Design Patterns and Frameworks (DPF) Announcements

Dr. Sebastian Götz
Software Technology Group
Department of Computer Science
Technische Universität Dresden

WS 19/20, Oct 15, 2019

Slides from Prof. Dr. U. Aßmann

Design Patterns and Frameworks

Goals and Contents of the Course

Main Goals

- Know several different kinds of patterns
  - Basic kinds of incentives for design patterns
- Explain patterns for **variability** and **extensibility** of systems
- Understand frameworks and product lines better
- Explain systematic structures for systems with >100KLOC
  - Layered frameworks
  - Facets
- Understand a different way of object-oriented design
  - Role-based design

Standard Problems to Be Solved By Design Patterns

- Variability
  - Exchanging parts easily
  - Variation, complex parametrization
  - Static and dynamic
  - For product lines, framework-based development
- Extensibility
  - Software must change
- Gluing (bridging, adapting, connecting)
  - Overcoming architectural mismatches
  - Coupling software that was not built for each other
- Others:
  - Optimization: making things more efficient
    - Antagonistic to flexibility
  - Structuring of interactive applications
    - Grasping common patterns of flow in software systems

Goal: Variability Patterns

- Variability (Variation, Exchange, Parametrization)
  - Expressing commonality and variability
  - We fix a common part (a *framework*) and parametrize it at *variation points (variability)*
  - *Framework instantiation patterns* describe variations of frameworks
- Understanding **Templates** and **Hooks**
  - Template Method vs Template Class
  - Dimensional Class Hierarchy, Bridge
- Understanding creational patterns
  - Factory Method, Factory Class, Builder
- Variability design patterns for frameworks
- Variability concerns
  - Exchange of communication
    - Dynamic call (e.g., ChainOfResponsibility)
  - Exchange of policy
  - Exchange of material in data-based applications

Goal: Extensibility Patterns

- Extensibility
  - For new, unforeseen product variants
  - For evolution
  - For dynamic change
- Understanding extensibility patterns
  - ObjectRecursion vs TemplateMethod, Objectifier (and Strategy)
  - Decorator vs Proxy vs Composite vs ChainOfResponsibility
  - Visitor, Observer (EventBridge)
- Parallel class hierarchies as implementation of facets
  - Understand facets as non-partitioned subset hierarchies
  - Layered frameworks as a means to structure large systems, based on facets
- Template/Hook Extension:
  - Code skeletons are *extended* at *hooks*
  - Frameworks can have hooks that can be extended (beyond variation)
- Framework extension patterns

Goal: Gluing Patterns for Overcoming Architectural Mismatches

- Glue patterns
  - Understand architectural mismatch
  - Understand patterns that bridge architectural mismatch
- Adaptation, bridging, connections
  - Of communication protocols
  - Between heterogeneous components (different representations, different locations, different control flow structure)
- Anonymous communication
  - For exchange of commmunicators
- Scalable communication
  - At runtime, in distributed systems

Goal: A Basic Tool: Role Modeling

- For all of that, a basic tool set is role modeling
  - Which roles does an object play in the application?
- It tells how design patterns occur in applications
  - Reenskaug. Summarized in the book “Working with Objects”, 1995
- Role-model based design
  - Why design patterns are role models of class diagrams
  - Understand the difference between roles and objects, role types and classes
  - Understand role mapping to classes
    - How roles can be implemented
  - Understand role model composition
  - Understand composite design patterns as composition of role models

Goal: Framework Patterns

- Understand variabilities in frameworks
  - Introducing different types of hooks for frameworks and components (TH patterns)
  - Understanding framework variability patterns
- Studying extensible framework hook patterns
  - Role Object pattern
  - Layered frameworks, implemented by Role Object
- Patterns document frameworks
  - Patterns play an important role on how a framework is instantiated
  - Whitebox, blackbox, layered, T&H framework

Goal: Structuring Interactive Applications with Tools&Materials

- Understand the central metaphors of the Tools-and-Materials architectural style for the construction of interactive applications
  - Know an example of a pattern language
- Interactive applications can be pretty complex
- TAM (tools-and-materials, Werkzeug-Automat-Material, WAM) is a *pattern language for interactive applications*
- Nice metaphors that help thinking, constructing, maintaining interactive applications

