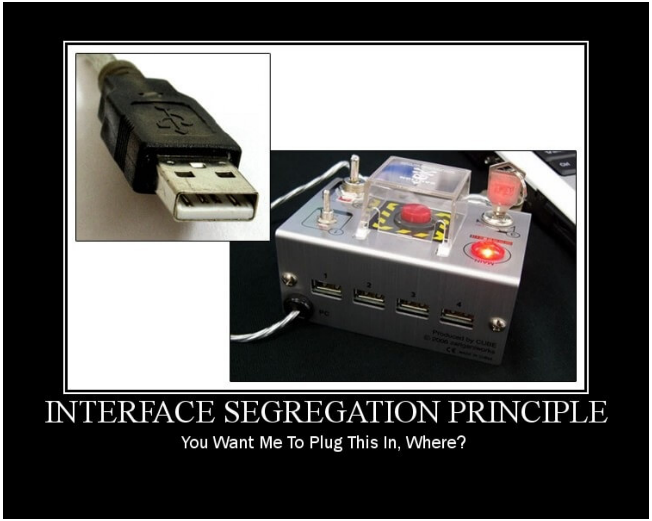
SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should

Design Principles

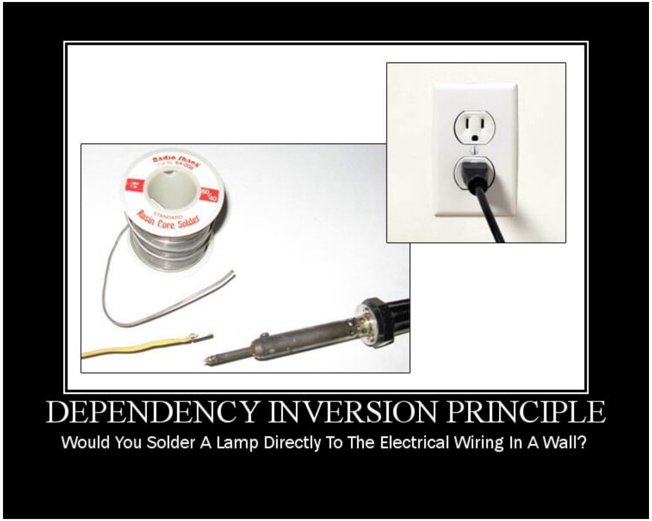
Design Principles

LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

Design Principles

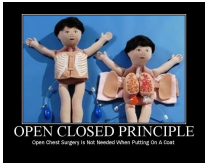

INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

Design Principles

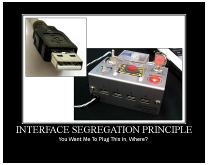

Radio Shack
60/40
Rosin Core Solder
DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

Design Principles: SOLID

SINGLE RESPONSIBILITY PRINCIPLE
Just Because You Can, Doesn't Mean You Should



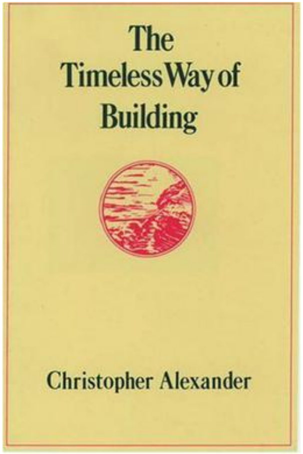
LISKOV SUBSTITUTION PRINCIPLE
If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction


INTERFACE SEGREGATION PRINCIPLE
You Want Me To Plug This In, Where?

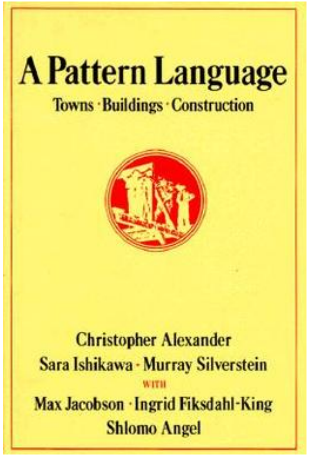
DEPENDENCY INVERSION PRINCIPLE
Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

SOLID Motivational Posters by Derick Bailey, used under CC BY-SA 3.0 US

From Design Principles to Design Patterns (1979)

