Patterns: Capturing Preventing and Reacting Facets of the Systems Behavior

Prof. dr. Herwig Mannaert

Computation World, Athens, 2009

What does Wikipedia say ?

- A **design pattern** in architecture and computer science is a formal way of documenting a solution to a design problem in a particular field of expertise. The idea was introduced by the architect Christopher Alexander (1977) in the field of architecture and has been adapted for various other disciplines, including computer science, pedagogical, and interaction design.
- A pattern must explain why a particular situation causes problems, and why the proposed solution is considered a good one.

What does Wikipedia say ?

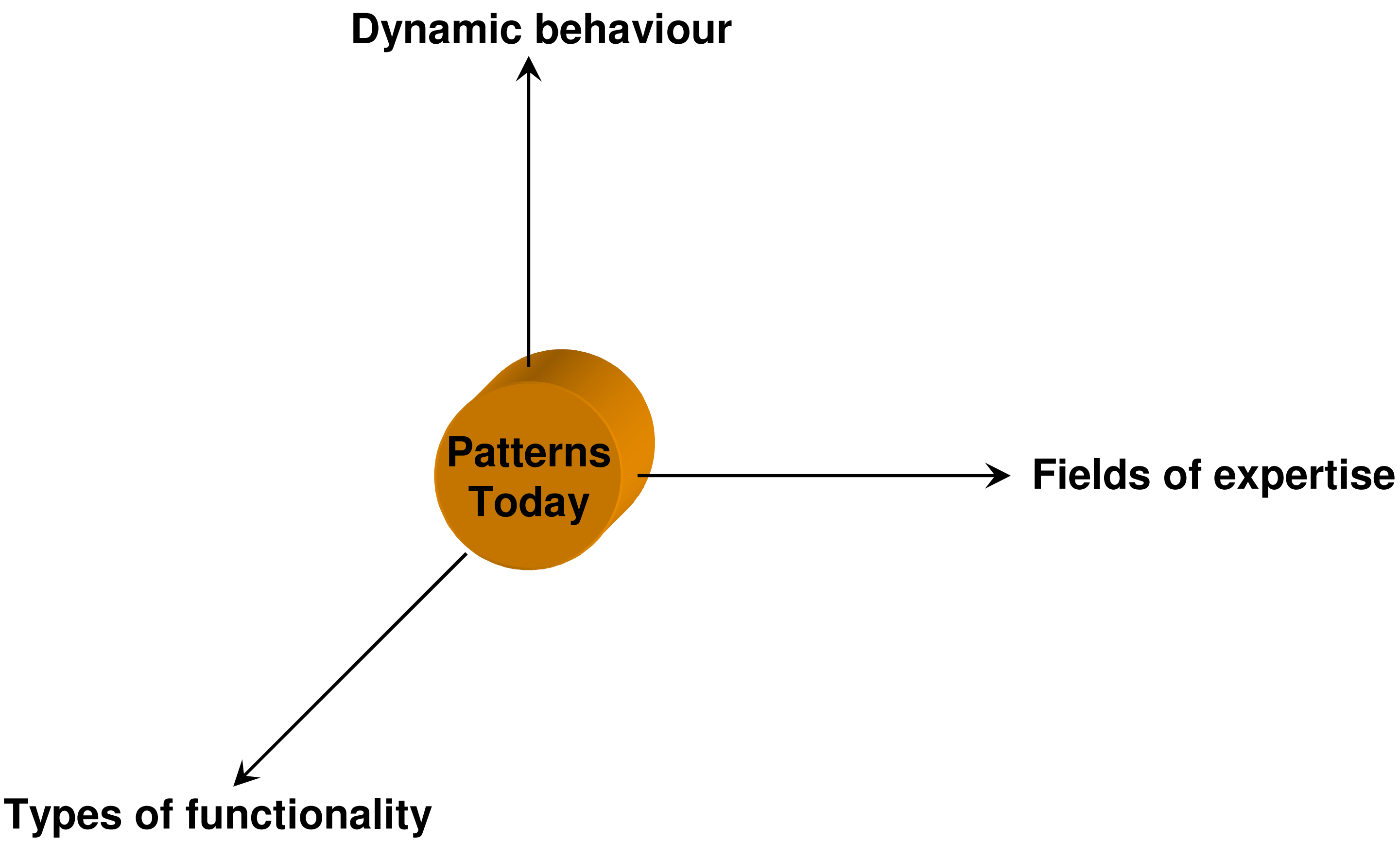
- **Pattern** in architecture is the idea of capturing architectural design ideas as archetypal and reusable descriptions.
- A pattern would not tell the designer how many windows to put in the room; instead, it would propose a set of values to guide the designer toward a decision that is best for their particular application.
- In software engineering, a **design pattern** is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