Overview of the Course

Tools & Materials

Layered Frameworks

Part 3: Frameworks

Metapatterns
and Framework patterns

Role Models

Composite Patterns

Part 2: Roles

Part 1: Basic Patterns

Variability Patterns

Extensibility Patterns

Glue Patterns

Intro

Employment and Usage

Design Patterns and Frameworks

Introduction

History

- Beginning of the 70s: the window and desktop metaphors (conceptual patterns)
  - Smalltalk group in Xerox Parc, Palo Alto
- 1978/79: MVC pattern for Smalltalk GUI. Goldberg and Reenskaug at Xerox Parc
  - During porting Smalltalk-78 for Norway in the Eureka Software Factory project [Reenskaug]
- 1979: Alexander's “The Timeless Way of Building”
  - Introduces the notion of a *pattern* and a *pattern language*
- 1987: W. Cunningham, K. Beck: OOPSLA paper “Using Pattern Languages for Object-Oriented Programs”
  - Discovered Alexander's work for software engineers by applying 5 patterns in Smalltalk
- 1991: Erich Gamma. Design Patterns. PhD Thesis
  - Working with ET++, one of the first window frameworks of C++
  - At the same time, Vlissides works on InterViews (part of Athena)
  - Pattern workshop at OOPSLA 91, organized by B. Anderson
- 1993: E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns: Abstraction and Reuse of Object-Oriented Design. ECOOP 97, LNCS 707, Springer, 1993.
- 1994: First PLOP conference (Pattern Languages Of Programming)
- 1995: GOF book.
- 1997: Riehle on role models and design patterns
- 2005: Collaborations (class-role models) in UML

Alexander's Laws on Beauty

- Christopher Alexander. "The timeless way of building" . Oxford Press 1977.
  - Hunting for the "Quality without a name":
  - When are things "beautiful"?
  - When do things "live"?
- Patterns grasp centers of beauty
- You have a language for beauty, consisting of patterns (a *pattern language*)
  - Dependent on culture
- Beauty cannot be invented
  - but must be combined/generated by patterns from a pattern language
- The "quality without a name" can be reached by pattern composition in pattern languages

