Design Patterns 448.058 (VO)

Michael Krisper
Georg Macher

02.10.2019

The Team

Michael Krisper
michael.krisper@tugraz.at
Uncertainty and Risks
in Cyber-Security

Georg Macher
georg.macher@tugraz.at
Safety & Security
in Automotive &
Autonomous Driving

In memoriam:
† Christian Kreiner

ITI - Institute for Technical Informatics

Inffeldgasse 16, 1st Floor

Bachelor's Thesis, Master's Thesis, Projects, Seminar, PhD
Topics Presentations on Tue, 8.10.2019 at 14:30 in NXP Seminarroom (IE01090)

Learning Goals for Course

Design Patterns Theory

- What is a design pattern? Why do we need them?
- What are principles behind design patterns?
- How to describe design patterns?
- What is a pattern language?

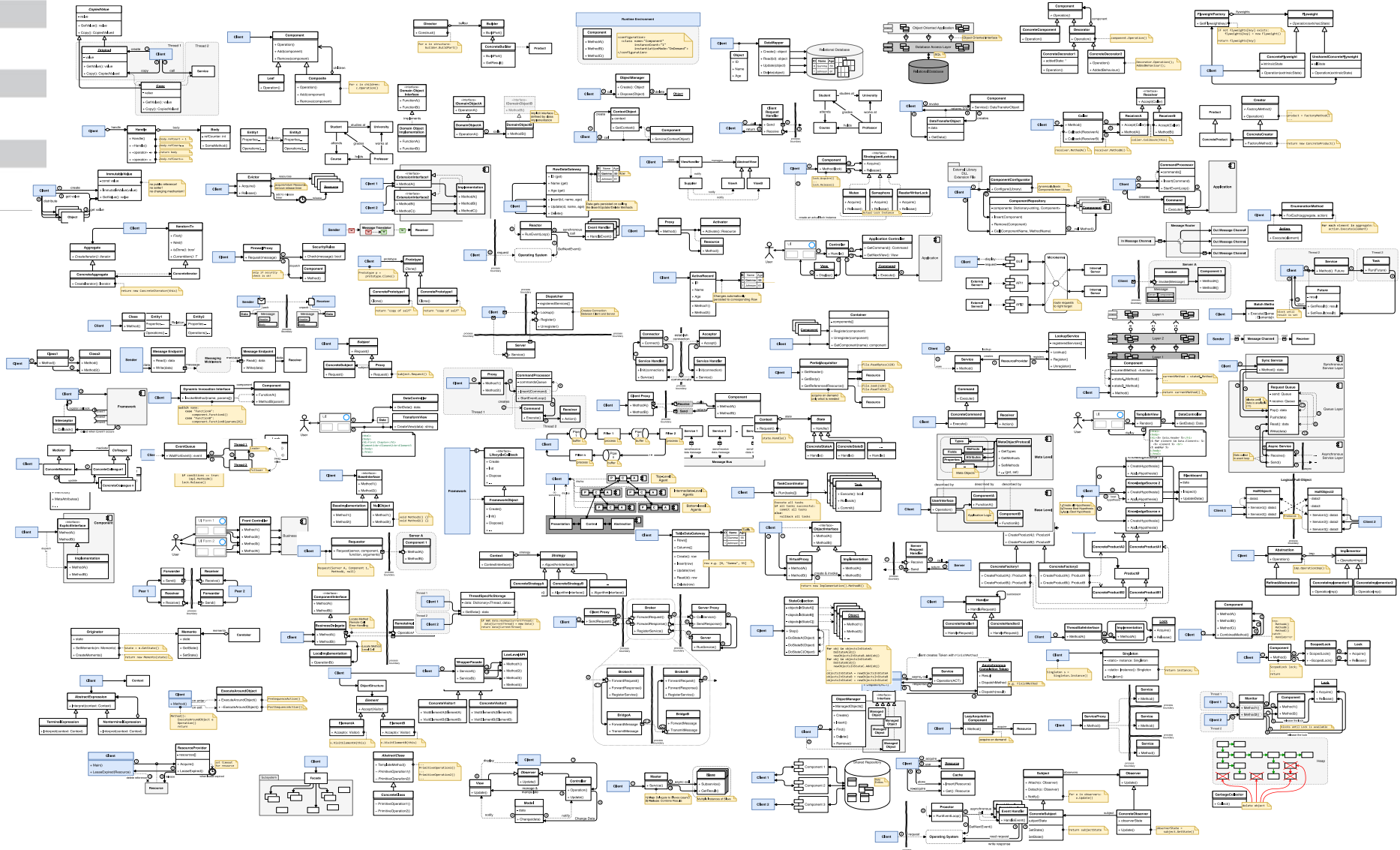
Application of Design Patterns

- When to use what?