What Wikipedia tells us

- Patterns are heuristics, rules of thumb
- Documenting patterns is basically knowledge engineering in a particular field of expertise
- Pattern fundamentals are discipline agnostic
- The number of disciplines or fields that use patterns is currently quite limited
- Patterns are often limited to design patterns

➔ Seems a major interdisciplinary research area

Pattern Research Possibilities


Dynamic behaviour

Patterns Today

Fields of expertise

Types of functionality

Patterns: Type of Functionality

- Identification of behaviour patterns that may lead to malfunctioning, for instance due to data mining of past incidents:
  - possible examples are irregular equipment monitoring data, leading to cascades of outages
  - possible information system malfunctionings that lead through trouble reports to positive feedback mechanisms causing eventual system breakdown
- Designing proactive repair patterns that cope with the described behavior patterns and may avoid the corresponding cascade effects

Patterns: Fields of Expertise

- Studying and describing patterns in other disciplines or fields of expertise.
- Such a pattern at the business level, could:
  - lead to patterns for 'a business transaction', or to business process or even enterprise patterns
  - give business process descriptions a more deterministic basis
  - create a more straightforward mapping of business patterns to the software patterns

➔ contribute to service productivity gains

Pattern: Dynamic Behaviour

- Introducing a substantiated temporal dimension in design patterns: extending static design patterns to 'evolvable design patterns�much, i.e. patterns that are designed to allow for evolving systems along certain degrees of freedom:
  - in software where evolvability is a crucial and problematic issue, but also in many other areas
  - in large-scale designs like microchips or airplanes where the evolvability is also very limited **even** on the drawing table
- Studying the combinatorial or cascade effects that are caused by applying changes to design patterns, as these cascade effects are clearly a major inhibitor towards evolvability of systems and/or their designs

Pattern Research

- Do patterns possibly constitute a major initerdisciplinary research area ?

- What are the major facets we should study ?

IARIA Working Group on Computation

Expert Panel on

Patterns: Capturing, Preventing, and Reacting Facets of the Systems Behavior

Dr. Harald Gjermundrod

University of Nicosia

UNIVERSITY OF NICOSIA
ΠΑΝΕΠΙΣΤΗΜΙΟ ΛΕΥΚΩΣΙΑΣ

Outline

- A Little About Me
- The Many Facets of Systems
- Patters in Systems
- Summary and Questions

A Little About Me

- Worked on a monitoring and control framework for the electric power grid for my PhD
  - Developed a prototype of this framework which is being field tested in the US
  - Condensation function mechanism used by application architect/ developer to move application logic into the middleware
- During post-doc worked on Grid Computing
  - Middleware tester (EGEE Project)
  - Software developer for a client interface to monitor and manage a local Grid site (g-Eclipse Project)
- Joined the University of Nicosia as a research faculty fall of 2008
- Summary: Young hands-on researcher

The Many Facets of Systems

- Information/content provided by a systems can be
  - Tightly controlled; information in and out are tightly controlled
  - Loosely controlled; little if no control (Internet)
- Structure of the systems (including the application layer)
  - Static in that once the system is in operation it doesn’t change
  - Dynamic in that users/services can come and go
- Type of system
  - Large operates in the wide-area setting
  - Small operates in the local-area setting
  - Private/limited access system
  - Public access system
- Purpose of the system
  - Control critical infrastructures or life support systems
  - Operational for business or entertainment purpose

Patterns in Systems

- Can use patterns for the architecture
- Can use patterns for the various building blocks and how they are put together
- Once the system is operation what patters can we look for and for what purpose
  - Patterns in the usage of the system or flow of information at the application layer
  - Patters in the availability of behavior of the IT infrastructure itself
- When are the patters specified
  - During the implementation time, designed into the system
  - During the tuning phase of the systems
  - Dynamically while the system is operational

Patterns in Systems (Cont.)

- Will any large system develop its own "personality", i.e. it will be unique in the way it behaves
- The only ones that know how it behaves and how it reacts to tweaking are the system administrators
- Can any best-practice be defined for what patterns to look for and how to react or is each system unique
  - The specific patterns (with respect to input and parameter setting) to look for have to be tailored to each specific system
  - Is it possible for a large system to operate automatically or will there always be a human-in-the loop.

