Software Requirements

L9: Requirements Patterns

Prof. W. Maalej – @maalejw

Overview

1 **Motivation**

2 Requirements Templates

3 Requirements Patterns Catalogues

What is a Pattern?

"Each pattern describes a **problem** which occurs over and over **again** in our environment, and then describes the **core of the solution** to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"

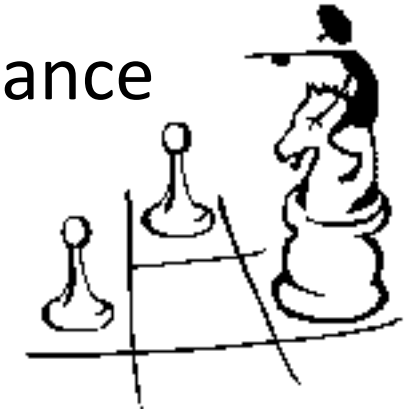
Christopher Alexander,
The Timeless Way of Building 1977
Job: Architect (of buildings)

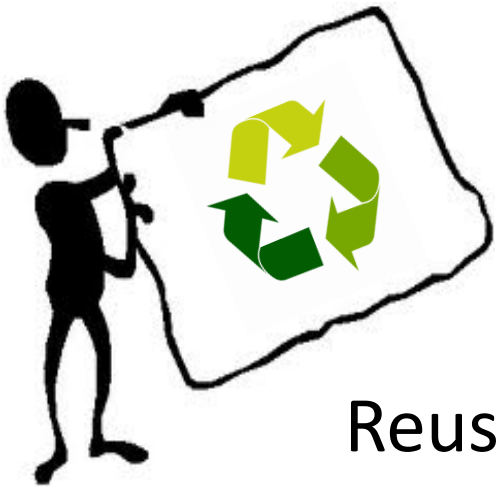
Patterns in software engineering

- Design patterns
  - A set of **good solutions** to a commonly occurring problem in software design
- Design anti-patterns
  - A set of **bad solution** to commonly occurring problems in software design
- Testing patterns/anti-patterns
- Deployment patterns/anti-patterns
- Management patterns/anti-patterns
- Requirements patterns/anti-patterns

Benefits of patterns

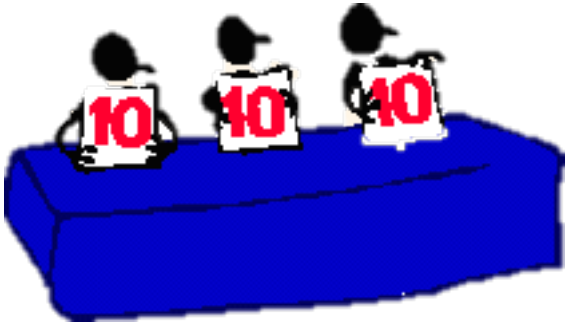
Guidance





Reuse

Consistency



Performance/

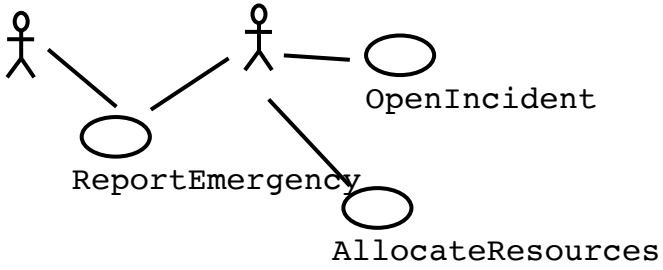
Productivity

Requirements pattern

- A requirements pattern is a **template** which includes types of information for each common **type of requirements** a project might encounter
  - Can be tailored to specific domains
  - For the purpose of reuse
- **Requirement reuse:** Reuse of knowledge acquired in the requirements engineering phase

Requirements reuse strategies

- **Specific pattern-based reuse**
  - Artifact-oriented patterns
  - Domain-oriented patterns



- **Refinement-oriented reuse**
  - Define how the goals can be achieved
  - Adopt a goal oriented modeling language as i* or KAOS

- **Template-oriented reuse**
  - Provide templates for requirements, written in a natural language
  - Usually constrained by syntactical rules

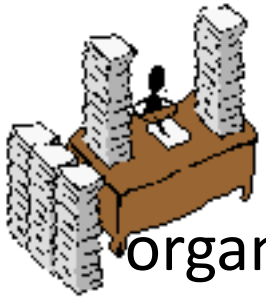
When to use pattern?

- A company is involved in **multiple** software **projects**
- Projects are conducted within **similar contexts**
  - Same/similar customers
  - Same/similar domains
- Existence of **different business units** in the same company
- Compliance to strict **standards**