Design Patterns in Detail

- Know core ideas and application of important design patterns! (~50)

Overview over all Patterns in the Course:



If you tell me, I will listen.

If you show me, I will see.

But if you let me experience, I will learn.

老子 (Lǎozǐ, 500 BC)

What is a pattern?

A proven solution for a (recurring) problem.

- But it's not a concrete solution!
- A concrete solution is just one example.
- A Pattern is rather a **solution idea**, **scheme,** or **template**.

Patterns are a universal principle:

- Economics (Etzioni, 1964)
- Social Interaction (Newell,Simon, 1972)
- Architecture (Alexander et. al., 1975)
- Software (General awareness from 1990's on)

Purpose of Design Patterns

- Easier knowledge transfer
- Efficient problem solving by reusing existing ideas
  “Don’t reinvent the wheel”
- Establishes a common vocabulary, terminology, or language
- Increases usefulness of an idea by generalizing the solution

Standard Literature

- **GOF**: Design Patterns – Elements of Reusable Object-Oriented Software (Gamma, Helm, Johnson, Vlissides, 1995)
- **POSA1**: Pattern-Oriented Software Architecture Volume 1: **A system of patterns** (Buschmann, Meunier, et al., 1996)
- **POSA2**: Pattern-Oriented Software Architecture Volume 2: **Patterns for Concurrent and Networked Objects** (Schmidt et al., 2000)
- **POSA3**: Pattern-Oriented Software Architecture Volume 3: **Patterns for Resource Management** (Kircher and Jain, 2004)
- **POSA4**: Pattern-Oriented Software Architecture Volume 4: **Pattern Language for Distributed Computing** (Buschmann, Henney, and Schmidt, 2007)

Types of Design Patterns

Architectural Patterns

- Fundamental structural patterns
- Stencils for whole architectures
- Examples: Layers, Pipes-And-Filters, Broker, Model-View-Controller, Microkernel, Async-Await

Design Patterns

- Solution templates for more isolated problems
- Examples: Composite, Adapter, Proxy, Factory