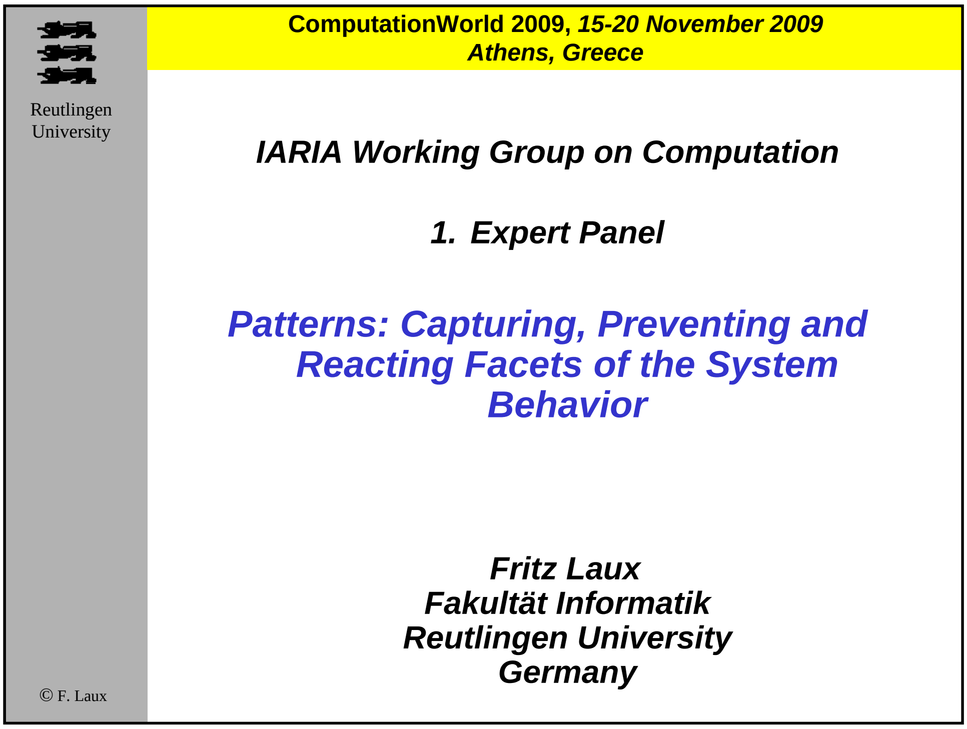
Facets of the Systems Behavior Questions

- Capturing
  - Who should capture it and define what to capture? system users, system administrators, "intelligent" monitoring systems
  - What are we capturing? Behavior of the IT infrastructure, application running on top of the system, or a hybrid (are they coupled)
  - Security and privacy issues of what is captured. Might not want to publish or make available information that the system is vulnerable
- Preventing
  - What are we preventing?
    - System crash, system overload, loose of control, leak of sensitive information
- Reacting
  - Do we want to make a survivable system w.r.t. intentional and accidental "failures"
  - Can we "trust" automatic reaction? Depending on the system, maybe

Selecting the Applicable Patterns?

- Can any commonality be abstracted and patterns be proposed?

- Some can probably be found for systems belonging in the same category
  - Systems for critical infrastructures
  - The Internet itself may have become a critical infrastructure itself
  - Non critical systems
    - Content management/distribution system
    - Grids, clouds, cooperate management systems
    - Banking systems?