The impact on requirements engineering

Elicitation

Support **discovering** of stakeholder's needs



Documentation

Make it uniform, organized, comprehensive

Negotiation

Consequences known right away

Analysis

Explicit relationships

Validations

Systematic

Management

Easier traceability

Overview

1 Motivation

2 **Requirements Templates**

3 Requirements Patterns Catalogues

Rupp's requirements templates

1. **Autonomous requirements**

   System functions that are independent of user-interactions

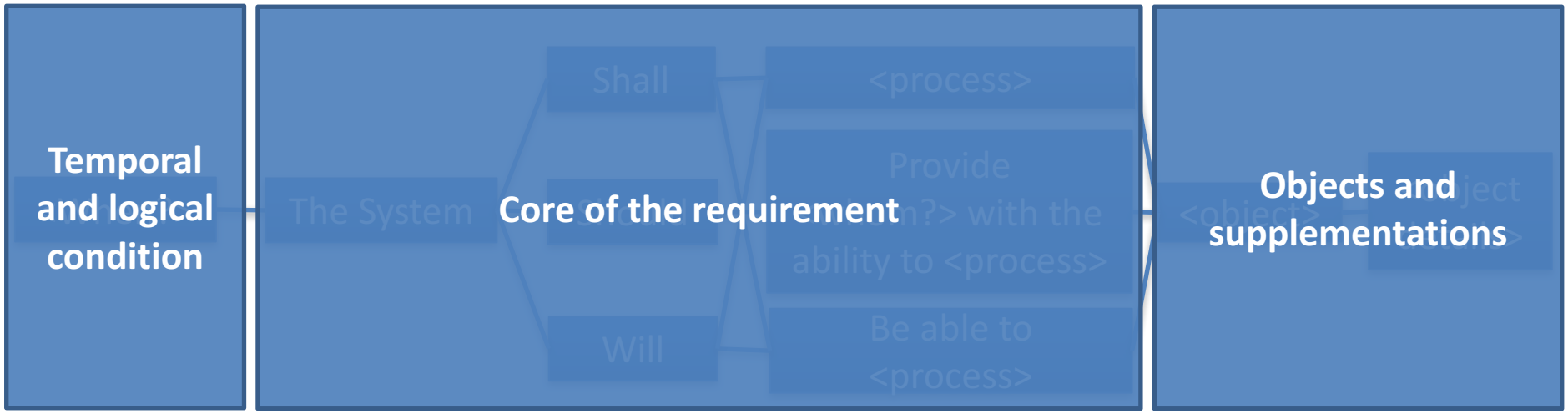
2. **User Interaction Requirements**

