Design Patterns and Frameworks
1) Introduction

Prof. Dr. U. Aßmann

Chair for Software Engineering

Faculty of Informatics

Dresden University of Technology

WS 11-0.1, 10/8/11

1) History and Introduction

2) Different classes of patterns

3) Where can patterns be used?

Literature (To Be Read)

- ► Start here: A. Tesanovic. What is a pattern? Paper in Design Pattern seminar, IDA, 2001. Available at course home page.

- ► Alternatively: GOF: Introduction.

- ► Brad Appleton. Patterns and Software: Essential Concepts and terminology. http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html Compact introduction into patterns.

- ► http://www.hillside.net/plop/pastconferences.html

Prof. Uwe Aßmann, Design Patterns and Frameworks

Secondary Reading

► D. Riehle, H. Zülinghoven, Understanding and Using Patterns in Software Development. Theory and Practice of Object Systems 2 (1), 1996. Explains different kinds of patterns.
http://citeseer.ist.pst.edu/riehle96understanding.html

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

Prof. Uwe Aßmann, Design Patterns and Frameworks

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 a "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 compositum, 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

|	a	b	c
x	60	30	10
y	50	30	20
z	80	10	10

Views

a=50%
b=30%
c=20%
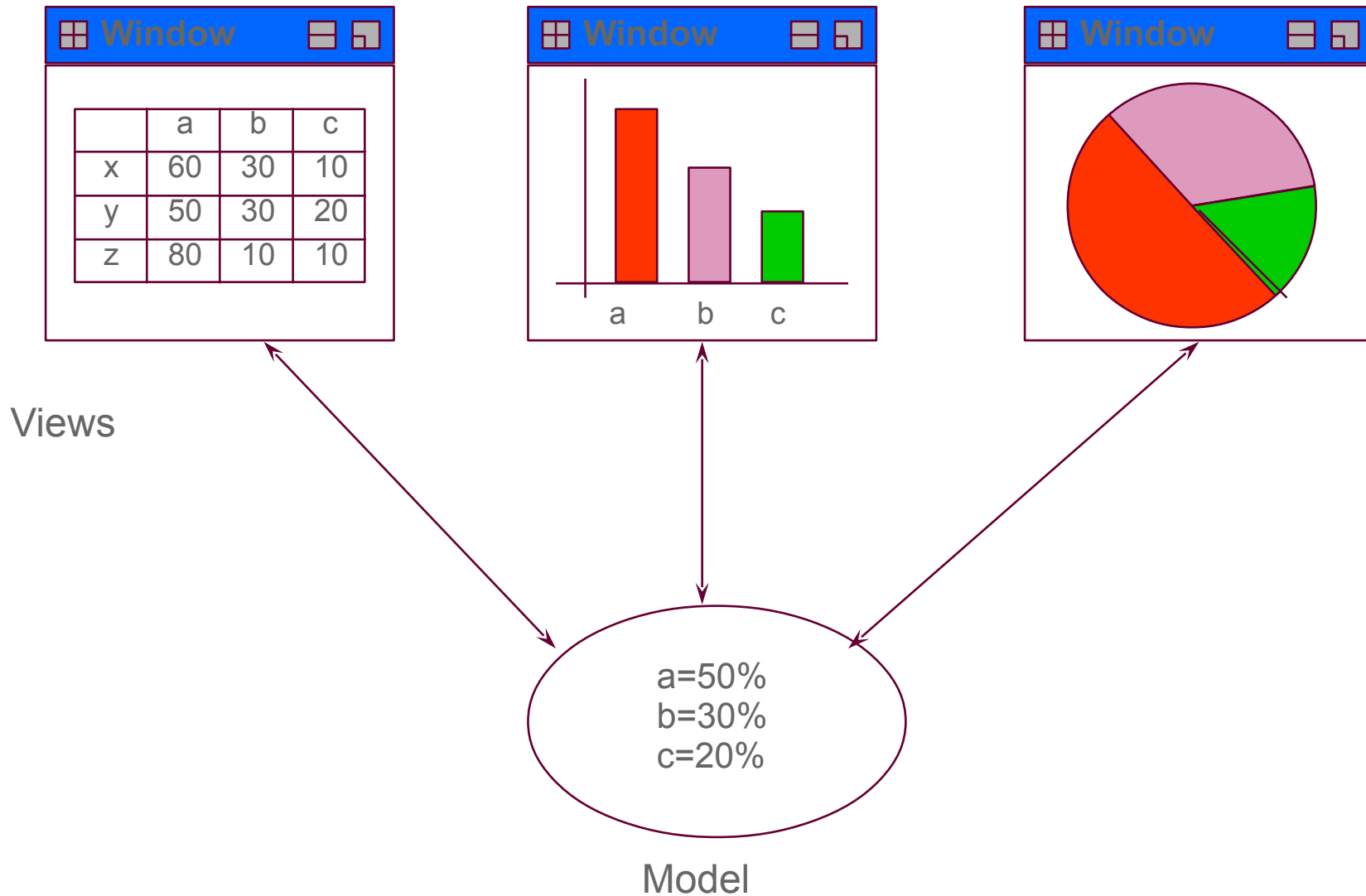
Model

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, Compositum 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