Reutlingen University

ComputationWorld 2009, *15-20 November 2009*
Athens, Greece

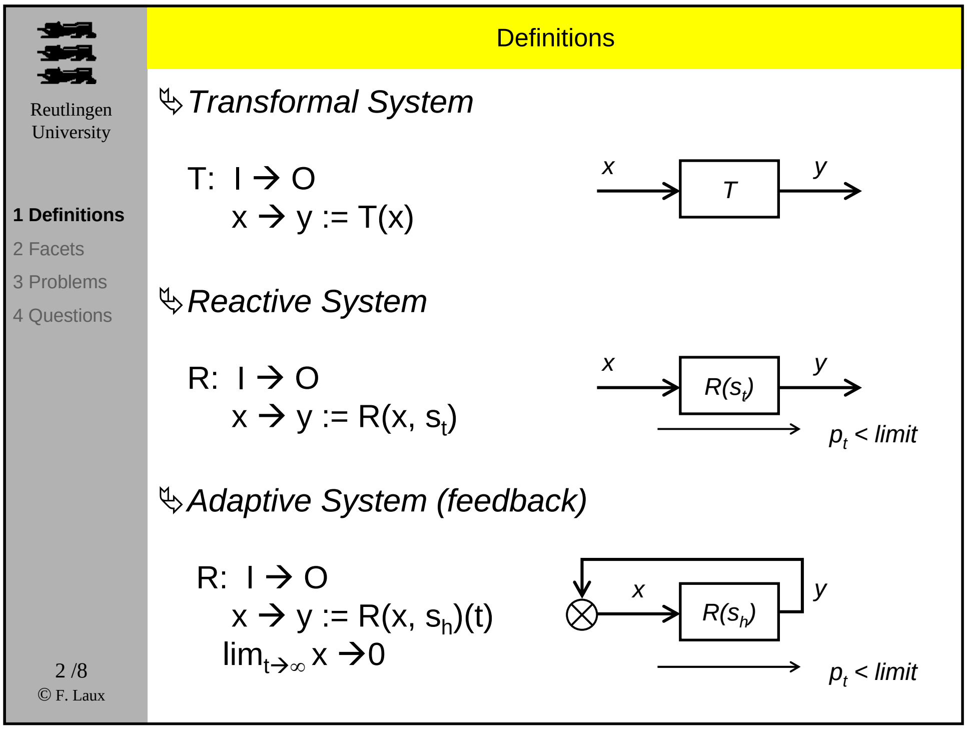
IARIA Working Group on Computation

1. Expert Panel

Patterns: Capturing, Preventing and Reacting Facets of the System Behavior

Fritz Laux
Fakultät Informatik
Reutlingen University
Germany

© F. Laux



Reutlingen University

1 Definitions
2 Facets
3 Problems
4 Questions

Definitions

- *Transformal System*

T: I → O
$x \rightarrow y := T(x)$

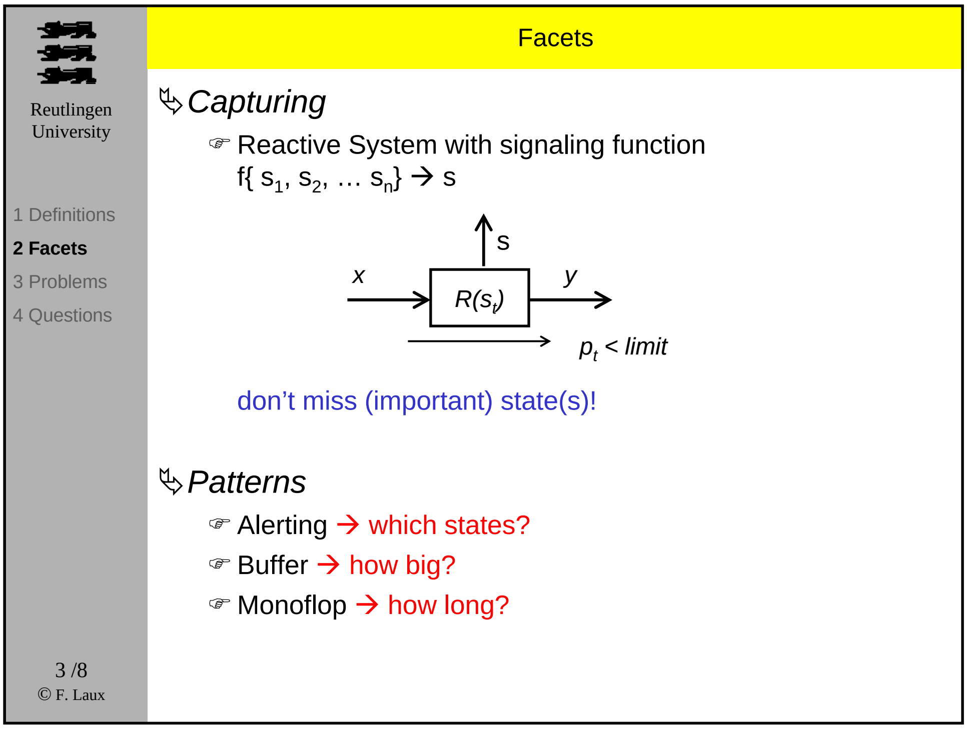
- *Reactive System*

R: I → O
$x \rightarrow y := R(x, s_t)$

- *Adaptive System (feedback)*

R: I → O
$x \rightarrow y := R(x, s_h)(t)$
$\lim_{t\rightarrow\infty} x \rightarrow 0$

2 /8
© F. Laux



Reutlingen University

1 Definitions
2 Facets
3 Problems
4 Questions

Facets

Capturing

- Reactive System with signaling function
  $f\{ s_1, s_2, \dots s_n\} \rightarrow s$