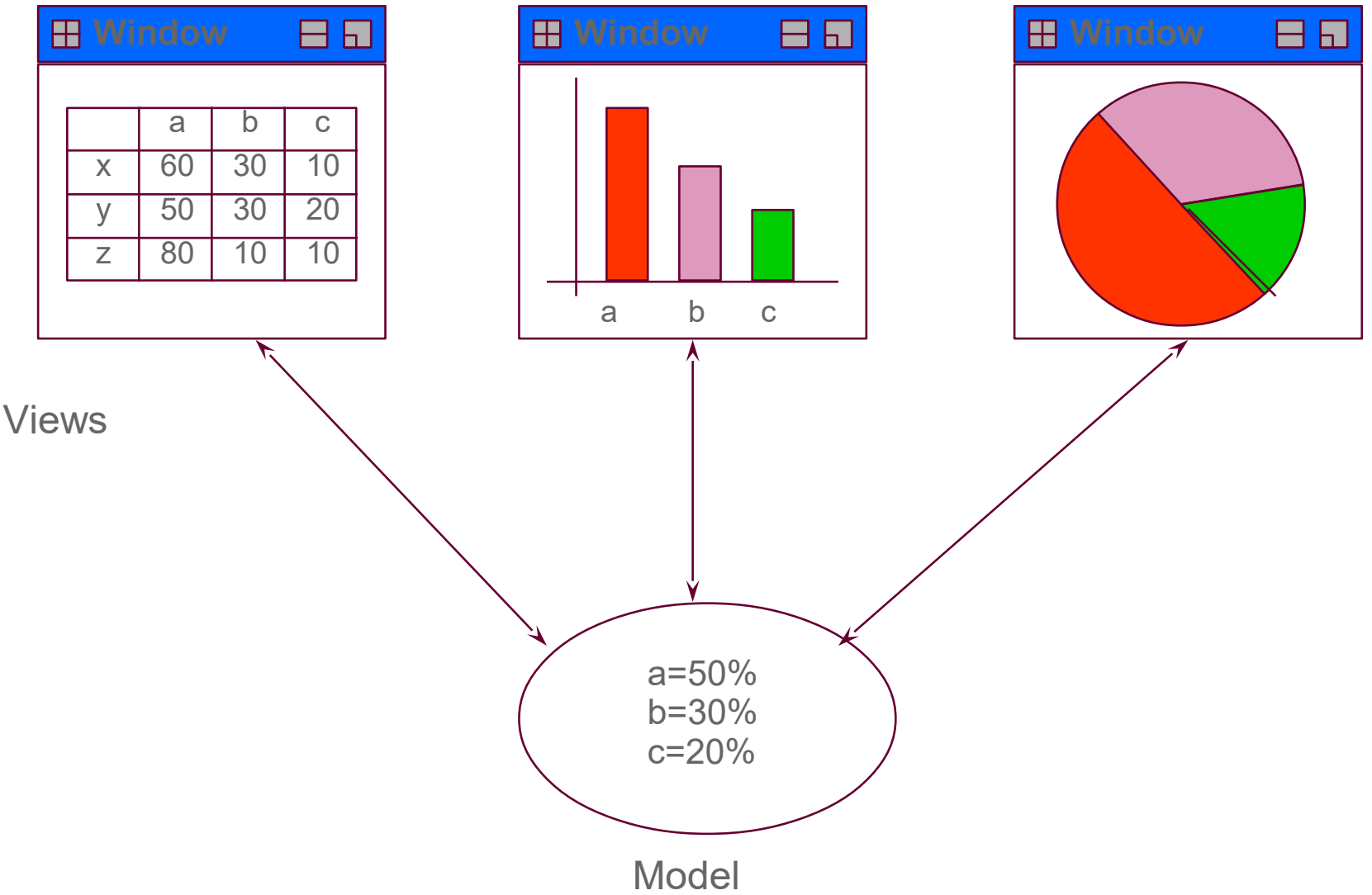
The Most Popular Definition

- A Design Pattern is
  - A description of a standard solution for
  - A standard design problem
  - In a certain context
- Goal: Reuse of design information
    - A pattern must not be “new”!
  - A pattern writer must have an “aggressive disregard for originality”
- In this sense, patterns are well-known in every engineering discipline
  - Mechanical engineering
  - Electrical engineering
  - Architecture

Example: Model/View/Controller (MVC)

- MVC is a agglomeration of classes to control a user interface and a data structure
  - Developed by Goldberg/Reenskaug 1978, for Smalltalk
- MVC is a complex design pattern and combines the simpler ones composite, strategy, observer.
- Ingredients:
  - Model: Data structure or object, invisible
  - View: Representation(s) on the screen
  - Controller: Encapsulates reactions on inputs of users, couples model and views

Views as Observer

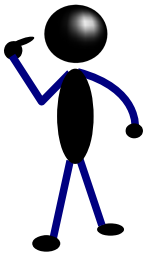


Patterns

- Pattern 1: Observer: Grasps relation between model and views
  - Views may register at the model (observers).
  - They are notified if the model changes. Then, every view updates itself by accessing the data of the model.
    - Views are independent of each other. The model does not know how views visualize it.
  - Observer decouples strongly.
- Pattern 2: Composite: *Views* may be *nested* (represents trees)
  - For a client class, Composite unifies the access to root, inner nodes, and leaves
  - The MVC pattern additionally requires that
    - There is an abstract superclass View
    - The class CompositeView is a subclass of View
    - And can be used in the same way as View
- Pattern 3: Strategy: The relation between *controller* and *view* is a *Strategy.*
  - There may be different control strategies, lazy or eager update of views (triggering output), menu or keyboard input (taking input)
  - A view may select subclasses of *Controller,* even dynamically. Strategy allows for this dynamic exchange (variability)

What Does a Design Pattern Contain?

- A part with a “bad smell”
  - A structure with a bad smell
  - A query that proved a bad smell
  - A graph parse that recognized a bad smell
- A part with a “good smell” (standard solution)
  - A structure with a good smell
  - A query that proves a good smell
  - A graph parse that proves a good smell
- A part with “forces”
  - The context, rationale, and pragmatics
  - The needs and constraints



Structure for Design Pattern Description (GOF Form)

- Name (incl. Synonyms) (also known as)
- Motivation (purpose)
  - also “bad smells” to be avoided
- Employment
- Solution (the “good smell”)
  - Structure (Classes, abstract classes, relations): UML class or object diagram
  - Participants: textual details of classes
  - Interactions: interaction diagrams (MSC, statecharts, collaboration diagrams)
  - Consequences: advantages and disadvantages (pragmatics)
  - Implementation: variants of the design pattern
  - Code examples
- Known Uses
- Related Patterns

Purposes of Design Patterns

- Improve communication in teams
  - Between clients and programmers
  - Between designers, implementers and testers
  - For designers, to understand good design concepts
- Design patterns create an “ontology of software design”
  - Improvement of the state of the art of software engineering
  - Fix a glossary for software engineering
  - A “software engineer” without the knowledge of patterns is a programmer
  - Prevent re-invention of well-known solutions
- Design patterns document abstract design concepts
  - Patterns are “mini-frameworks”
  - Documentation, In particular frameworks are documented by design patterns
  - May be used to capture information in reverse engineering
  - Improve code structure and hence, code quality

1.2 Different Kinds of Patterns

What is a Pattern?