forces

"bad smell" ⟶ "good smell"
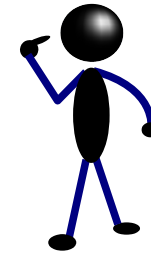
Structure for Design Pattern Description (GOF Form)

- 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

Purpose Design Pattern

▶ 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

Standard Incentives For Using Patterns

- ▶ Easy System
  - System structure
  - Easy communication
  - Easy protocols
- ▶ Easy Testability
  - Null object
  - Static preprocessing
- ▶ Easy Evolution
- ▶ Easy Documentation
- ▶ Easy Reuse!!
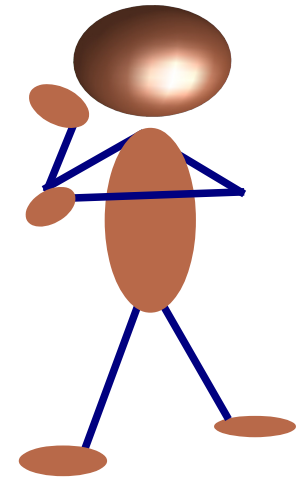
Different Kinds of Patterns

What is a Pattern?

- ▶ There is no "the pattern"
- ▶ At least, research is done in the following areas:
  - Conceptual patterns
  - Design Patterns
    - Different forms
  - Antipatterns
  - Implementation patterns (programming patterns, idioms, workarounds)
  - 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]

Prof. Uwe Aßmann, Design Patterns and Frameworks

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"

Prof. Uwe Aßmann, Design Patterns and Frameworks

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]

▶ Question: what is here the "abstraction from the concrete form"? (in terms of Riehle/Züllighoven)

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 style (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)

▶ Question: what is here the "abstraction from the concrete form"?

Prof. Uwe Aßmann, Design Patterns and Frameworks
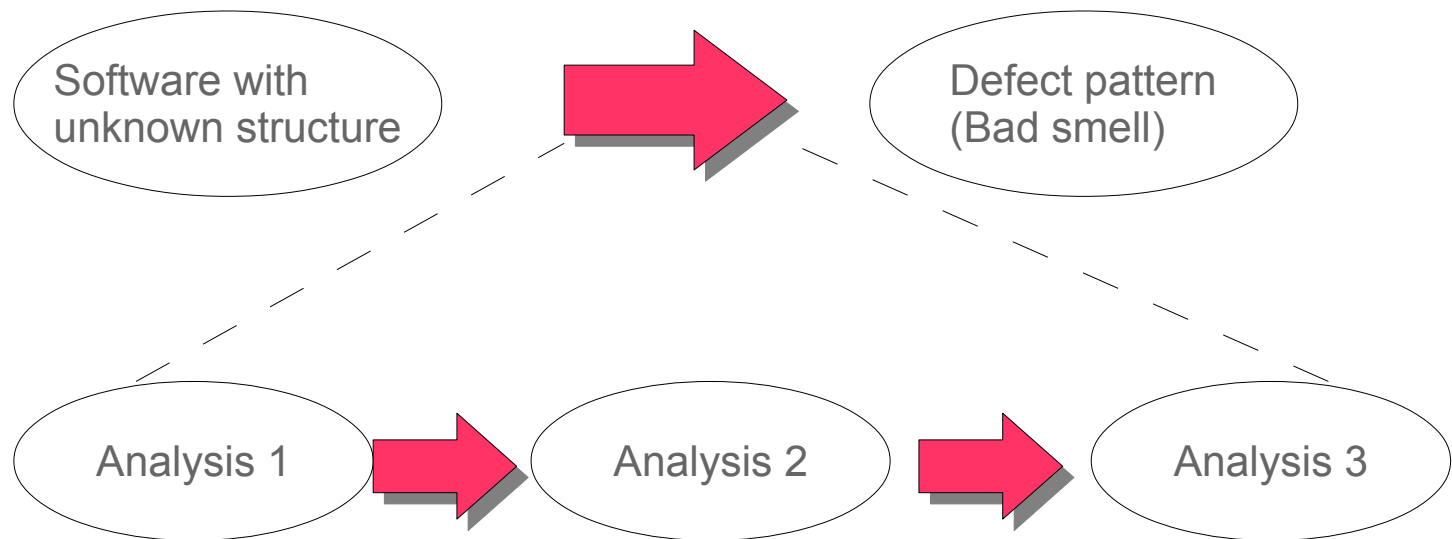
Refactorings Transform Antipatterns Into Design Patterns