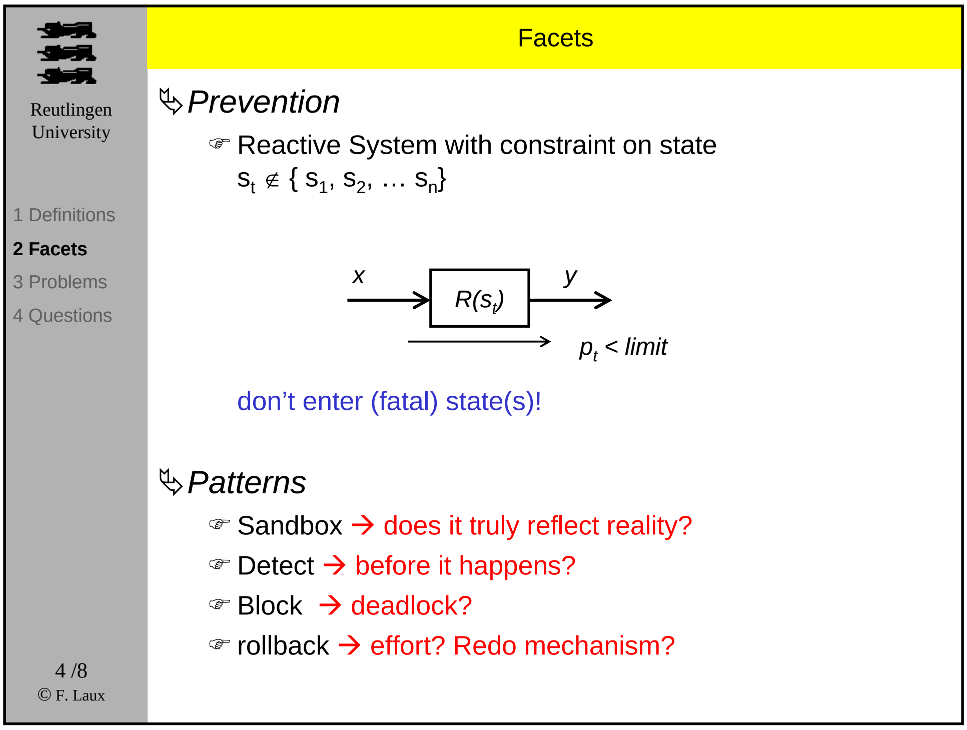
s
x
$R(s_t)$
y
p_t < limit

don't miss (important) state(s)!

Patterns

- Alerting → which states?
- Buffer → how big?
- Monoflop → how long?

3 /8
© F. Laux



Reutlingen University

1 Definitions
2 Facets
3 Problems
4 Questions

Facets

Prevention

- Reactive System with constraint on state
  $s_t \notin \{ s_1, s_2, \dots s_n\}$

x
$R(s_t)$
y
p_t < limit

don't enter (fatal) state(s)!

Patterns

- Sandbox → does it truly reflect reality?
- Detect → before it happens?
- Block → deadlock?
- rollback → effort? Redo mechanism?

4 /8
© F. Laux

Reutlingen University

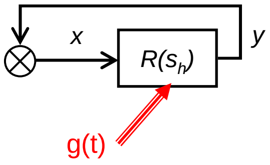
1 Definitions
2 Facets
3 Problems
4 Questions

Facets

Evolvable

- ☞ Adaptable System $R(x, s_h)(t) \rightarrow 0$

  with moving target $R(x, s_h)(t) \rightarrow g(t)$



don't be (ever) satisfied!

Questions

- ☞ Need for limiting g(t)?
- ☞ Need for self-modifying programs (change R over time)?

5 /8
© F. Laux

Reutlingen University

1 Definitions
2 Facets
3 Problems
4 Questions

SW Patterns Classifications

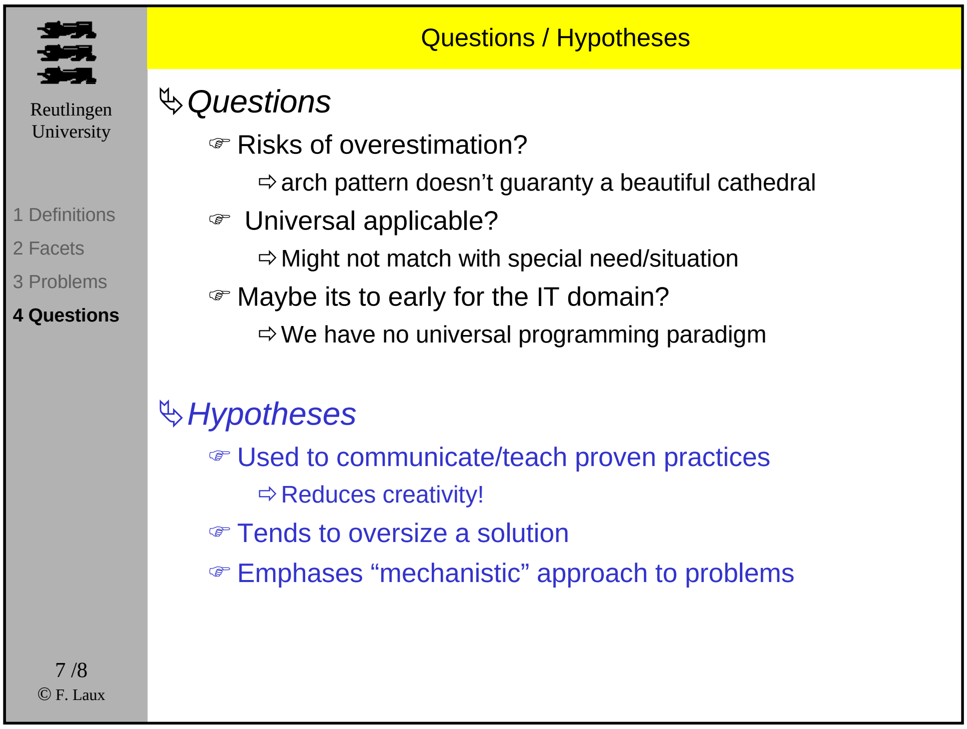
So many (partially) orthogonal classifications

- ☞ Difficult to classify → structure and navigation are interrelated
- ☞ Many interrelations → see GoF patterns
- ☞ How can we solve this problem?