   Requirements defining reactions on user actions

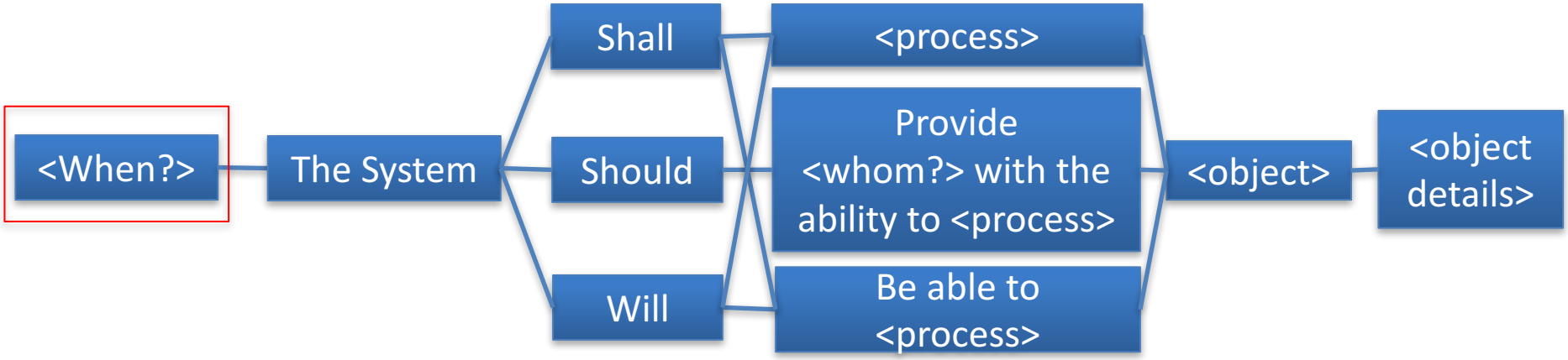
3. **Interface Requirements**

   System functions that depend on trigger events from other systems

Rupp's requirements template structure

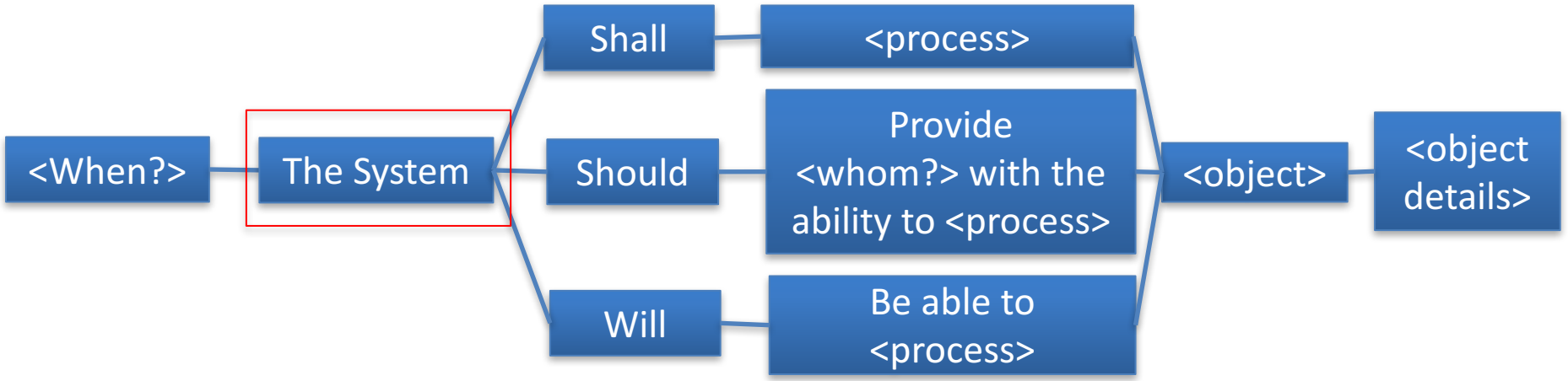


Rupp's requirements template



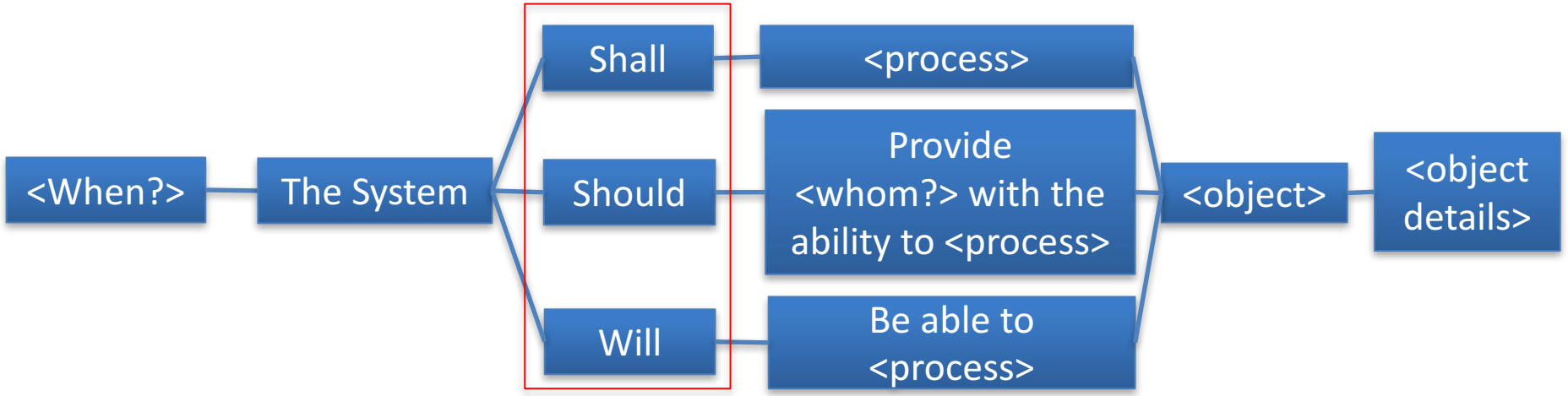
- Define under which condition the functionality in the requirements needs to be performed

Rupp's requirements template



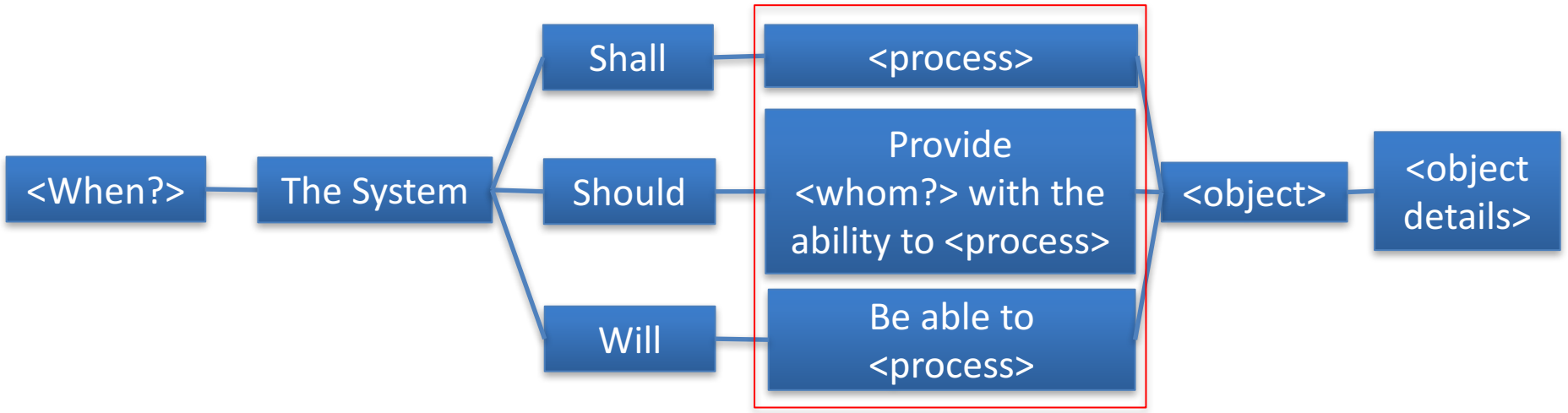
- The system that provides the functionality