▶ A DP can be a goal of a refactoring

▶ Structurally, a **refactoring** is an operator on the code (a metaprogram)

■ Semi-formal: Fowler's book on refactorings uses graph rewrite rules to indicate what the refactorings do

■ Formal: Refactorings can be realized in program transformation and metaprogramming libraries and tools

· Recoder (recoder.sf.net) is such a tool

· Eclipse, Netbeans contain refactorings

Prof. Uwe Aßmann, Design Patterns and Frameworks
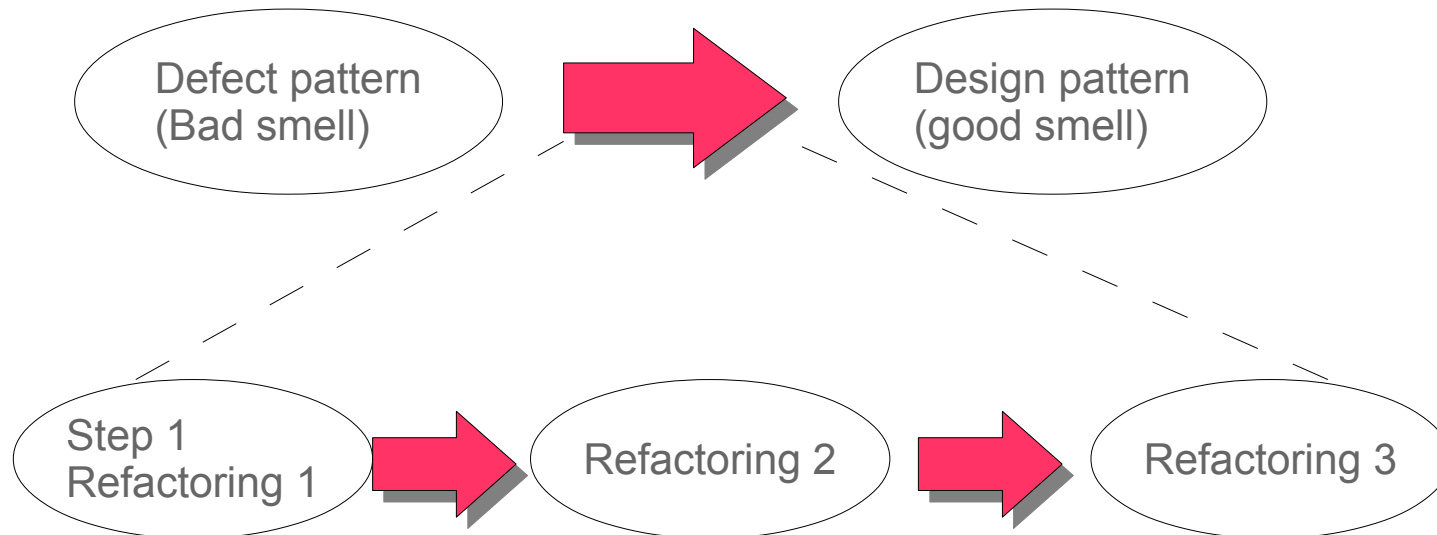
Process Patterns