Example

- ☞ Development time
  - ⇨ Requirements capturing patterns → how precise?
  - ⇨ Analysis patterns → which model?
  - ⇨ Design patterns → which model?
  - ⇨ Programming patterns → which paradigm?
  - ⇨ Testing patterns → validation vs. proofing?
- ☞ Run time
  - ⇨ Tuning patterns → autonomic vs. manual?
  - ⇨ Logging patterns → privacy?
  - ⇨ Monitoring pattern → how detailed?

6 /8
© F. Laux



Reutlingen University

1 Definitions
2 Facets
3 Problems
4 Questions

Questions / Hypotheses

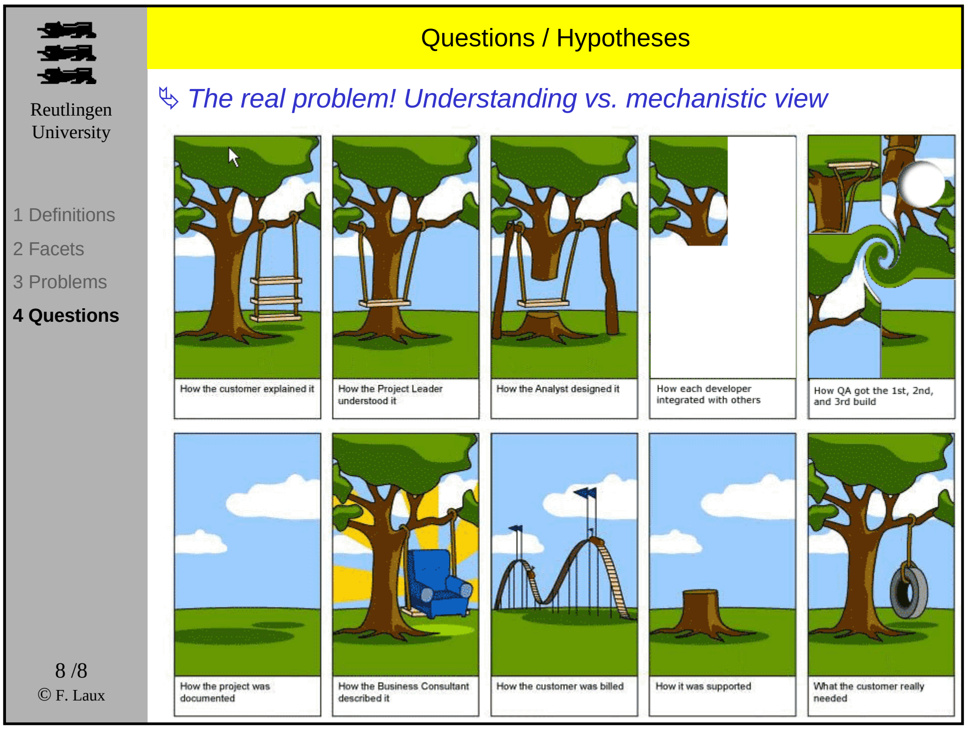
↳ *Questions*

- ☞ Risks of overestimation?
  - ⇨ arch pattern doesn't guaranty a beautiful cathedral
- ☞ Universal applicable?
  - ⇨ Might not match with special need/situation
- ☞ Maybe its to early for the IT domain?
  - ⇨ We have no universal programming paradigm

↳ *Hypotheses*

- ☞ Used to communicate/teach proven practices
  - ⇨ Reduces creativity!
- ☞ Tends to oversize a solution
- ☞ Emphases "mechanistic" approach to problems

7 /8
© F. Laux



Reutlingen University

1 Definitions
2 Facets
3 Problems
4 Questions

Questions / Hypotheses

↳ *The real problem! Understanding vs. mechanistic view*

How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How each developer integrated with others
How QA got the 1st, 2nd, and 3rd build
How the project was documented
How the Business Consultant described it
How the customer was billed
How it was supported
What the customer really needed

8 /8
© F. Laux

Patterns Research

IARIA Position Statement

1 Introduction

In information technology, terms and concepts are often used without a common underlying definition or understanding. In order to have broad and common understanding of the notion *patterns*, we start from the description provided by Wikipedia.

"A design pattern in architecture and computer science is a formal way of documenting a solution to a design problem in a particular field of expertise. The idea was introduced by the architect Christopher Alexander (1977) in the field of architecture and has been adapted for various other disciplines, including computer science, pedagogical, and interaction design." "A pattern must explain why a particular situation causes problems, and why the proposed solution is considered a good one."

Wikipedia also clearly indicates that the main application areas of patterns are currently architecture and and software engineering.

