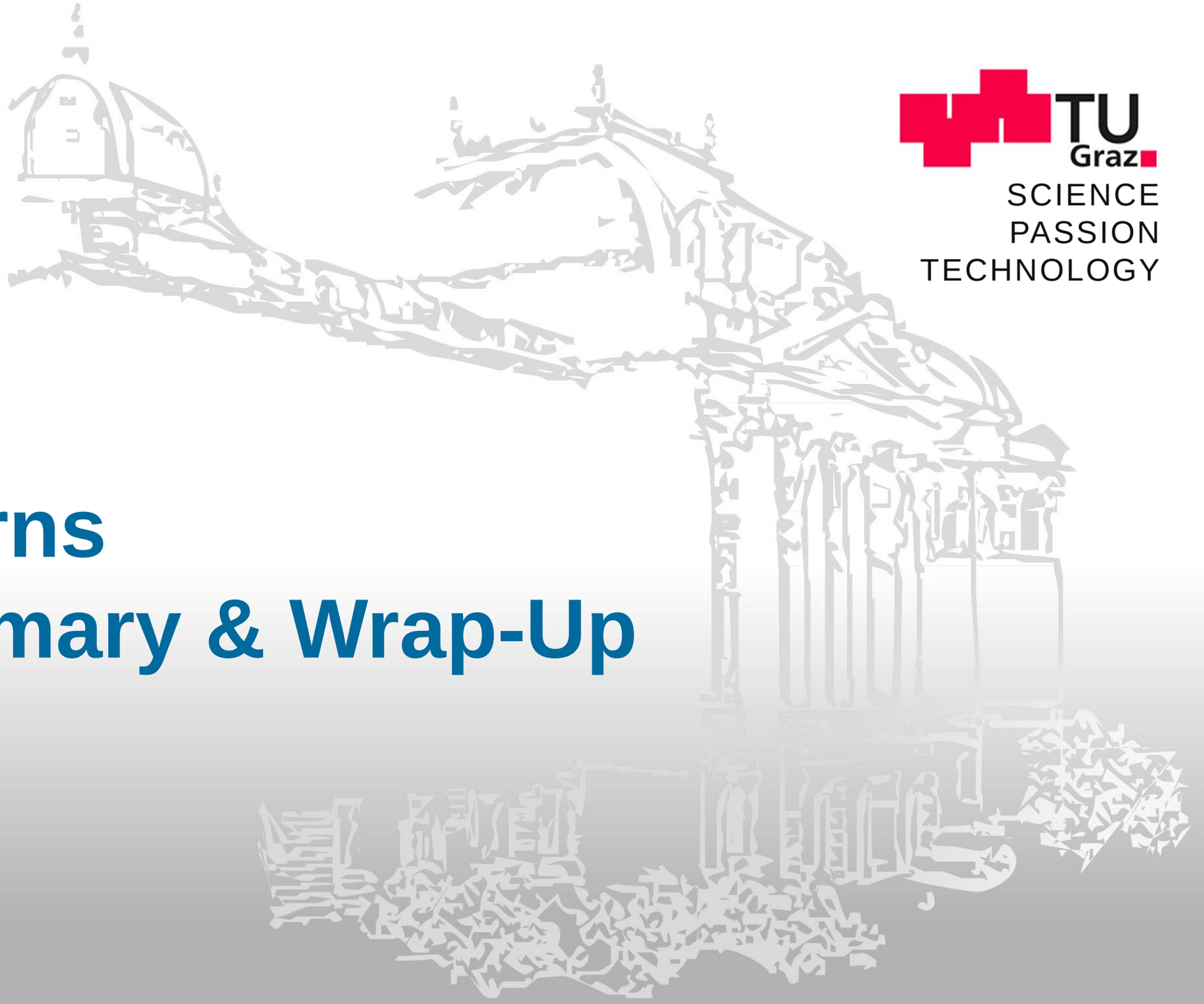


TU Graz
SCIENCE
PASSION
TECHNOLOGY

Design Patterns
Part 12: Summary & Wrap-Up

Schedule

Date	from	to	Content
07.10.2020	13:00	16:00	Introduction, Organisation
14.10.2020	13:00	16:00	Theory, Principles, and Guidelines, Iterator
21.10.2020	13:00	16:00	Adapter, Facade, Decorator, Proxy
28.10.2020	13:00	16:00	Layers, Broker, Pipes & Filters, Master/Slave, Client/Server, Broker
04.11.2020	13:00	16:00	Factory Method, Abstract Factory, Builder, Singleton, Prototype, Memento, State, Flyweight
11.11.2020	13:00	16:00	Visitor, Strategy, Command, Composite, Template Method, Fluent Interface
18.11.2020	13:00	16:00	Mediator, Bridge, Blackboard, Microkernel, Messages (Message, Endpoint, Translator, Router), Observer
25.11.2020	13:00	16:00	Locks (Mutex, Semaphor, Condition Variable), Scoped Locking, Double Checked Locking, Monitor, Future/Asynchronous Completion Token, Active Object, Thread Specific Storage, Async-Await
02.12.2020	13:00	16:00	Lazy Acquisition, Eager Acquisition, Partial Acquisition, Caching, Pooling, Leasing, Garbage Collector, Scoped Resource, Active Record
09.12.2020	13:00	16:00	Chain of Responsibility, Counted Pointer, Interpreter/Abstract Syntax Tree, Proactor, Reactor
13.01.2021	12:00	15:00	Model-View-Controller, Model-View-Viewmodel, Model-View-Presenter, Presentation-Abstraction-Control
20.01.2021	13:00	16:00	Summary and Exam Preparation
27.01.2021	**13:00**	**15:00**	**Exam**