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

Prof. Uwe Aßmann, Design Patterns and Frameworks
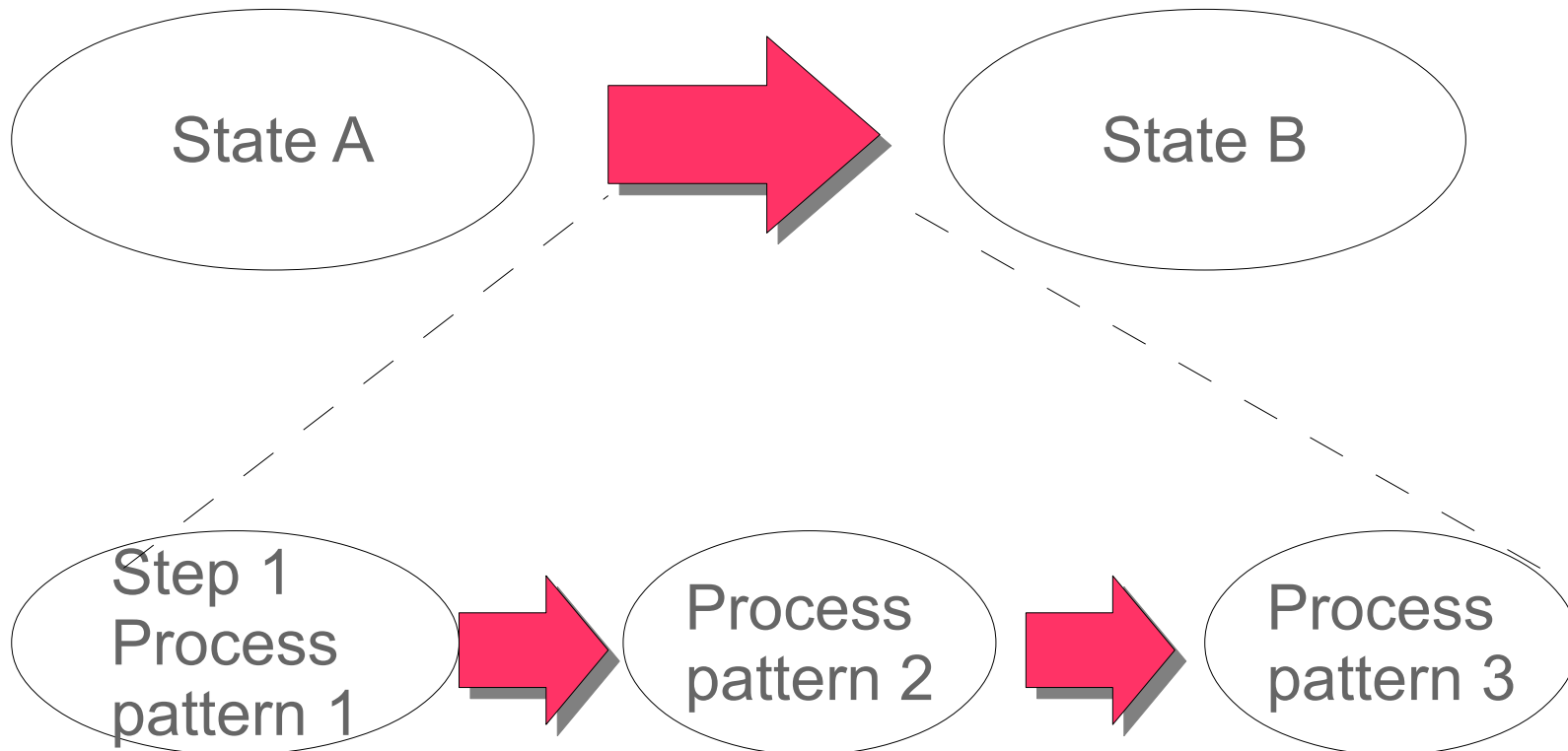
Process Patterns

▶ When process patterns are automized, they are called workflows

▶ Workflow management systems enable us to capture and design processes

- ARIS on SAP

- Intentia

- FlowMark (IBM)

- and many others

▶ Examples:

- "Work-and-Let-Be-Granted"

- "Delegate-Task-And-Resources-Together"

▶ Question: what is here the "abstraction from the concrete form"?

Reengineering Patterns

▶ Also in the software reengineering process, common (process) patterns can be identified

▶ Examples

  ▪ "Read-All-Code-In-One-Hour"

  ▪ "Write-Tests-To-Understand"

▶ S. Demeyer, S. Ducasse, O. Nierstrasz. Object-oriented Reengineering Patterns. Morgan-Kaufmann, 2003

▶ Question: what is here the "abstraction from the concrete form"?

Prof. Uwe Aßmann, Design Patterns and Frameworks

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

► Question: what is here the "abstraction from the concrete form"?