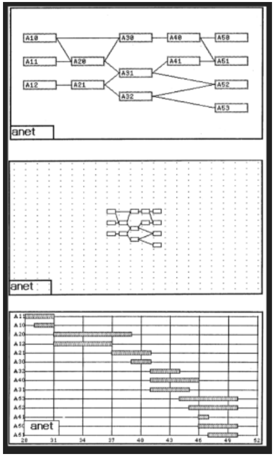


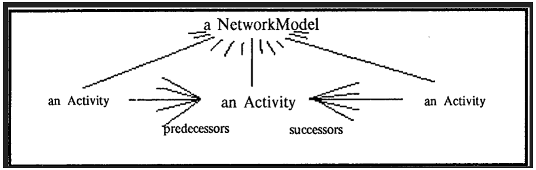
Domain


A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

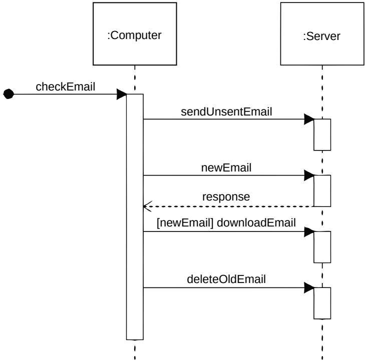
Language

From Design Principles to Design Patterns: Old Example

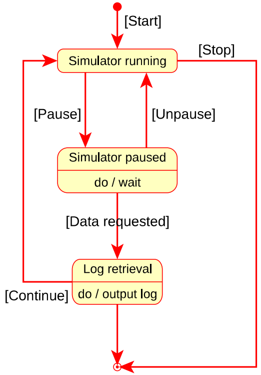




From Design Principles to Design Patterns: Modern Example (UML)

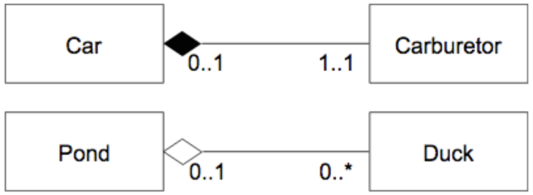


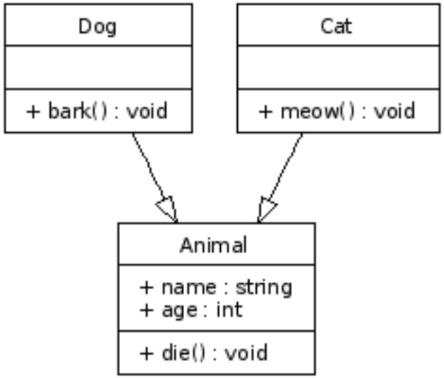
Sequence Diagram by Coupling_loss_graph.svg
Licensed under CC BY-SA 3.0 via Commons
`https://commons.wikimedia.org/wiki/File:CheckEmail.svg`



State Diagram by Fred the Oyster
Licensed under CC BY-SA 4.0 via Commons
`https://commons.wikimedia.org/wiki/File:UML_State_diagram.svg`

From Design Principles to Design Patterns: Modern Example (UML)

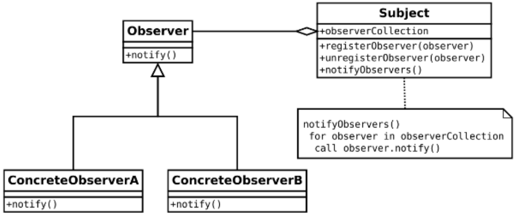




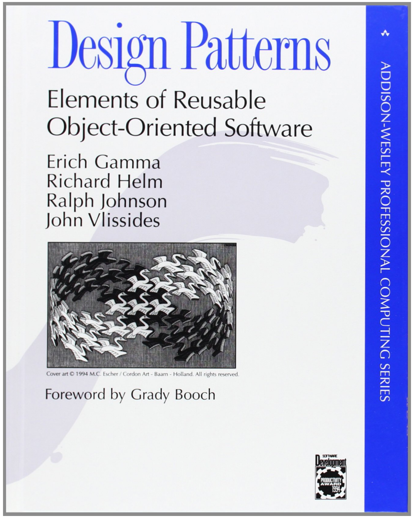
Class Diagrams

Example Design Pattern

Observer
Intent: Define a 1-m dependency between objects so that when one object changes state, all its dependencies are notified and updated automatically.
Application: All mouse listeners, keyboard listeners.



Design Patterns


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES
Software Development Productivity Award 1994

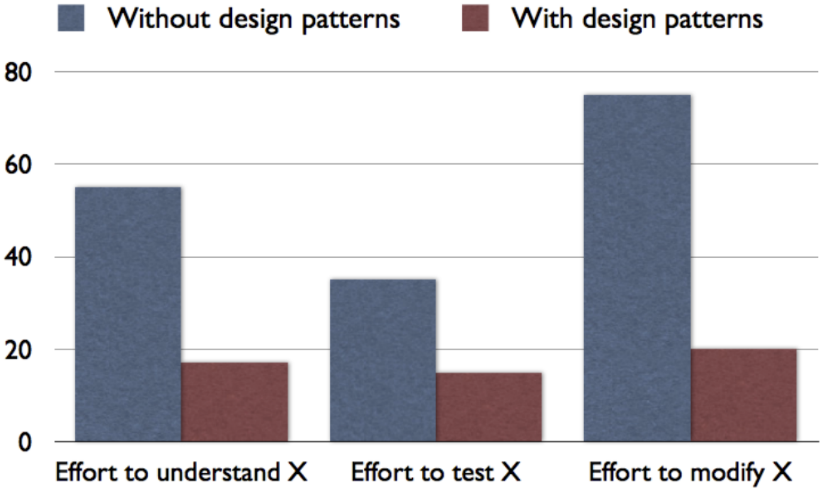
Language

Intent
Applicability
Consequences
...