Rupp's requirements template



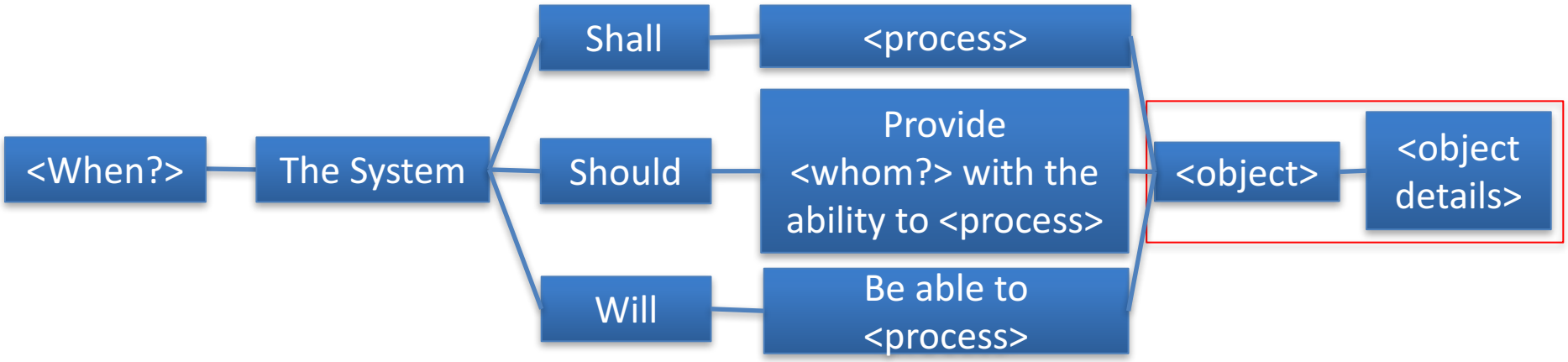
- Indicates the importance of the requirement
- Three options available:
  - legally binding > use keyword "shall"
  - strongly recommended > use keyword "should"
  - used in future > use keyword "will"

Rupp's requirements template



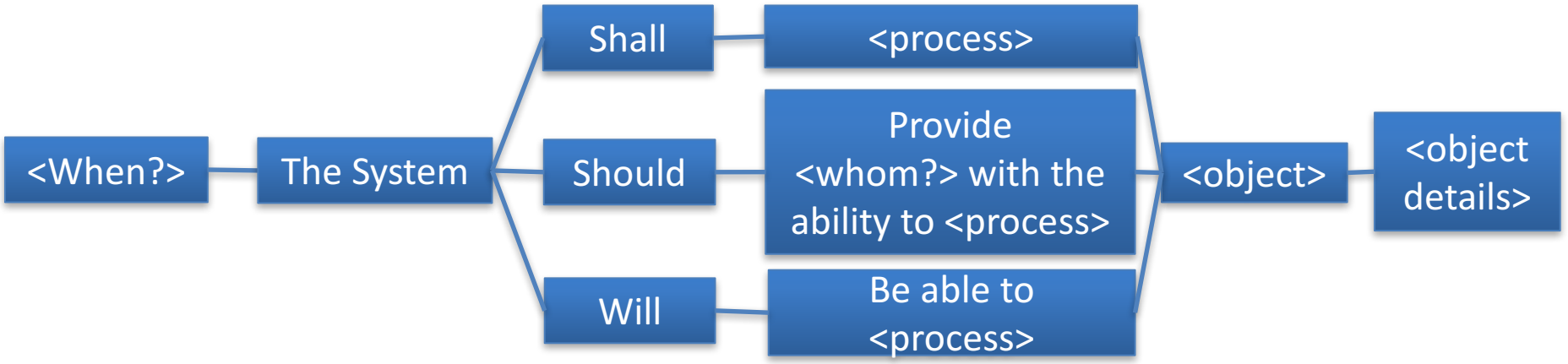
- The required functionality
- Three pattern options:
  - **<process>** : applies to user-independent requirements
  - **provide <whom?> with the ability to <process>**: applies to functionality that the system provides to specific users
  - **Be able to <process>**: applies to requirements that the system performs as reactions to trigger events

Rupp's requirements template



- The object and its properties that require the functionality

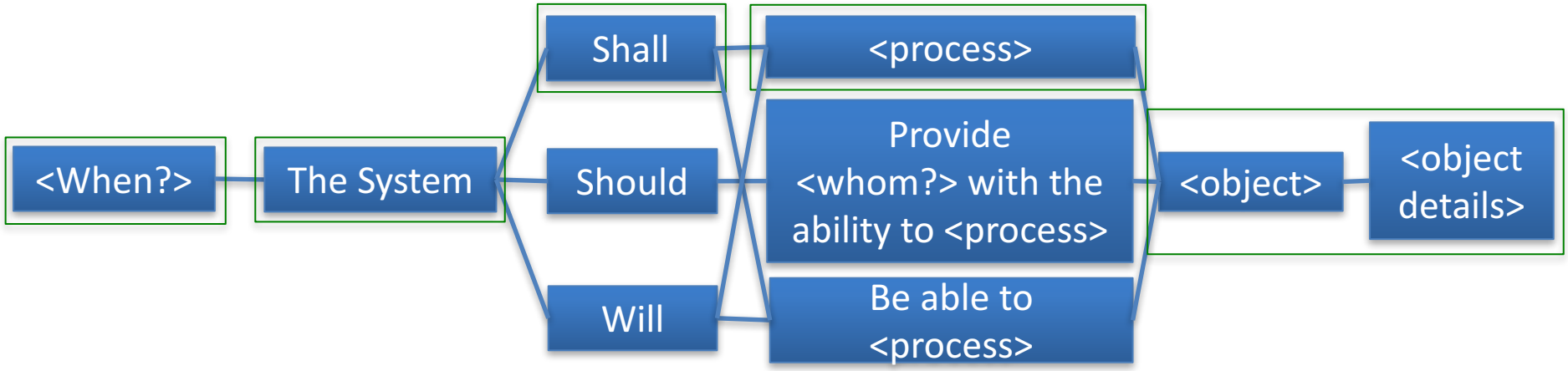
Rupp's requirements template example



Example:

If the power-button is pressed for longer then 3 seconds, the system shall open up the task manager.