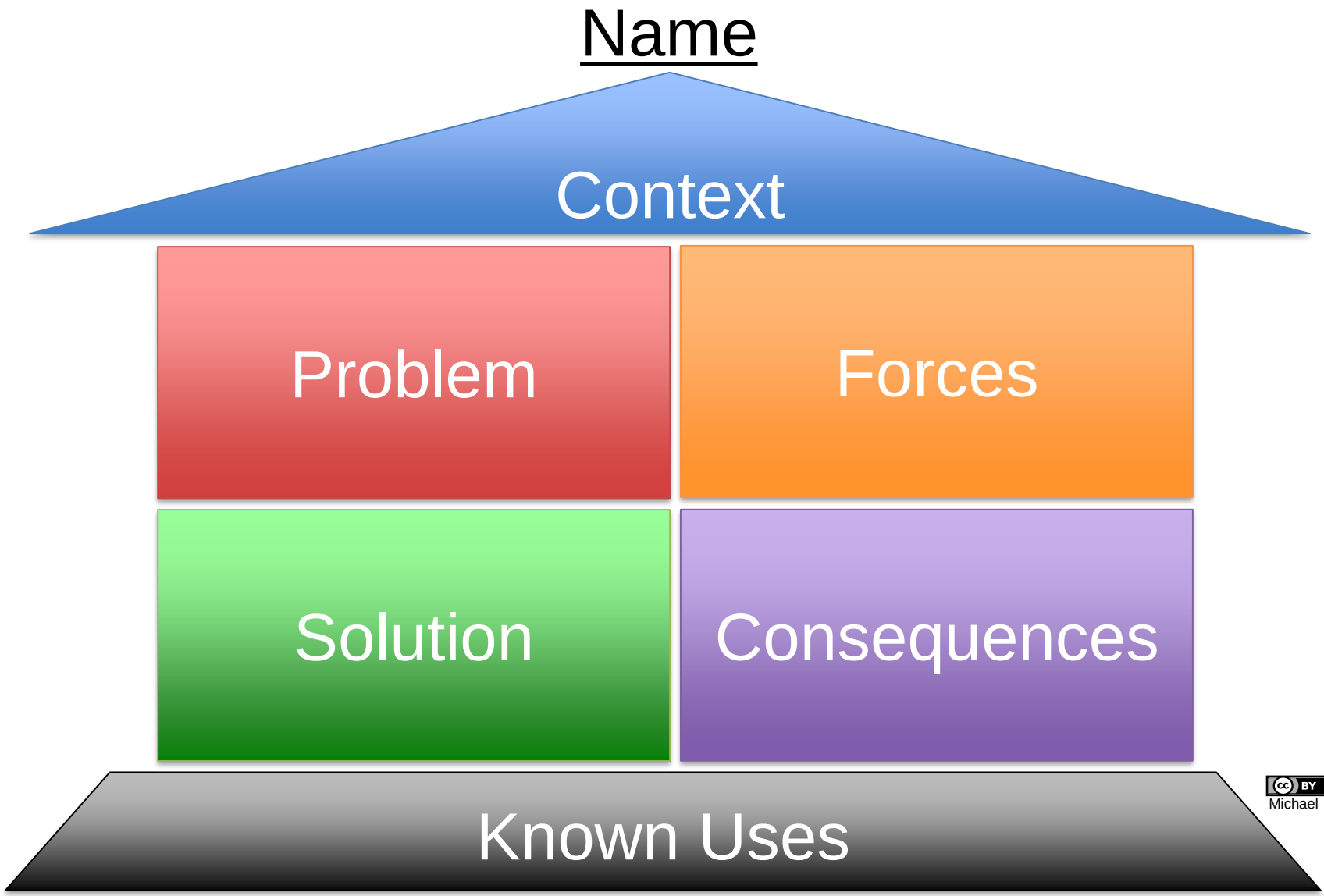
Idioms

- Fine-Grained Patterns for problems in specific programming languages or environments
- Examples: Counted Pointer, Scoped Locking, Variadic Macros

Pattern format

- **Name**: A catchy name for the pattern
- **Context**: The situation where the problem occurs
- **Problem**: General Problem Description
- **Forces**: Requirements and Constraints - Why does the problem hurt in this context?
- **Solution**: Generic Description of a proven solution.
  Static Structures, Dynamic Behaviour, Actionable Steps
- **Consequences (Rationale, Resulting Context):**
  - What are the benefits and drawbacks? Pro and Contra?
  - What are the liabilities, limitations and tradeoffs?
  - How are the forces resolved?
- **Known-Uses**: Real Life Examples

The Design Pattern House



Michael Krisper

Alexandrian Pattern Format

Name

92 BUS STOP*

Picture

Context

. . . within a town whose public transportation is based on MINIBUSES (20), genuinely able to serve people, almost door to door, for a low price, and very frequent, [illegible] to be bus stops within a few hundred feet of every house and workplace. This pattern gives the form of the bus stops.

✣ ✣ ✣

Problem and Forces

Bus stops must be easy to recognize, and pleasant, with enough activity around them to make people comfortable and safe.

Bus stops are often dreary because they are set down independently, with very little thought given to the experience of waiting there, to the relationship between the bus stop and its surroundings. They are places to stand idly, perhaps anxiously, waiting for the bus, always watching for the bus. It is a shabby experience; nothing that would encourage people to use public transportation.

The secret lies in the web of relationships that are present in the tiny system around the bus stop. If they knit together, and reinforce each other, adding choice and shape to the experience, the sys[illegible] make up such a system are extremely subtle. For example, a system as simple as a traffic light, a curb, and street corner can be enhanced by viewing it as a distinct node of public life: people wait for the light to change, their eyes wander, perhaps they are not in such a hurry. Place a newsstand and a flower wagon at the corner and the experience becomes more coherent.

The curb and the light, the paperstand and the flowers, the awning over the shop on the corner, the change in people's pockets—all this forms a web of mutually sustaining relationships.

The possibilities for each bus stop to become part of such a web are different—in some cases it will be right to make a system that will draw people into a private reverie—an old tree; another time one that will do the opposite—give shape to the social possibilities—a coffee stand, a canvas roof, a decent place to sit for people who are not waiting for the bus.

Solution and Consequences

92 BUS STOP

Two bus stops.

Therefore:

Build bus stops so that they form tiny centers of public life. Build them [illegible] to neighborhoods, work communities, parts of town. Locate them so that they work together with several other activities, at least a newsstand, maps, [illegible], seats, and in various combinations, corner groceries, smoke shops, coffee bar, tree places, special road crossings, public bathrooms, squares. . . .