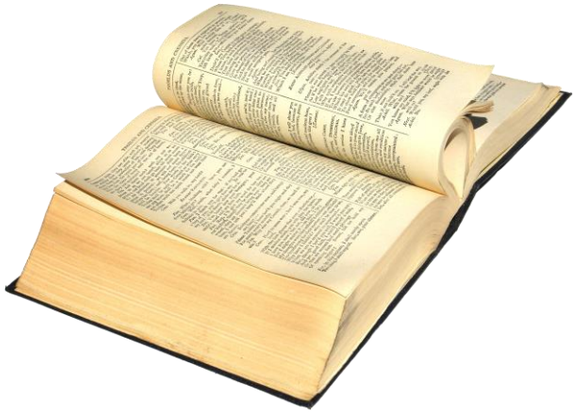
Patterns in this course…

- **Wrapping**: Adapter, Façade, Decorator, Proxy
- **Creation**: Factory Method, Abstract Factory, Builder, Prototype, Singleton, Flyweight
- **Behaviour**: Strategy, Command, State
- **Architecture**: Layers, Pipes & Filters, Broker, Master-Slave, Client-Server
- **Collections**: Iterator, Visitor, Composite
- **Communication**: Observer, Bridge, Broker, Mediator, Blackboard, Microkernel, Client-Dispatcher-Server/Lookup, Messaging & Service-Orientation: Message, Message-Endpoint, Message-Translator, Message-Router, MVC
- **Concurrency**: Locks, Monitor, Active Object, Future, Scoped Locking, Thread-Specific Storage, Double-Checked-Locking, Async/Await, Proactor, Reactor
- **Resources**: Lazy Acquisition, Eager Acquisition, Partial Acquisition, Caching & Pooling, Leasing, Garbage Collector, Scoped Resource
- **Others**: Memento, Counted Pointer, Chain of Responsibility, Interpreter/Abstract Syntax Tree

Learning Goals

Design Patterns Theory

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



Design Patterns in Detail

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

Application of Design Patterns

- When to use what?

Learning Goals

- You know common design patterns and their core idea (approx. 50 patterns).
- You can apply them in software development regardless of the programming language or development environment.
- You can derive the consequences of design patterns and see the design decisions.
- You decide if the consequences of a pattern are acceptable or not.
- You avoid overengineering and misuse of patterns.
- You can make reasonable design decisions by balancing out the forces, consequences, and requirements for arbitrary problems and contexts.