Rupp's requirements template example



If the power-button is pressed for longer then 3 seconds, the operating system shall open up the system settings dialog.

EARS template structure

- EARS: **E**asy **A**pproach **R**equirements **S**yntax [Mavin et al RE09]
  - A structured aid of expressing requirements
  - Generic syntax for **functional** requirements

    [Trigger][Precondition] Actor Action [Object]

Example

Trigger:	[When the Add button is pressed]
Precondition:	[and at least one item is selected],
Actor:	the system
Action:	shall add the selected items to
Object:	the shopping cart.

EARS template types

- Differentiates between five different requirement template types
    - Ubiquitous
    - Event-driven
    - Unwanted behaviors
    - State-driven
    - Optional features

(Non-) ubiquitous requirements

Ubiquitous Requirements

- Are universally valid
- Do not require a trigger
- Define a fundamental system property, rather then a software function

Non-Ubiquitous Requirements

- Require an event or trigger
- Denote a state of the system, or an optional feature
- Most requirements are of this type

EARS templates

Template Name	Template
Ubiquitous	The <system name> shall <system response>
Event-Driven	**WHEN** <trigger><optional precondition> the <system name> shall <system response>
Unwanted Behavior	**IF** <unwanted condition or event>, THEN the <system name> shall <system response>
State-Driven	**WHILE** <system state>, the <system name> shall <system response>
Optional Feature	**WHERE** <feature is included>, the <system name> shall <system response>
Complex	(combinations of the above)

Event-driven requirements template

WHEN <trigger><optional precondition> the <system name> shall <system response>

- Initiated by a trigger
- Use the “When” keyword

Examples

- **When** the home button of the Smartphone is pressed, the Android OS shall turn on the screen.

Unwanted behavior requirements template

IF <unwanted condition or event>, THEN the <system name> shall <system response>

- Handle unwanted Behavior
  - Errors, failures, faults
- Use of “if” and “then” keywords

Examples

- **If** the battery charge drops below 10%, **then** the OS shall display a warning message.

State-driven requirements template

WHILE <system state>, the <system name> shall <system response>

- Trigger while the system is in a specific state
- Use the “While” or “During” keyword

Example

- **While** in Low Power Mode, the OS shall keep the display brightness at the minimum level.

Optional feature template

WHERE <feature is included>, the <system name> shall <system response>

- Invoked only in specific system parts, having particular feature
- Use the “Where” keyword

Example:

- **Where** a physical Back-button is present, the OS shall hide the appropriate software Back-button.

Complex requirements template

Complex Requirements:

- Assess conditional events involving multiple triggers, states, or optional features
- Combine keywords “When”, “If” and “then”, “While”, and “Where”

Examples:

- **While** on battery power, **If** the battery charge drops below 10% remaining, **then** the OS shall display a warning message.

Improving existing requirements using EARS

Original	Rewritten
The software shall activate the loudspeaker.	**While** the loudspeaker button is pressed, the system shall activate the loudspeaker. (State-Driven Requirement)
The system shall decrease brightness to save battery.	**While** the battery charge is below 10% remaining, the system shall keep the display brightness at the minimum level. (State-Driven Requirement)

Overview

1 Motivation

2 Requirements Templates

3 **Requirements Patterns Catalogues**

Requirements patterns catalogue

- Rationale
  - Collect knowledge about a particular type of requirement in a convenient way for requirements stakeholders and requirements processes

Structure of a requirements pattern

- Sections:
  - Guidance
  - Goal
  - Content
  - Template
  - Examples
  - Extra Requirements
  - Considerations for development and testing
  - Glossary

- Not one-size-fits-all

[Withall 2007]

Building a requirements patterns catalog

- Iterative elicitation of Requirements Patterns
  - In the beginning systematic
  - In the future opportunistic

- Refinement and extension driven by Case Studies

Requirements pattern: example 1

Name	Interface Load Time	
Goal	Load the system user interfaces quickly, Offer a good response time to the user	
Form 1	Text	The system shall response to user interface actions in %amountOfTime% %timeUnit% at most
	Parameters	**amountOfTime**: is a number that indicates time duration **timeUnit**: is the unit of time corresponding to the units for responding an user interface action
	…	
Form 2	The system shall response to %intActions% user interface actions in %amountOfTime% %timeUnit% at most	
	…	

Pattern forms (also called template)

- Captures a basic essence of a requirement
- Usually, mutually exclusive in a project
- A basic form usually contains
  - **The text**
  - The description
  - **The comment**
  - The author
  - The sources and keywords
  - …

Requirements pattern: example 2

Name	Data Recovery	
Goal	Ensure that the system recovers after crashes.	
Form 1	**Text**	The system shall not lose data in case of crash.
	Comment	Focuses on the “what” (data)
Form 2	**Text**	The system shall follow some recovery procedure in case of crash.
	Comment	Focused on the “how” (procedure)