✣ ✣ ✣

Related Patterns, Epilogue

Make a full gateway to the neighborhood next to the bus stop, or place the bus stop where the best gateway is already—MAIN GATEWAY (53[illegible]) [illegible] to the patterns for PUBLIC OUTDOOR ROOM (69), PATH SHAPE (121), and A PLACE TO WAIT (1[illegible]); PROVIDE A FOOD STAND (93): place the seats according to sun, [illegible] view—SEAT SPOTS (241). . . .

451 452 453

Excerpt from [*Alexander, Christopher (1979). The Timeless Way of Building. Oxford University Press. ISBN: 978-0-19-502402-9*]

How Design Patterns emerge?

Design Patterns are found - not invented!

They emerge out of real use-cases/known-uses

1. Find patterns in real solutions
   ➔ At least three Known-Uses, Real Projects!
2. Write down the core idea and experiences
   ➔ Context, Problem, Forces, Solution, Consequences
3. Discuss with others (often & repeatedly)
4. Improve Pattern (and repeat discussions)
5. Publish! (Conferences, Books, Blogs)
6. Continue to improve, apply and discuss pattern

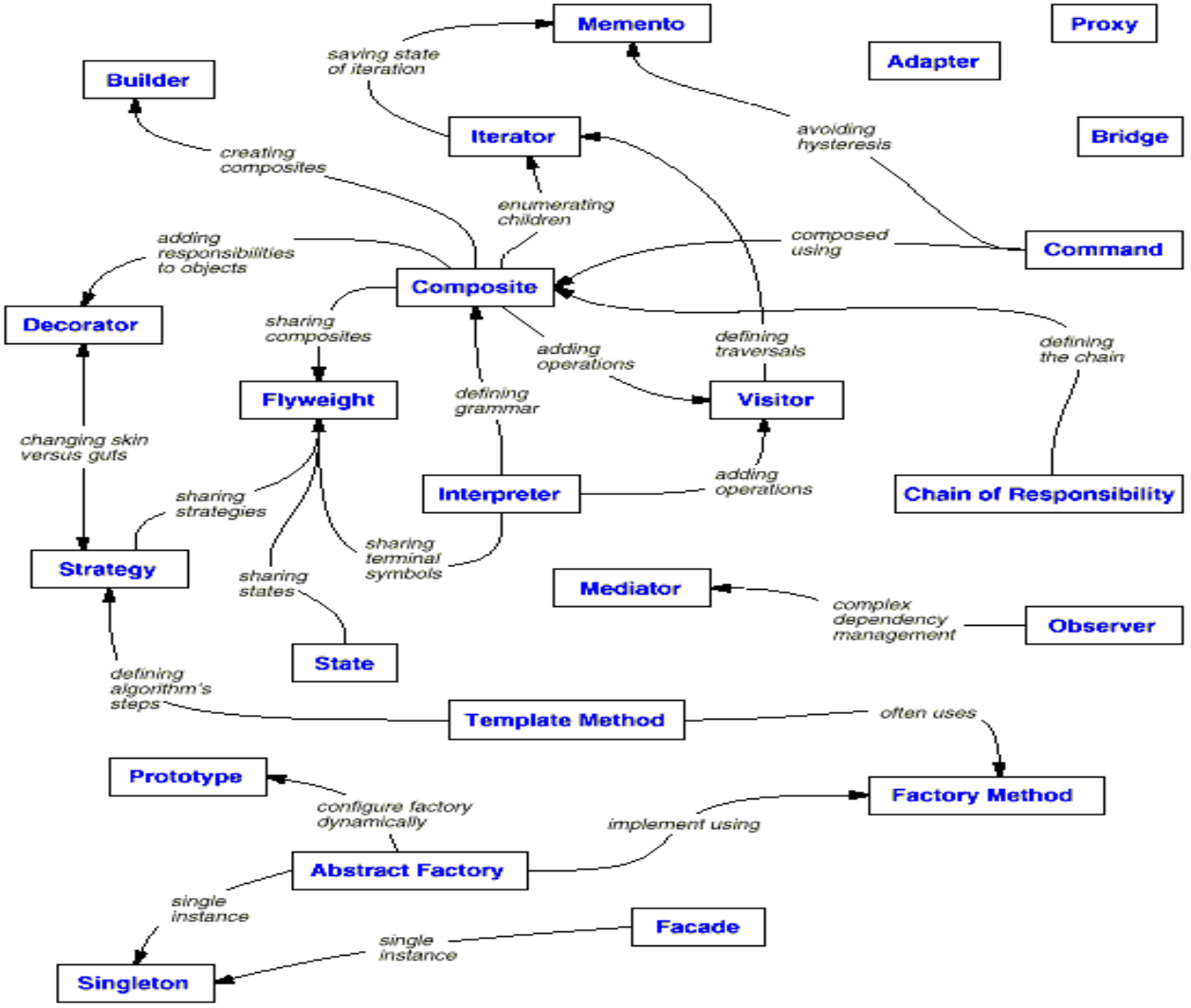
Pattern Languages

… are coherent systems of patterns.

- Patterns
- Relations
- Principles (Guidelines for design and evolution):
  - How to create / implement
  - Beneficial combination of patterns
  - How to change/evolve

Daily Life Examples: Cooking, Sports, Crafts, Sailing, Architecture, Programming, …

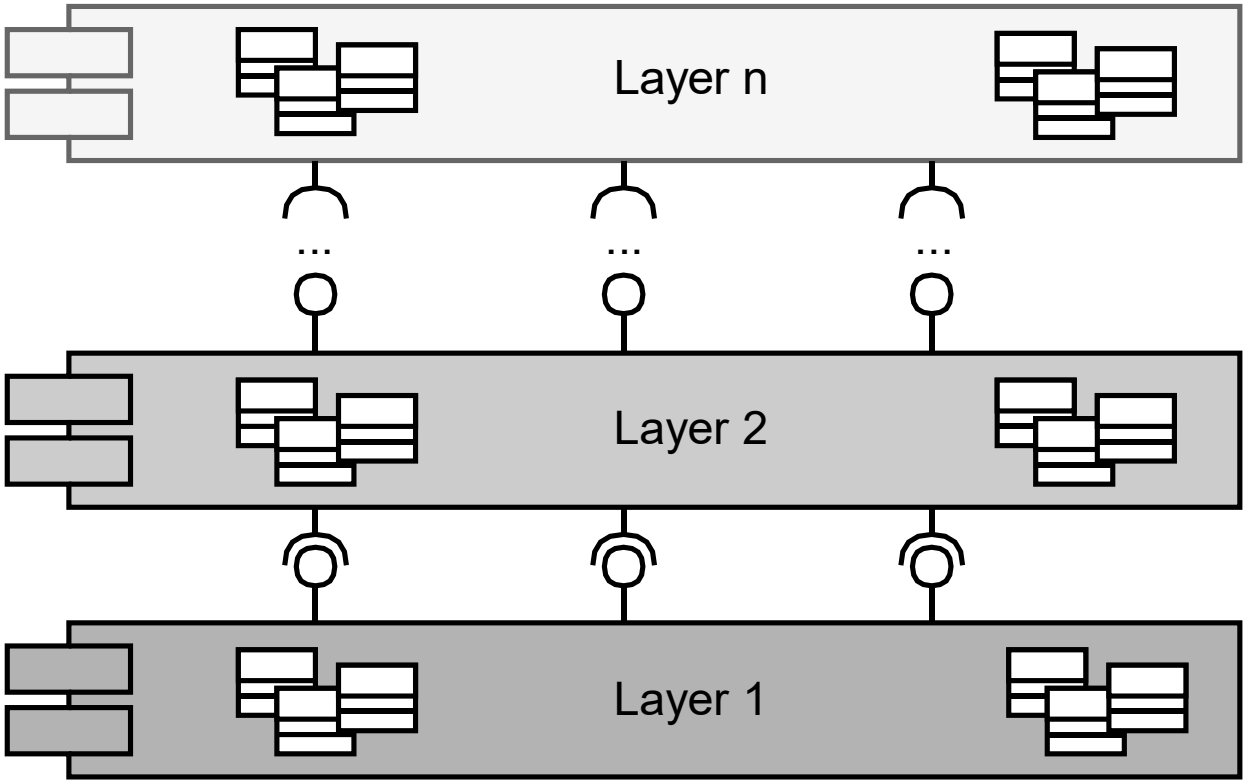
GOF Pattern Language



Graphic from [Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995, ISBN 0-201-63361-2]

Layers

Split your system into layers based on abstraction levels



Layers

Context: Large systems that require decomposition

Problem:

- Many functions and responsibilities
- Hard to understand structure, many dependencies

Forces:

- Changes should be limited to one component
- Clear boundaries of responsibility
- Interfaces should be stable
- Parts should be exchangeable
- Parts should be reusable
- Smaller groups for easier understandability, maintainability

Solution:

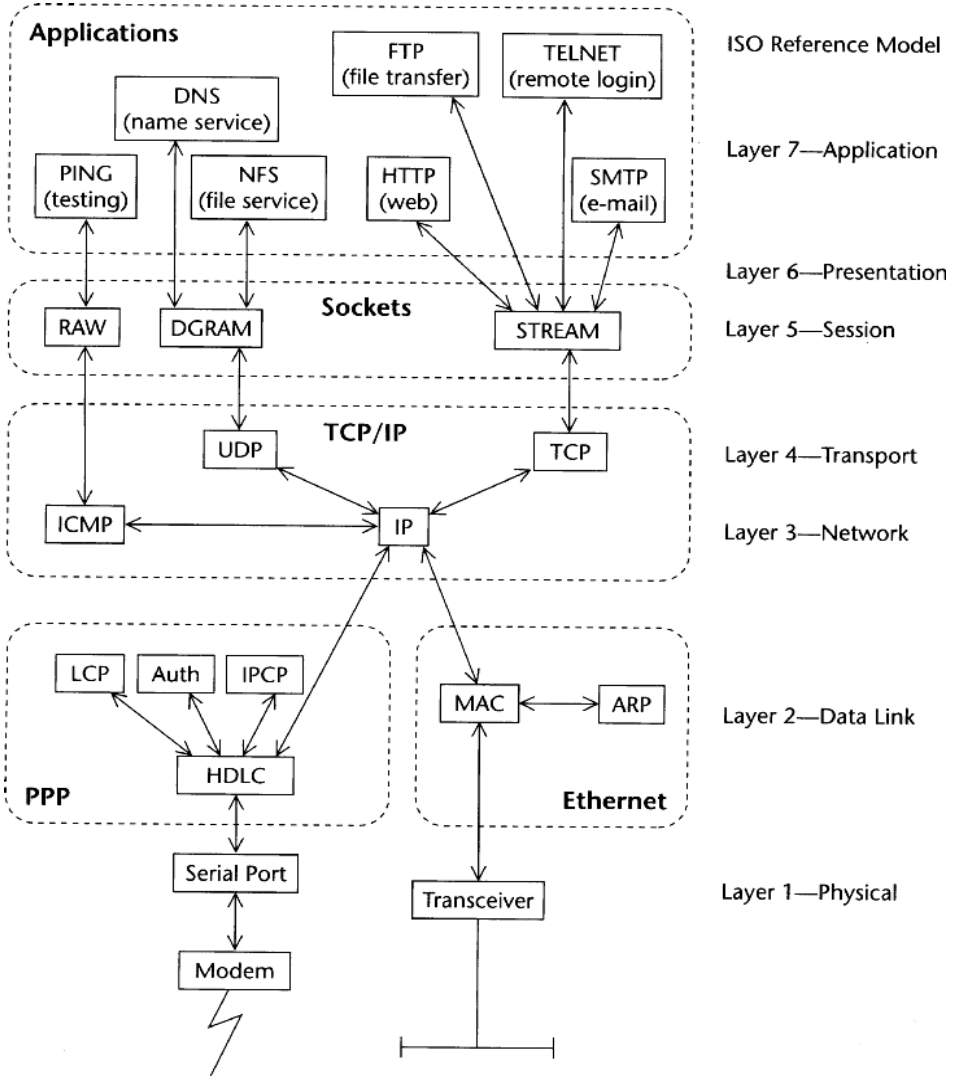
- Structure the function into appropriate number of layers, based on their abstraction levels
- Every layer uses defined services of sublayer
- Every layer provides defined services to upper layer

Consequences:

+ Dependencies/Changes are kept local
+ Defined Interfaces between Layers
+ Layers are exchangeable & reusable
- Lower efficiency
- No fine grained control of sublayers
- Changes cascade and are costly
- Right granularity is difficult to find

Layers – Known Uses

- Network Stack
- Virtual Machines
- API's
- Operating Systems
- Companies
- Cities
- …



Layers – Implementation Issues

- Who composes the layers at runtime?
- How are Interfaces defined?
- Workarounds / Skip layers?
- Stateless / Stateful Implementations?
- Layers are Black Boxes