- There is not “the pattern”
- At least, research is done in the following areas:
  - Conceptual patterns
  - Design Patterns
    - Different forms
  - Antipatterns
  - Implementation patterns (programming patterns, idioms, workarounds)
  - Enterprise patterns
  - Process patterns
    - Reengineering patterns
  - Organizational patterns
- General definition:
  - A pattern is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts [Riehle/Zülinghoven, Understanding and Using Patterns in Software Development]

Conceptual Patterns

- A **conceptual pattern** is a pattern whose form is described by means of the terms and concepts from an application domain
  - Based on metaphors in the application domain
- Example: conceptual pattern “desktop”
  - Invented in Xerox Parc from A. Kay and others
    - Folders, icons, TrashCan
    - Drag&Drop as move actions on the screen
  - Basic pattern for all windowing systems
  - Also for many CASE tools for visual programming
  - Question: what is here the “abstraction from the concrete form”?
- We will revisit in the Tools-And-Materials (TAM) pattern language
  - It works on conceptual patterns such as “Tool”, “Material”, “Automaton”

Design Patterns

- "A **design pattern** superimposes a *simple structure* of a relation in the static or dynamic semantics of a system"
  - Relations, interactions, collaborations
  - Nodes: objects, classes, packages
- "A design pattern is a named nugget of insight which conveys the essence of a proven solution to a recurring problem within a certain context amidst competing concerns" [Appleton]

Different Types of Design Patterns

- Fundamental Design Pattern (FDP)
  - A pattern that cannot be expressed as language construct
- Programming Pattern, Language Dependent Design Pattern (LDDP)
  - A pattern that exists as language construct in another programming language, but is not available in general
- Architectural pattern
  - A design pattern that describes the coarse-grain structure of a (sub)system
  - A design pattern on a larger scale, for coarse-grain structure (macro structure)
- Framework Instantiation Patterns
  - Some design patterns couple framework variation points and application code (*framework instantiation patterns*)
- Design patterns are “mini-frameworks” themselves
  - Since they contain common structure for many applications
  - Design patterns are used in frameworks (that's how they originated)
  - Hence, this course must also say many things about frameworks

Programming Pattern (Idiom, LDDP)

- An *idiom* is a pattern whose form is described by means of programming language constructs.
- Example: The C idiom of check-and-returns for contract checking
  - The first book on idioms was Coplien's Advanced C++ Programming Styles and Idioms (1992), Addison-Wesley

```
public void processIt (Document doc) {
  // check all contracts of processIt
  if (doc == null) return;
  if (doc.notReady()) return;
  if (internalDoc == doc) return;

  // now the document seems ok
  internalProcessIt(doc);
}
```

```
private void internalProcessIt (Document doc) {
  // no contract checking anymore

  // process the document immediately
  walk(doc);
  print(doc);
}
```