Requirements patterns extensions

- Enrich a pattern by extending its basic form
- Provides a more detailed explanation of important concept

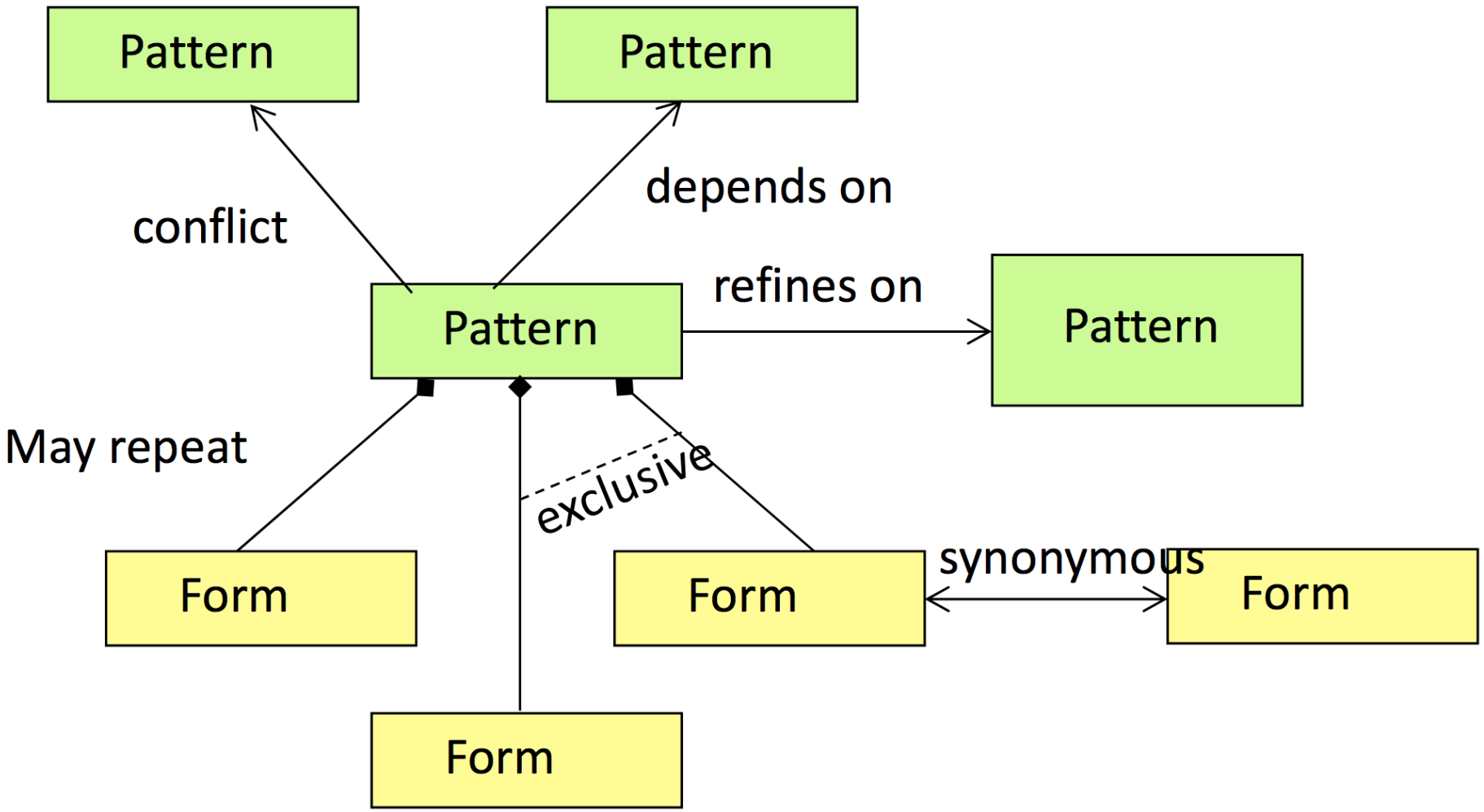
Name	Failure Alerts pattern	
…		
Form 1	**Text**	The system shall give an alert in case of failure
	…	
Form 1 Extension	**Alert Types**	Lists the types of alerts provided by the system.
	Failure Types	Lists the types of failures provided by the system.

Parameters

- Requirements pattern can contain parameters
  - e.g. Integer, Character, String, Enumeration,…
- Parameters may fulfill correctness conditions

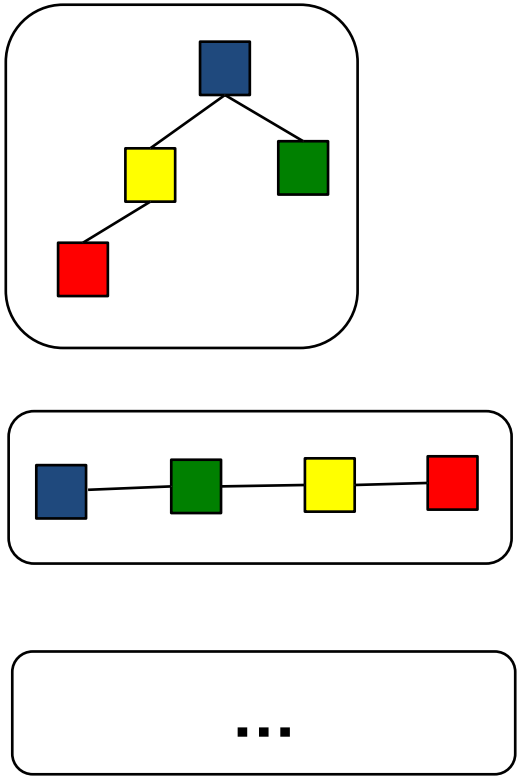
Name	Failure Alerts pattern	
…		
Form 1	**Text**	The system shall give an alert in case of failure.
	…	
Form 1 Extension	**Alert Types**	Alerts provided by the system shall be AlertTypes.
	Failure Types	Failures provided by the system shall be FailureTypes.