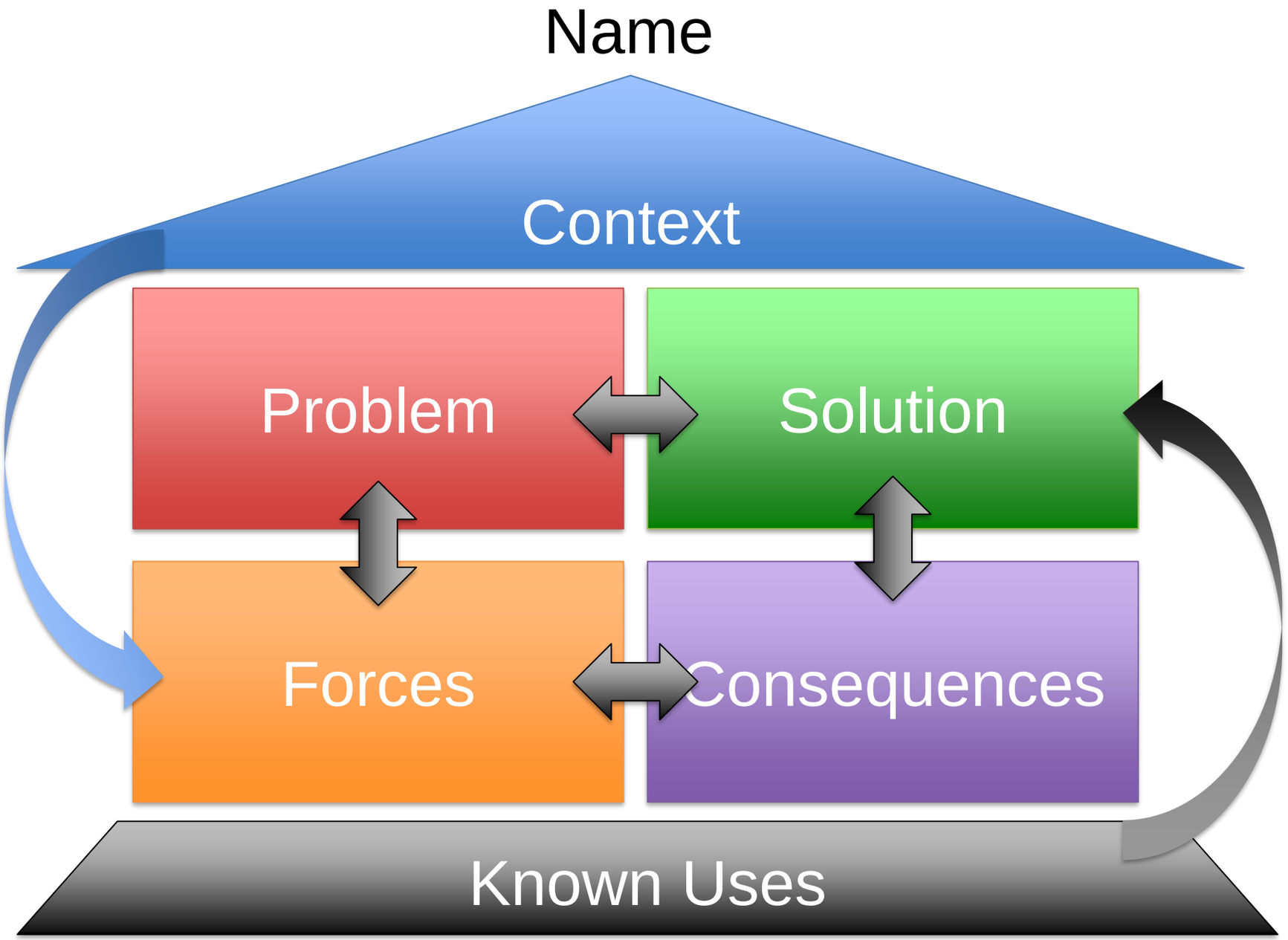
Design Patterns

What is a pattern?

A proven solution template for a recurring problem.

- **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
- **Consequences** (Rationale):
  What are the benefits and liabilities?
  What are the limitations and tradeoffs?
  How are the forces resolved?
- **Known-Uses**: Real Life Examples

Design Patterns House



SOLID Principles (in OOP)

- ***S*ingle Responsibility**: A class should have one, and **only one**, **reason to change**.
- ***O*pen Closed**: You should be able to **extend** a class's behavior, **without modifying** it.
- ***L*iskov Substitution**: Derived classes must be **substitutable** for their base **classes**.
- ***I*nterface Segregation**: Make **fine grained interfaces** that are client specific.
- ***D*ependency Inversion**: **Depend on abstractions**, not on concrete implementations.

Principles of Good Programming

- **Decomposition**
  make a problem manageable
  decompose it into sub-problems
- **Abstraction**
  wrap around a problem
  abstract away the details
- **Decoupling**
  reduce dependencies, late binding
  shift binding time to “later”
- **Usability & Simplicity**
  make things easy to use right, hard to use wrong
  adhere to expectations, make usage intuitive

Types of Design Patterns

Architectural Patterns

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

Design Patterns

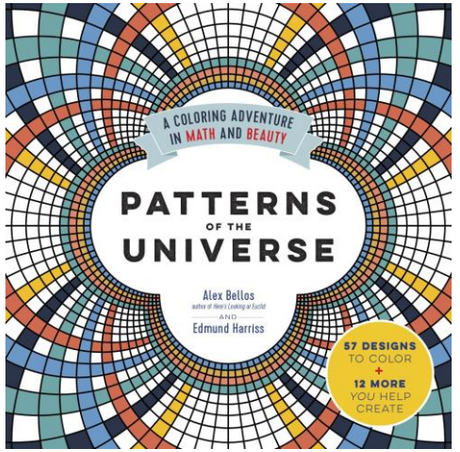
- Solution templates for more isolated problems
- Examples: Composite, Adapter, Proxy, Factory

Idioms

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

A few philosophical thoughts…

“Patterns are a universal principle”



- **How to transfer knowledge?**

- How to make knowledge explicit?
- How to make knowledge findable?
- How to make knowledge understandable?
- How to make knowledge applicable?

“Study hard what interests you the most in the most undisciplined, irreverent and original manner possible.”

— Richard Feynmann

Michael Krisper
michael.krisper@tugraz.at

Uncertainty and Risk Propagation
Expert Judgment for Cyber-Security
Vehicle CO_2 Simulation

Georg Macher
georg.macher@tugraz.at

Safety & Security
Automotive & Autonomous Driving
Distributed Industrial Systems

Remember us for your Projects/Seminars/Bachelor's/Master's Thesis

Institute of Technical Informatics
Web: https://iti.tugraz.at
Discord: https://discord.gg/rFXPjW3