Prof. Uwe Aßmann, Design Patterns and Frameworks

In This Course

► We will mainly treat design patterns

- Conceptual patterns

- Architectural patterns

- Framework instantiation patterns

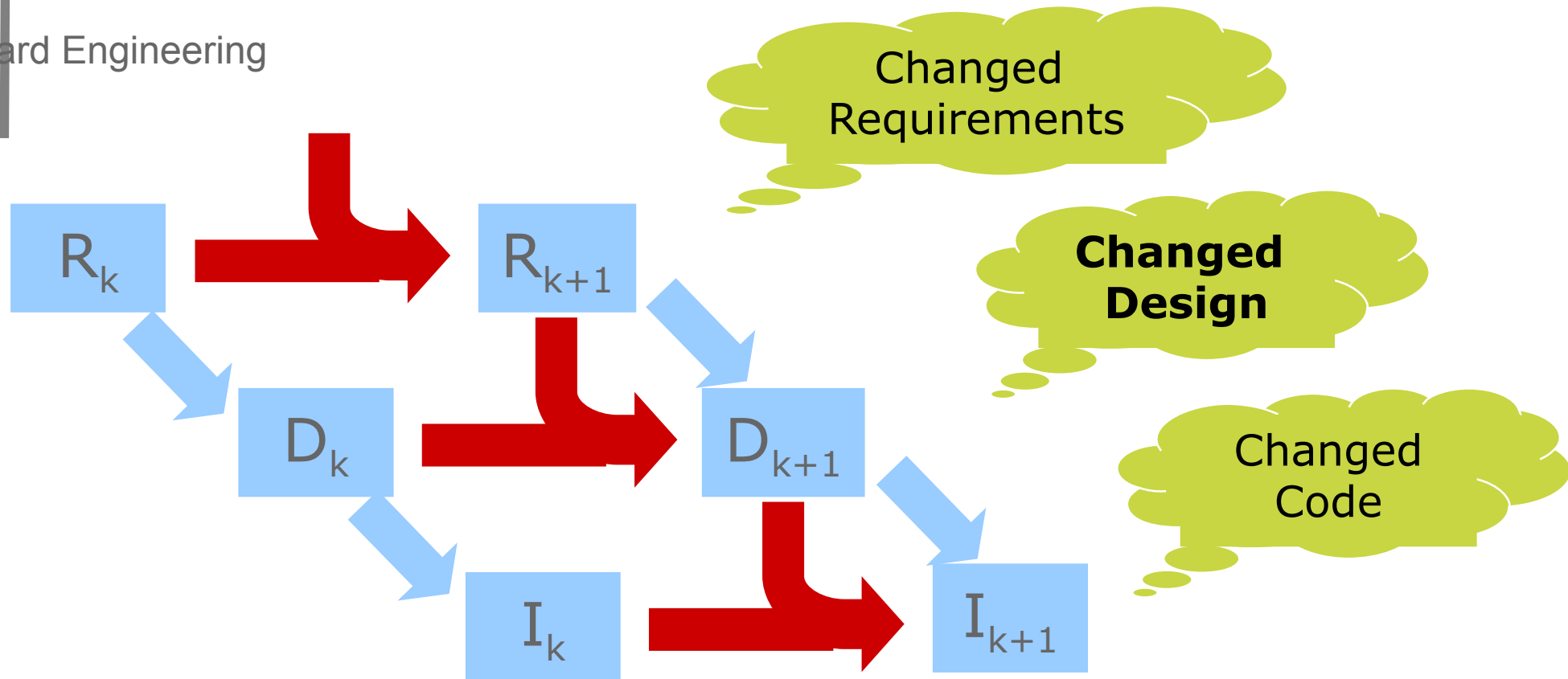- Very few LDDP and workarounds

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 artefacts

▶ 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.