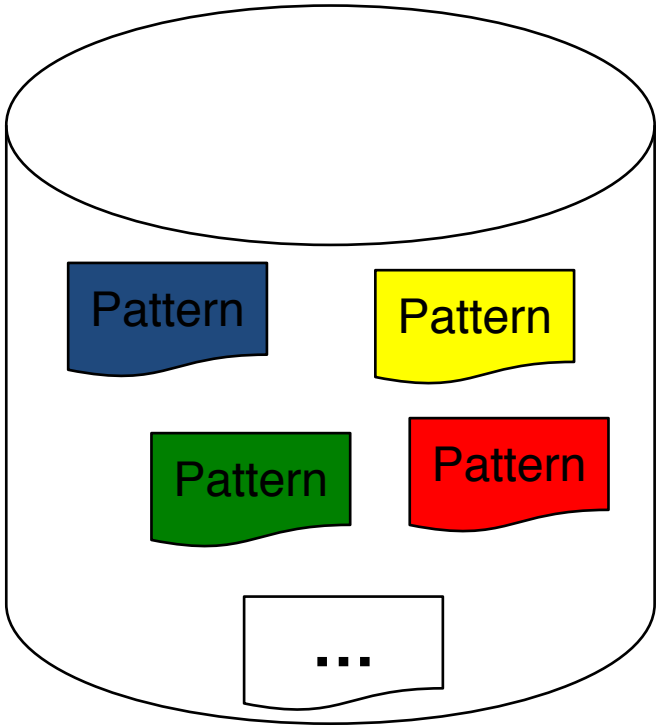
Patterns catalogue structure



Requirements patterns catalogues

- A catalog is a collection of patterns
- Different kind of classification possible





Classification based on quality models

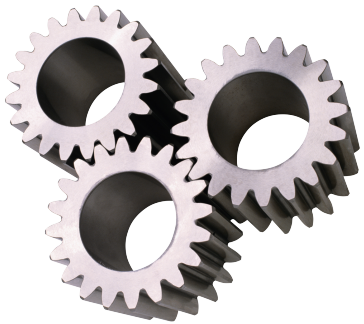
- Quality characteristics defined in quality models can be used as “classifier” for the catalogue
- Quality characteristics are …

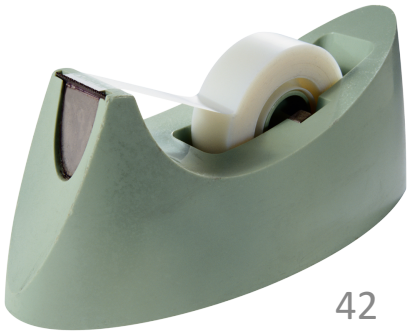
“The set of characteristics and the relationships between them which provide the basis for specifying quality requirements and evaluating quality”

ISO/IEC 14598-1,
Software product evaluation 1999.

Quality characteristics from the ISO/IEC 25010 Standard

- **Functionality**
  - Suitability
  - Accuracy
  - Interoperability
  - Security
  - Functionality Compliance
- **Usability**
  - Understandability
  - Learnability
  - Operability
  - Attractiveness
  - Usability Compliance
- **Maintainability**
  - Analyzability
  - Changeability
  - Stability
  - Testability
  - Maintainability Compliance





Quality characteristics from the ISO/IEC 25010 Standard (ctd.)

- **Reliability**
  - Maturity
  - Fault Tolerance
  - Recoverability
  - Reliability Compliance