Domain

Java
C++?
Smalltalk?
With or without framework or metaprogramming support?
In any programming paradigm?

Design Patterns Hypothesis

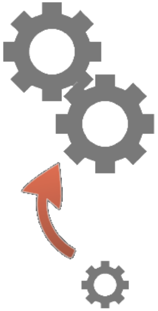


Application Frameworks

Reusable components

Inversion of control

Metaprogramming



Application Framework Examples

- Ruby on Rails
- Unity
- Qt
- Spring
- Castle Windsor
- Mocking/stubbing libraries

Other Programming Paradigms: Reactive

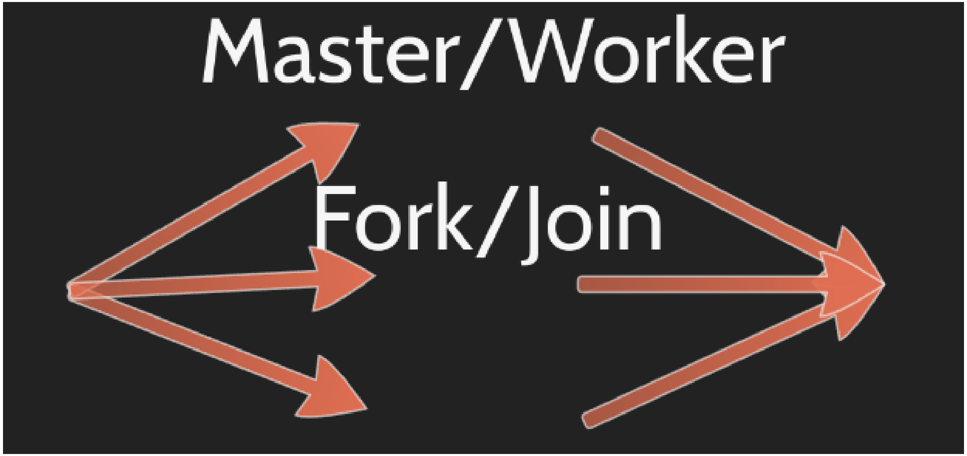
```
var up   = $('#up').asEventStream('click');
var down = $('#down').asEventStream('click');

var counter =
  // map up to 1, down to -1
  up.map(1).merge(down.map(-1))
  // accumulate sum
     .scan(0, function(x,y) { return x + y });

// assign observable value to jQuery property text
counter.assign($('#counter'), 'text')
```

Replaces observer?

Other Programming Paradigms: Parallel



Creates new patterns? (Search for
Concurrency patterns)

Design and constructions: Domain-specific languages

- Text processing. Regular expressions
- Build systems
- Graphics, drawing, documents (HTML, PostScript, etc)
- Solutions for specialized problems (Prolog, Modelica, etc)
- Embedded languages
- Generators

Design and constructions: Tools, libraries

- Translation (gettext, linguist)
- Debuggers (when things go wrong)
- Static analysis and testing (to prevent things from going wrong)
- Profiling (to reduce resource usage)

Are Design Patterns language deficiencies?

- Before/after method $\rightarrow$ Decorator
- First-class types $\rightarrow$ Factories
- Macros $\rightarrow$ Interpreter
- First-class methods (functional programming) $\rightarrow$ Strategies
- Multiple dispatch (Common Lisp, Julia) $\rightarrow$ Visitor

Or simply the canonical way of solving a problem in a *given language* (or version of a language)?

Some things you need to know for the course

- UML. Some students did not take a course that used UML before, and many forgot. There are links to tutorials – the first seminar is the first week and you need to be able to draw UML class diagrams for it (not copying diagrams from webpages or books).
- Linux systems. The labs have been tested on the IDA computer system, which used Ubuntu 20.04 (and was upgraded to 22.04 right before the course started). Some labs can be done at home on a Windows machine as well, but using WSL is a lot easier since you get a full Ubuntu system with access to your Windows file system.
- Git. You will need to hand in your labs via gitlab.liu.se. If you don't know git, take a quick tutorial. Some instructions are on the course homepage.
- Pointers and data. If you only worked with safe languages like Java, you might have trouble understanding the debugging lab.

Some things you need to know for the course

- Multiple programming languages. Some of you will only have programmed in a single programming language throughout all your courses. In this course, you might learn the basics of several languages per week (the course is broadening, not in-depth). This is a good skill for engineers to have.
- It is good to perform the labs in pairs of students from different backgrounds – one of the best ways of learning a subject is teaching it to someone else.

End

- Next lecture: Design patterns
- See the literature and the course page:
  `https://www.ida.liu.se/~TDDE45/info/lectures.en.shtml`
- Start preparing for the first seminar on Friday

www.liu.se

LINKÖPING
UNIVERSITY