Workaround

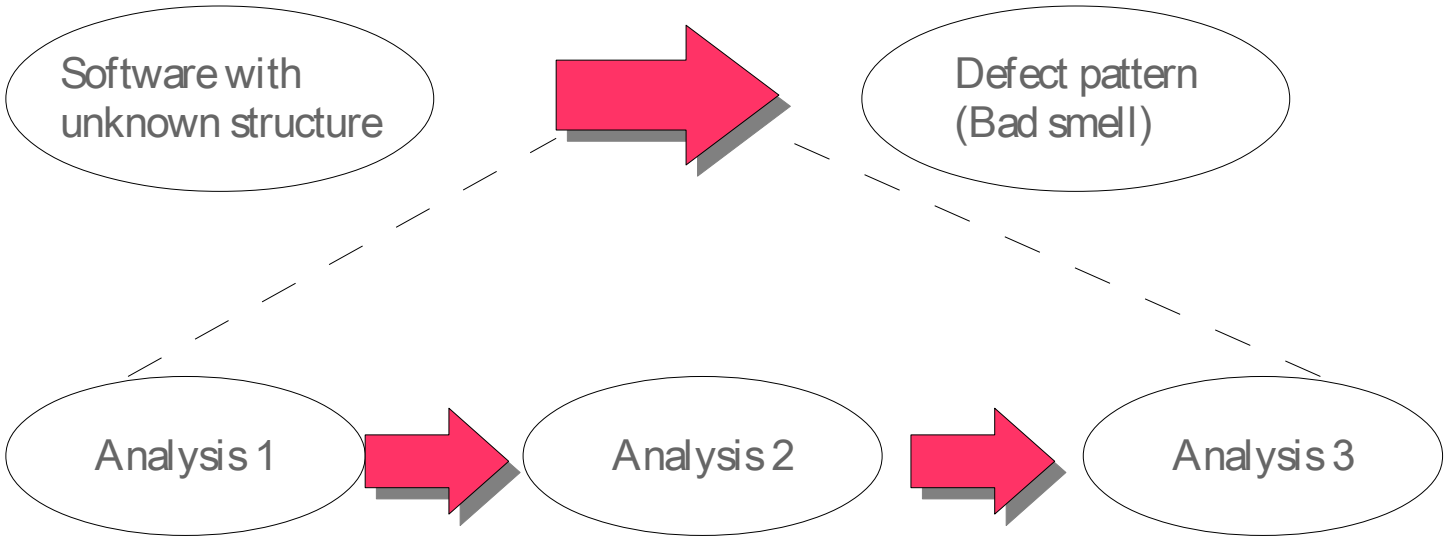
- A *workaround* is an idiom that works around a language construct that is not available in a language
- Example: Simulating polymorphism by if-cascades

```
public void processIt (Document doc) {
  // Analyze type of document
  if (doc->type == Text)
       processText((Text)doc);
  else i f (doc->type == Figure)
       processFigure((Figure)doc);
  else
       printf("unknown subtype of document");
}
```

```
void processText(Text t) {..}
void process Figure(Figure f) {..}
```

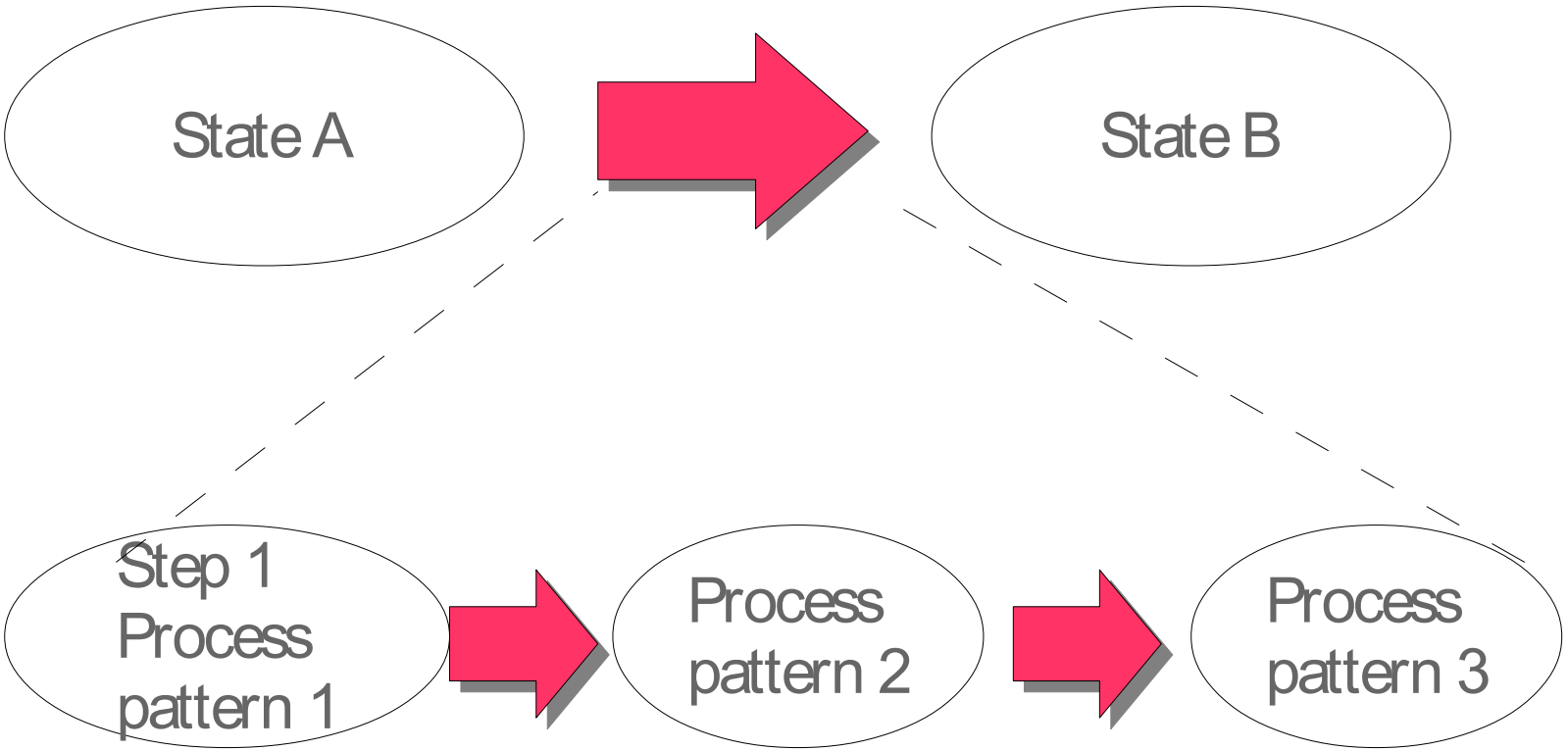
Antipatterns (Defect Patterns)