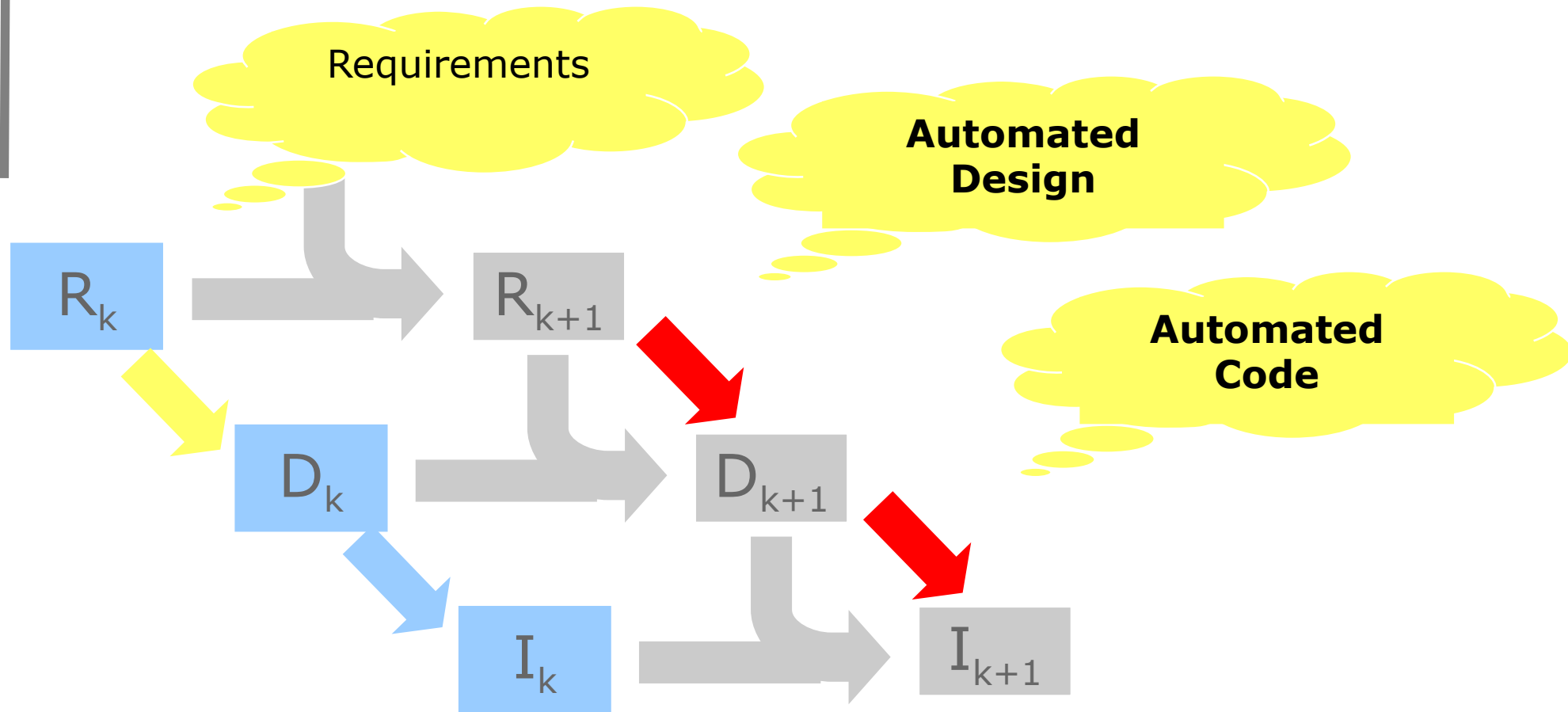▶ We will treat one larger example, the TAM pattern language

Where do Patterns Occur in Software Development?

Automated Design (CASE)