"Pattern in architecture is the idea of capturing architectural design ideas as archetypal and reusable descriptions." "A pattern would not tell the designer how many windows to put in the room; instead, it would propose a set of values to guide the designer toward a decision that is best for their particular application." "In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design. A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations."

Based on these widely accepted beliefs, we will build upon the following assumptions throughout this paper.

- Patterns are heuristics, rules of thumb.
- Documenting patterns is basically knowledge engineering in a particular field of expertise.
- Pattern fundamentals are discipline agnostic.
- The number of disciplines or fields that use patterns is currently quite limited.
- Patterns are often limited to design patterns.

Based on this, we also believe that patterns constitute an interdisciplinary and potentially important research area. In the following sections, we will identify three important areas in patterns research. In a first section, we identify a number of remaining problems and unsolved issues in the area of software engineering design patterns. A second section discusses the use of patterns as a means to achieve reactive system behaviour in information systems. Finally, we propose some additional research problems where pattern research could make a significant contribution.

2 Issues in Software Engineering

The use of patterns in software engineering is widespread, and in general considered to be successful. However, several serious issues remain, and need to be further investigated. We discuss some genreal issues first, and zoom in on the issue of suitability.

2.1 Current Limitations of Patterns

A first problem is the classification of patterns. Pattern classification is often only partially orthogonal. Therefore related patterns may occur in different pattern classes. For instance, structural patterns relate strongly to navigational patterns. The same design patterns apply in different phases of software development.

2.1.1 Questions

- How can we ensure great software by using patterns? Knowing the arch pattern does not necessarily lead to a beautiful cathedral.
- Are patterns universally applicable? A pattern might not match with special needs or in a certain situation, e.g. statement of Michal Zemlicka: *A pattern can be an anti-pattern in a different situation and/or for a different user.*
- Is it still to early to stick to patterns? We have no universal programming paradigm. Applying a software pattern is like using the differential pattern for a car without having transmission mechanics in place.

2.1.2 Hypotheses

- Patterns are used to communicate proven practices. But, as consequence they reduce the creativity when applying them.
- Patterns tend to oversize a solution, i.e. they do not fit into agile SW-development.
- Patterns emphasize a mechanistic approach to a problem. This blocks creativeness, hides the clear view to the main problems. For instance, the

main problem is still to formalize the true requirements of a client, not how to solve a requirement technically.

- Patterns are a substitute solution, e.g. statement of Hans-Werner Sehring: *Patterns are only necessary because programming language does not have an appropriate language construct for it.*

2.2 On the Suitability of Patterns

Patterns are often success stories, but the scope of their applicability must be precisely specified (and usually investigated). A pattern used outside its scope may be useless, or even worse, it can even play the role of an antipattern.

It is especially important for recommended development procedures. They are often tested under some specific conditions, but recommended as generally applicable. But appying the procedure out of scope can negatively influence the project. An example is the use of SOA as understood by OASIS and the big software vendors. It is designed to work, if applied within big enterprises playing the role of market leaders. It is therefore possible to expect that the environment can be quite stable (such clients have usually well-polished business processes often coded into manuals) and that there is enough resources (money, computational power, IT staff, etc.). Applying such methodologies for SME being in quickly changing environment, having very limited resources (IT staff inclusive) may lead to project failures. It does not mean that the use of SOA is the reason. There are SOA types well suited for SME and their requirements. In this case the problem is in the use of techniques (patterns) developed for different environment and simply applied here.

The same holds for antipatterns. For example, integration of legacy systems and third-party applications is not recommended for object-oriented systems or for SOA systems in the sense of OASIS. For control systems and software confederations (one of SOA types) it is a standard and recommended turn.

Patterns are good practices, but they may require experienced people to be successfully applied. In order to take real advantage from patterns, it is necessary to have proper knowledge and skills and to apply patterns within scope where they have been really successfully applied. Breaking that rule may cause project failure.

3 Use in Reactive System Behaviour

In this section, we explore some possibilities and issues in the use of patterns to create reactive behaviour in systems. We first briefly discuss the many facets of systems, of patterns, and of their interaction. Based on this, we present some research questions and issues.

3.1 Systems and Patterns

3.1.1 The Many Facets of Systems

Before looking at the various facets of patterns in systems, a brief look at the different facets of systems is outlined first. I will look at four different properties of systems; needless to say, there are many others.

- *Information/Content Provided by a Systems*: The content that the system provides or the information that flows through the system can be tightly controlled, loosely controlled or little or no control (Internet).

- *Structure of the Systems*: The structure with respect to the underlying infrastructure, the application running on top of this infrastructure, the service provided by the application, and the users of the system can be either Static, Dynamic, or a mixture.

- *Type of system*: A system can be a mixture and/or hybrid of the following: Large (operates in the wide-area setting), Small (operates in the local-area setting), Private (limited access system), and Public (open access system).