- Software can contain bad structure
  - No modular structure, only procedure calls
  - If-cascades instead of polymorphism
  - Casts everywhere
  - Spaghetti code (no reducible control flow graphs)
  - Cohesion vs Coupling (McCabe)



Process Patterns

- **Process patterns** are solutions for the process of making something



Process Patterns

- When process patterns are automatized, they are called workflows
- Workflow management systems enable us to capture and design processes
  - ARIS on SAP
  - Intentia
  - FlowMark (IBM)
  - and many others
- Example:
  - “Delegate-Task-And-Resources-Together”

Organizational Patterns

- Two well-known organizational patterns are
  - Hierarchical management
    - In which all communication can be described by the organizational hierarchy
  - Matrix organization
    - In which functional and organizational units talk to each other

In This Course

- We will mainly treat design patterns
  - Conceptual patterns
  - Architectural patterns
  - Framework instantiation patterns

Pattern Languages: Patterns in Context

- According to Alexander, patterns occur in *pattern languages*
  - A set of related patterns for a set of related problems in a domain
  - Similar to a natural language, the pattern language contains a vocabulary for building artifacts
- A structured collection of patterns that build on each other to transform forces (needs and constraints) into an architecture [Coplien]
  - Patterns rarely stand alone. Each pattern works in a context, and transforms the system in that context to produce a new system in a new context.
  - New problems arise in the new system and context, and the next “layer” of patterns can be applied.
- We will treat one larger example, the TAM pattern language

Design Patterns and Frameworks

General Notes

Elements of the Course

- Lecturing
  - Do not miss one, they should give you a short and concise overview of the material
- Reading
- Exercise sheets
  - You have one week to solve them on your own
  - After that, solutions will be explained in the exercise seminars

- http://st.inf.tu-dresden.de → Studies → Courses → Design Patterns and Frameworks
- http://st.inf.tu-dresden.de/teaching/dpf
- News are announced on this course page – check regularly!

Reading Along the Lectures

- Unfortunately, the course is not covered by any book
  - The GOF book is a prerequisite for the course, not it's contents!
- You have to read several research papers, available on the Internet
  - Marked by “Mandatory Literature (To Be Read)”
- Secondary Literature is non-mandatory, but interesting reading.
  - Can be done during the course

Literature (To Be Read)

- During the course, read the following papers, if possible, in sequential order. See also literature web page.
  - Every week, read about 1 paper (3-4h work)
- Start here:
  - A. Tesanovic. What is a pattern? Paper in Design Pattern seminar, IDA, 2001. Available at home page.
  - Brad Appleton. ***Patterns and Software: Essential Concepts and terminology.***
    http://www.sci.brooklyn.cuny.edu/~sklar/teaching/s08/cis20.2/papers/appleton-patterns-intro.pdf
    Compact introduction into patterns.
  - K. Beck, J. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, J. Vlissides. ***Industrial Experience with Design Patterns.*** Int. Conference on Software Engineering (ICSE) 1996. http://dl.acm.org/citation.cfm?id=227747