Requirements

Automated Design
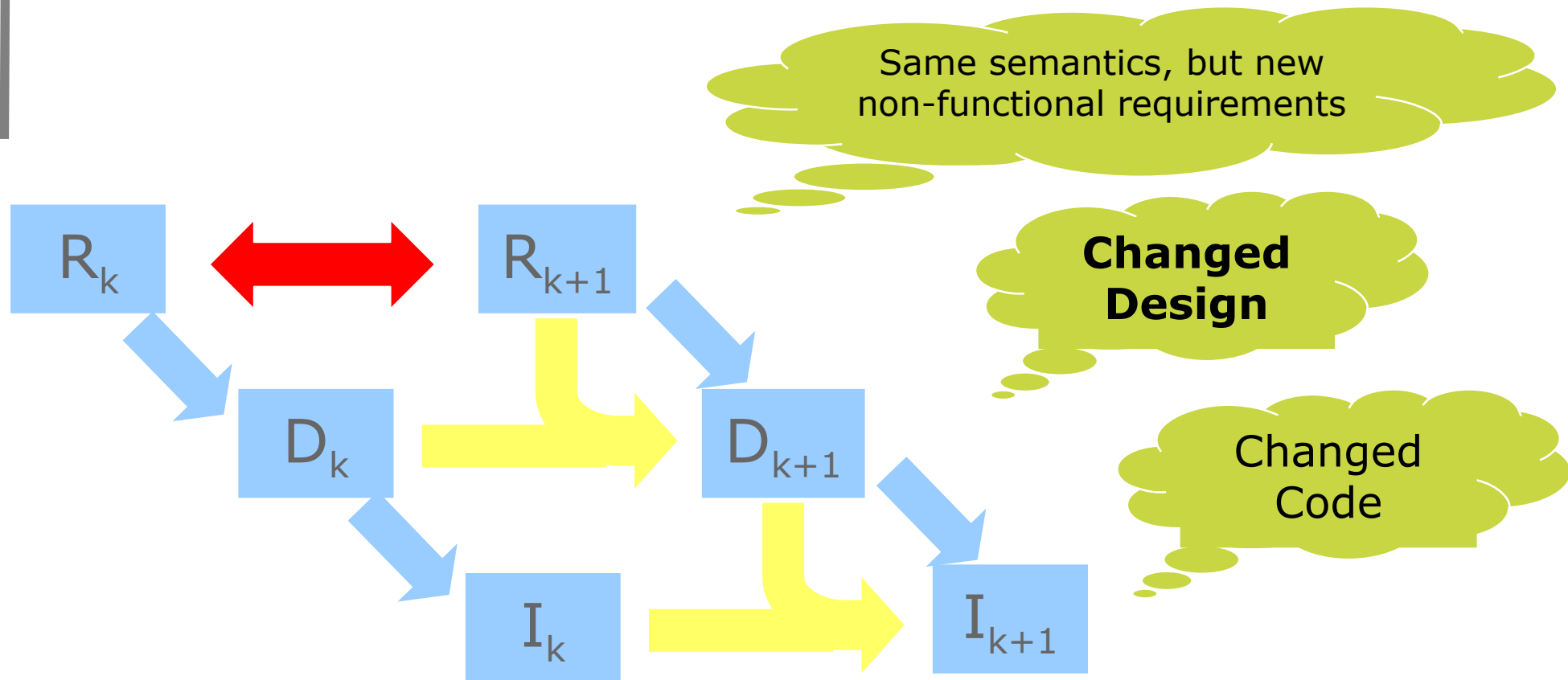
Automated Code

R_k → R_{k+1}

D_k → D_{k+1}

I_k → I_{k+1}

Support by CASE tools to a limited extend possible
Tools generate structure of design patterns into the code
(e.g., Together)

Prof. Uwe Aßmann, Design Patterns and Frameworks

Program Refinement

Same semantics, but new non-functional requirements

Changed Design

Changed Code

R_k ↔ R_{k+1}

D_k → D_{k+1}

I_k → I_{k+1}

Needs new non-functional requirements. For instance, optimization patterns speed applications up; adapters and bridges can be used for checking consistency

Automated Software Evolution (XP-like)

R_k → R_{k+1}

D_k → D_{k+1}

I_k → I_{k+1}

Changed Requirements

Changed Design

Changed Code
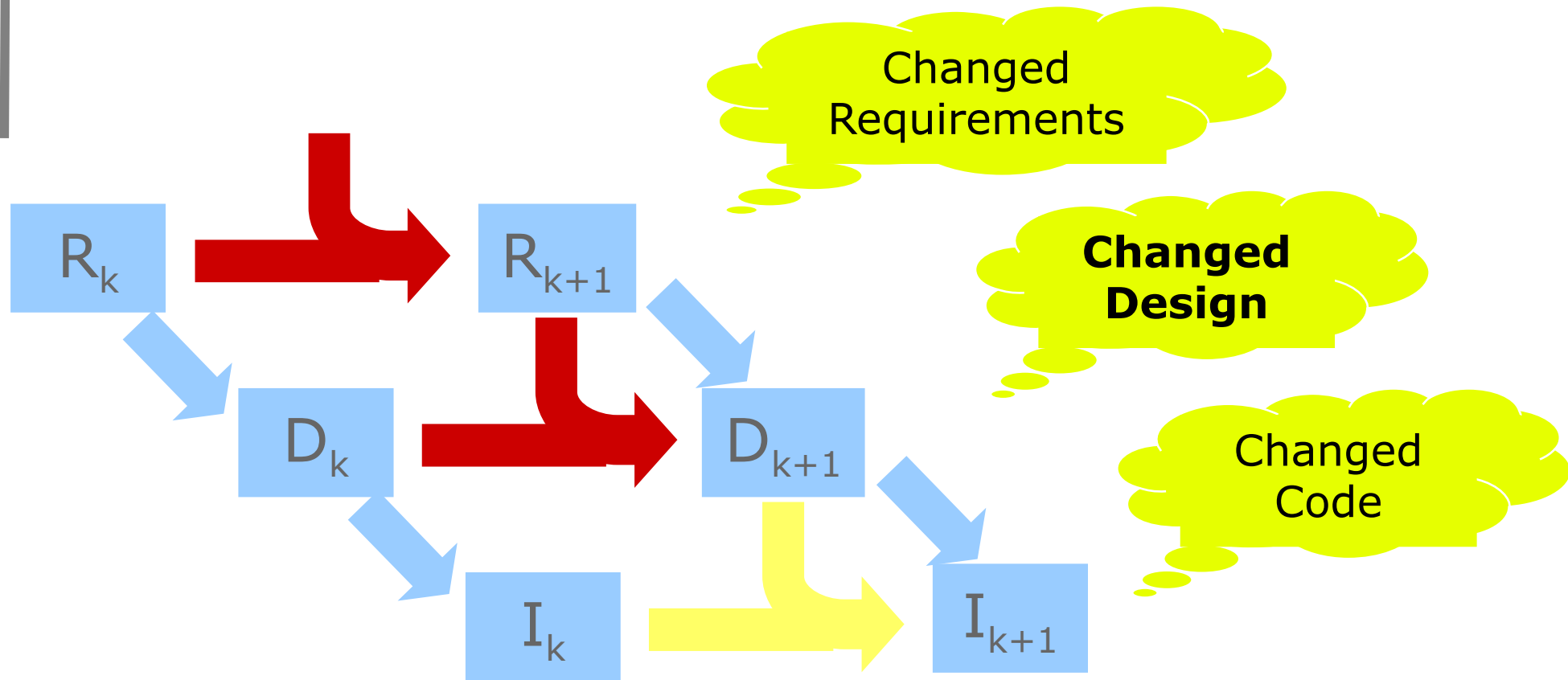
In XP, many adaptations can be automized by employing refactoring tools