- *Purpose of the system*: A system may serve various purposes and this will influence its operation and the way that it is operated/controlled. Systems can be used to control and monitor critical infrastructures, life support systems, business interactions, or just serve for entertainment purposes.

3.1.2 The Many Ways to Use Patterns

There are many ways that the term pattern can be used when discussing IT systems. The term usually refers to design patterns or various patterns used in the architecture, design, and construction/development of a system or software product/service. Another use for the term is encountered once the system is operational when we look for various patterns on how the system is behaving, flow of information (both at the application and IT infrastructure layer), usages of the system, etc. As far as the second usage of the pattern is concerned, at least the following questions must be answered for each system.

1. When do we specify such patterns?
    - During the implementation time, i.e. designed into the system.
    - During the tuning phase of the system.
    - Dynamically while the system is operational.
2. Who can specify the patterns?
    - The system architects and/or designers.
    - The system developers.
    - The system administrators.

3. Will any large system develop its own personality, i.e. it will be unique in the way it behaves? In this case, the only ones that know how it behaves and how it reacts to tweaking are the system administrators. For such systems, could any best-practice be defined (i.e. what patterns to look for and how to react) or is each system unique? The specific patterns (with respect to input and parameter setting) to look for may have to be tailored to each specific system.

3.2 Issues and Questions

3.2.1 Functional Questions

In summary, some questions could be investigated to test whether or not patterns for behavior of large systems are appropriate in-order from them to operate more reliably or to improve their survivability.

- Who is Capturing? Who should capture them and define what to capture: system users, system administrators, intelligent monitoring systems?

- What are we capturing? Some categories may be defined like:

  - Behavior of the IT infrastructure, application running on top of the system, or a hybrid (maybe they are coupled)
  - Security and privacy issues of what is captured. Might not want to publish or make available information that the system is vulnerable.

- What are we preventing? System crash, system overload, loose of control, leak of sensitive information.

- How are we reacting: Some categories may be defined like:

  - Do we want to make a survivable system w.r.t. intentional and accidental failures.
  - Can we trust automatic reaction?

Some commonality can probably be found for systems that belong to the same category. In this case commonality can be abstracted and patterns defined for groups of systems like moniroting and control systems for critical infrastructures.

3.2.2 Technical Questions

Capturing, preventing, and reacting facets of the system behaviour is quite complicated from a technical perspective, and we need to have a precise definition of the relevant terms. Reactive systems need to provide reactions to events. Therefore events need to be captured. This is only possible if the software takes notice of an event before the event disappears. Buffers of monoflops (mono-stable trigger circuit) can assure that event will not be missed. What buffer size is necessary? For how long has a monoflop to signal an event? Which events are worthwhile to capture?

To prevent a certain state it is necessary to analyze the next state before the transition takes place. Does a sandbox pattern truly reflect reality? What happens when a transition is blocked? Does this lead to a dead-lock? Is a rollback possible? Do we have a redo pattern? An evolvable system is like an adaptive system with a moving target. Are there limiting patterns for the goal function? How about self-modifying programs (reactive function R changes over time, not only due to history/state)? I see a strong need for patterns governing the limits of evolving programs (e.g. avoid dangerous behavior of a program)

4 Other Areas for Pattern Research

In this section we discuss two other areas that we consider to be eligible for pattern-related research: enterprise engineering and evolvable modularity.

4.1 Enterprise Engineering

Today, software engineering and information systems are of crucial importance for the automation of business processes in organizations. Many people believe that business processes and organizations should also be subject to systematic and accurate design and engineering. Enterprise engineering, related to systems engineering and software engineering, is the *discipline concerning the design and the engineering of enterprises, regarding both their business and organization.*

It seems logical that the design and engineering of enterprises could also make use of design patterns at the business or organizational level. Moreover, these business or organization patterns could be linked—or even automatically mapped—to the design patterns of the software systems that are used to implement or automate them.

4.2 Evolvable Modularity

Whether patterns are used in software engineering or in other application areas, we believe that it is important that they accomodate—or even enable—change in our fast changing environment. Patterns should not only be targeted at the fulfillment of static requirements, but also at the incorporation of a set of anticipated changes. This is quite similar to the idea of *design for change*, as already formulated by Dave Parnas in 1972. We mention here the following research activities.

- Introducing a substantiated temporal dimension in design patterns: extending static design patterns to 'evolvable design patterns, i.e. patterns that are designed to allow for evolving systems along certain degrees of freedom: - in software where evolvability is a crucial and problematic issue, but also in many other areas - in large-scale designs like microchips or airplanes where the evolvability is also very limited even on the drawing table

- Studying the combinatorial or cascade effects that are caused by applying changes to design patterns, as these cascade effects are clearly a major inhibitor towards evolvability of systems and/or their designs