Literature (To Be Read)

- [GOF, Gamma] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns. Addison-Wesley 1995. Standard book belonging to the shelf of every software engineer.
  - Prerequisite for the course
  - The book is called GOF (Gang of Four), due to the 4 authors
- Alternatively to GOF can be read:
  - Head First Design Patterns. Eric Freeman & Elisabeth Freeman, mit Kathy Sierra & Bert Bates.O'Reilly, 2004, ISBN 978-0-596-00712-6
  - German Translation: Entwurfsmuster von Kopf bis Fuß. Eric Freeman & Elisabeth Freeman, mit Kathy Sierra & Bert Bates. O'Reilly, 2005, ISBN 978-3-89721-421-7
- Alternatively, available at home page. If you have already studied GOF, do not read these. These paper stem from a Design Pattern seminar at Linköpings Universitet, IDA, 2001:
  - T. Panas. Design Patterns, A Quick Introduction. (on Composite, Visitor)
  - Veaceslav Caisin. Creational Patterns.
  - P. Pop. An overview of the automation of patterns.

Secondary Literature

- M. Fowler. Refactoring. Addision-Wesley, 1999.
- D. Riehle, H. Züllighoven, Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems, 1996
  http://dirkriehle.com/computer-science/research/1996/tapos-1996-survey.html
- D. Garlan, R. Allen, J. Ockerbloom. Architectural mismatch – or why it is so hard to build systems out of existing parts. Int. Conf. On Software Engineering (ICSE 95).
  http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=469757
- A. Abel. Design Pattern Relationships and Classification. Paper in Design Pattern seminar, IDA, 2001. Available at home page.
- T. Pop. Multi-Paradigm Design. Paper in Design Pattern seminar, IDA, 2001. Available at home page.

Other Literature

- T. Reenskaug, P. Wold, O. A. Lehne. Working with objects Manning.
  - The OOram Method, introducing role-based design, role models and many other things. A wisdom book for design. Out of print. Preversion available on the internet at
    http://heim.ifi.uio.no/~trygver/1996/book/WorkingWithObjects
- K. Beck. Extreme Programming. Addison-Wesley.
- H. Allert, P. Dolog, W. Nejdl, W. Siberski, F. Steimann. *Role-Oriented Models for Hypermedia Construction – Conceptual Modelling for the Semantic Web.*
  http://people.cs.aau.dk/~dolog/pub/ht2003.pdf

Please, Please Be Aware – There Will Be Pain!

- **This course is a research-oriented course**
- **It treats rather advanced material**
- **No book exists on all of that at all**
  - GOF only prerequisite
  - Please, collaborate! Read the articles, ask questions!
  - Do the exercise sheets
- Warning: **The oral exams can only be done if you have visited all lectures and solved all exercise sheets**
  - **The GOF Book alone is not sufficient**
- **Learn continuously!**
- **Be aware: you have not yet seen larger systems**
  - Middle-size systems start over 100KLOC

Learning Java with INLOOP

- If you don't know Java, yet...
- In our basic course on software technology, we have published a web-based self-learning system for Java
  - into which you can enter Java programs
  - which tests style and syntax of the programs
  - and runs a test suite against your program
- INLOOP gives you feedback about your programming abilities in Java

- INLOOP is an opportunity for you, please use it!

https://inloop.inf.tu-dresden.de/

The Positive Side

- If you follow carefully, you will discover an exciting world of beauty in software
- If you know all the patterns of the course, you will be a much better software engineer than the standard programmer
  - Most of the work has been discovered in the last 10 years, and is unknown to many programmers
- You will also be a much better manager,
  - because patterns and frameworks teach you how to master large systems and product lines in your company
- The gain is worthwhile the pain!

Oral and/or Written Exam/s

- There will be two weeks for oral exams
- Somewhen between February and March 2020
- The exam weeks will be announced early January.
- There will be no oral exams during the lecture period of summer term 2020. The next examination period will be August/September 2020.

- You can enroll by sending an Email to st-exams@mailbox.tu-dresden.de
- In your mail you need to indicate:
  - The preferred time (e.g., end of February) and/or times when you are away
  - The module you want to take the exam for (e.g., BAS3)
  - Your course of study (e.g., Master MINF)

The End