Reengineering
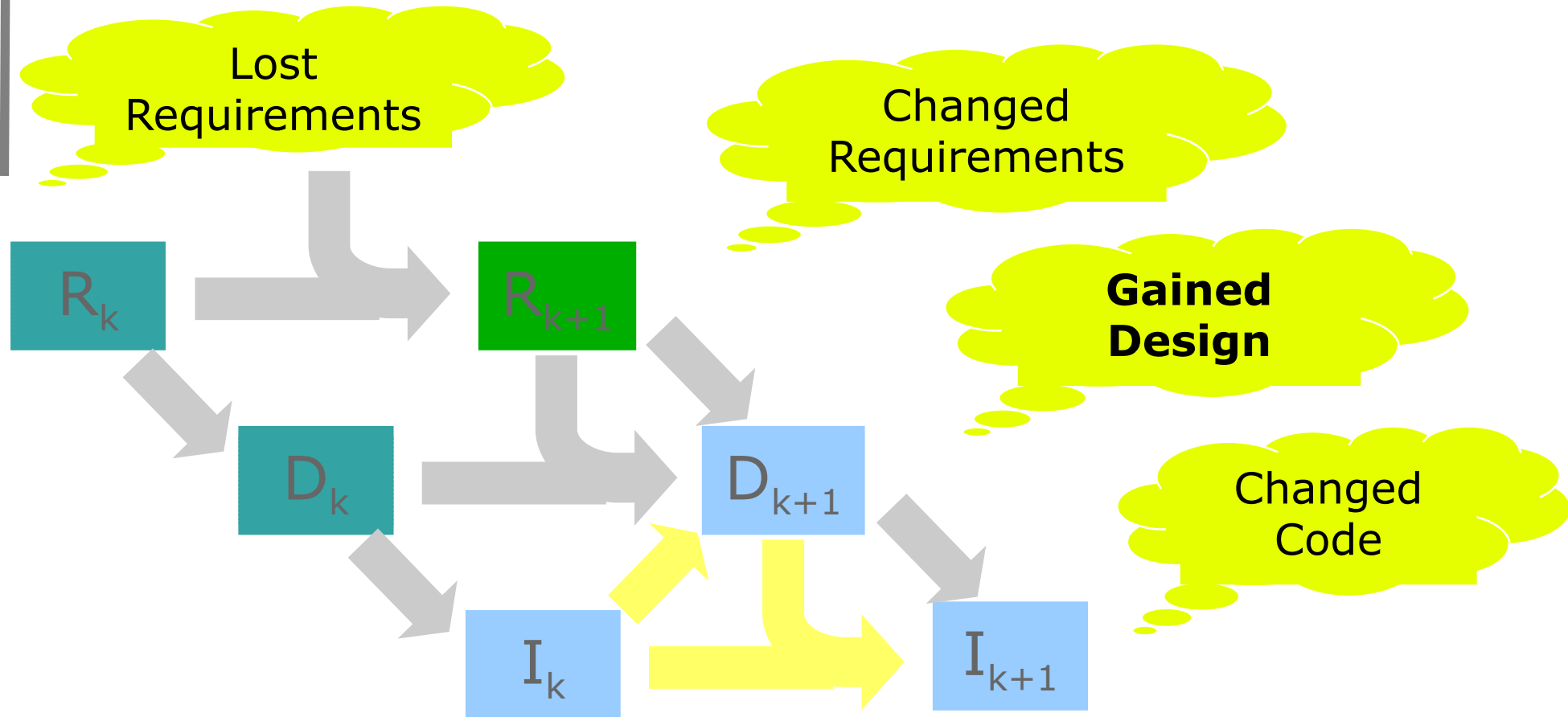
Automatic and semi-automatic recognition of design patterns is a hot research topic

The End