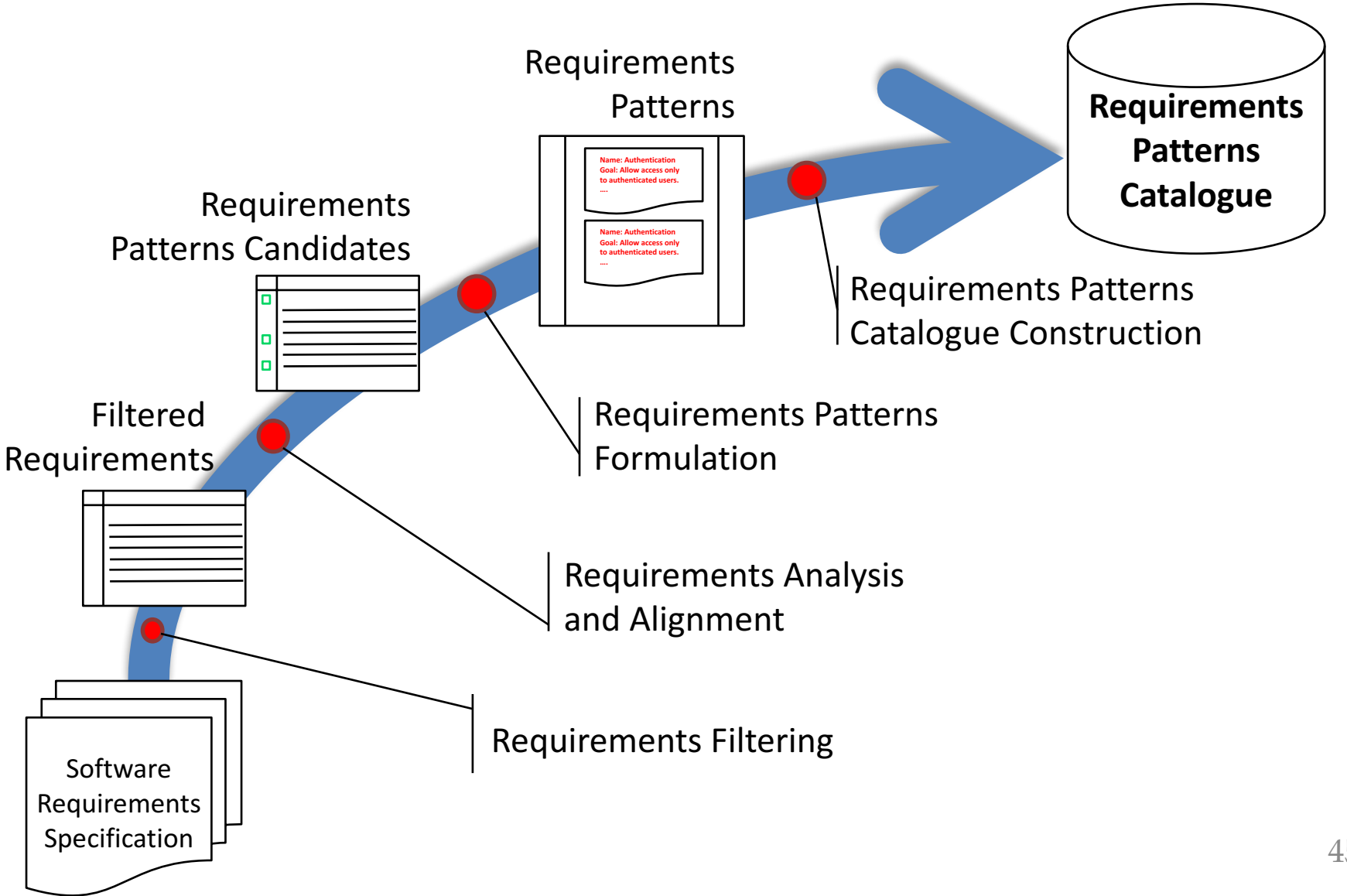
- **Efficiency**
  - Time Behaviour
  - Resource Utilization
  - Efficiency Compliance

- **Portability**
  - Adaptability
  - Installability
  - Co-Existence
  - Replaceability
  - Portability Compliance

Functional vs. Non-functional requirements

- Functional requirements are highly domain dependent
  - The pattern should capture the general-purpose form, so they can be refined
- Non-functional and non-technical requirements
  - **High reusability across different domains**

Requirements patterns catalogue construction process



Requirement analysis & alignment: example 1

The system shall respond to user interface actions in 10 seconds at most

→ Quality classifier: efficiency

→ 2 parameters

The system shall have unavailability periods shorter than 60 minutes per day

→ Quality classifier: reliability

→ 3 parameters

Requirement analysis & alignment example (2)

The system must be available 10 hours per day and 5 days per week.

→Quality classifier: reliability

→6 parameters

Requirements pattern candidates search

- Can be challenging:
  - Some requirements are too specific, and some are too general

- The effort of constructing and managing a Requirements Pattern can outweigh the benefits of its usage
  - Find a suitable tradeoff

Requirements pattern development guide I

1. Does the expected benefit outweigh the effort needed?
   - Expectation: How often will it be used in the future?
   - Value: What value does it deliver when applying it?
   - Cost: How much time is needed to create and maintain it?
2. Construct a skeleton for the pattern
3. Derive some usage examples

Requirements pattern development guide II

4. Analyze the examples
   - Identify commonalities
   - Identify potentials for improvements
   - Refine them to fit the associated Requirements Pattern
5. Identify the parameters
   - Quality aspect (e.g. performance, flexibility)
   - Identify parameter types and its constraints
     - E.g. integer range from 1 to 60

Requirements pattern development guide III

6. Construct the templates
   - Identify the possible alternatives and variants:
     - Synonyms, Extensions,…
   - Identify the possible properties:
     - Multiplicity,…
7. Validate:
   - Apply it, Peer review,…
8. Finalize
9. Update Catalogue
   - Update associations (conflicts, synergies,…)

Case study mining

1. Collect Case Studies from real-world projects
2. Analyze and extract a set of observations
3. Identify patterns
    1. Find identical observations from different projects
    2. Mark them as pattern candidates
4. Elaborate the pattern candidates
    1. Derive pattern descriptions
    2. Publish them in a central pattern repository

Summary

1. Requirements patterns guide the elicitation process, lead to higher quality requirements and easier validation

2. Requirements templates help in creating high quality requirements in a cost and time efficient way

3. Patterns catalogues organize requirements patterns based on a scheme, e.g., the quality characteristics