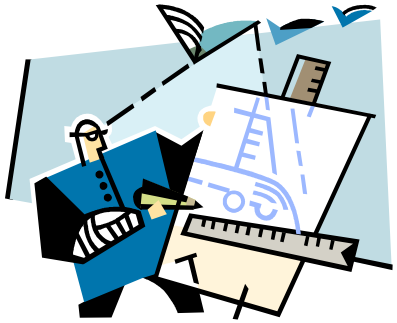
SECAPPDEV 2008
Security Architectures

Riccardo Scandariato

Wouter Joosen

KATHOLIEKE UNIVERSITEIT
LEUVEN
DistriNet

Architecture



Users

Architect

Developers

Creates

Creates

Creates

Prescribes

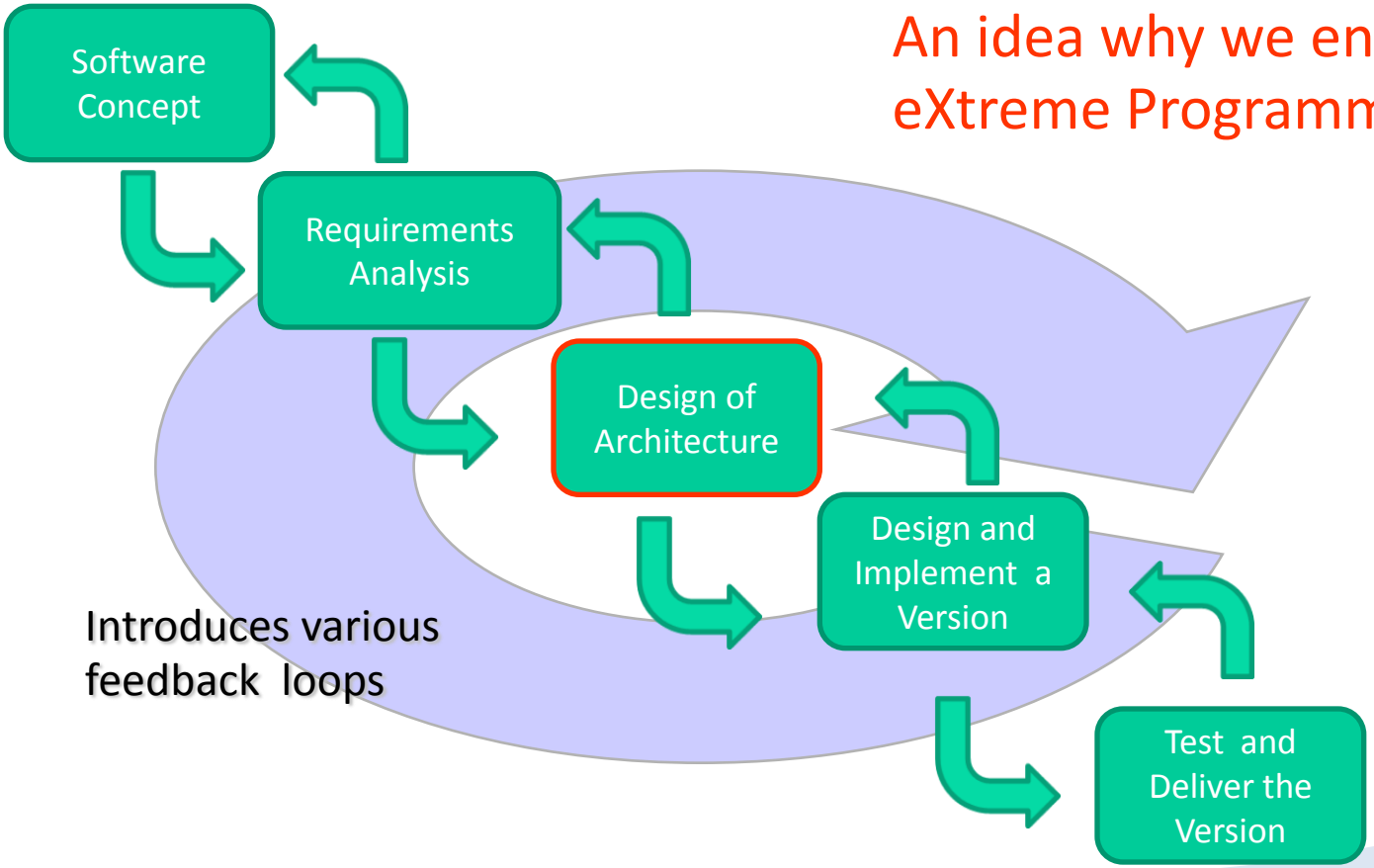
Prescribes

Requirements

Software Architecture

Software Product

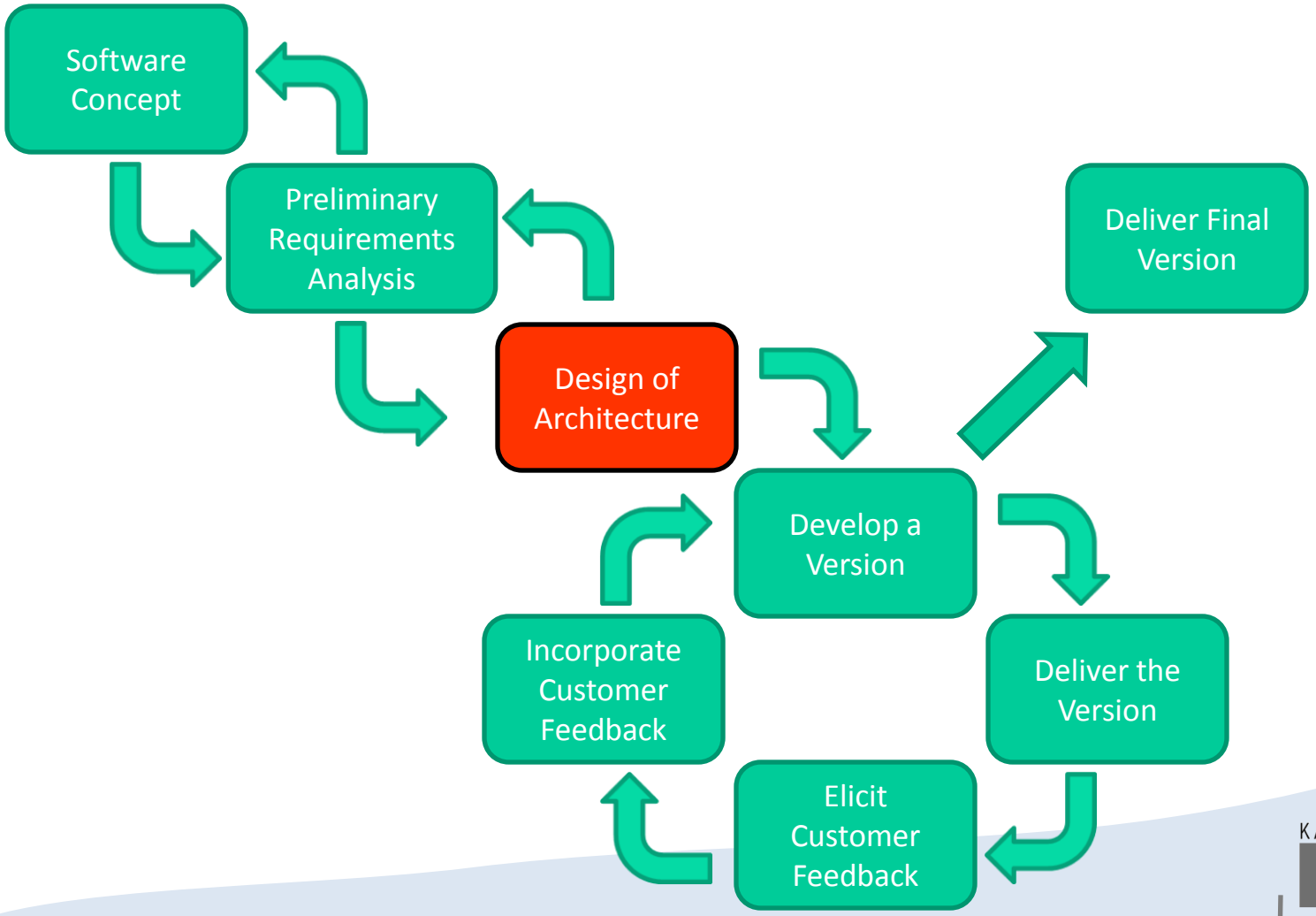
Iterative Software Development



An idea why we ended up with eXtreme Programming?

Software Architecture

Sign off



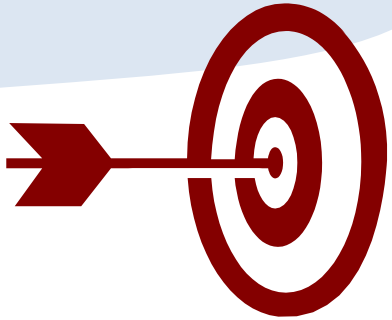
The play

- Act I – Prologue
  - Introduction to Software Architectures
- Act II – Security on stage
  - Security Architectures with Patterns
- Final rehearsal
  - A case study

Act I
Software Architectures

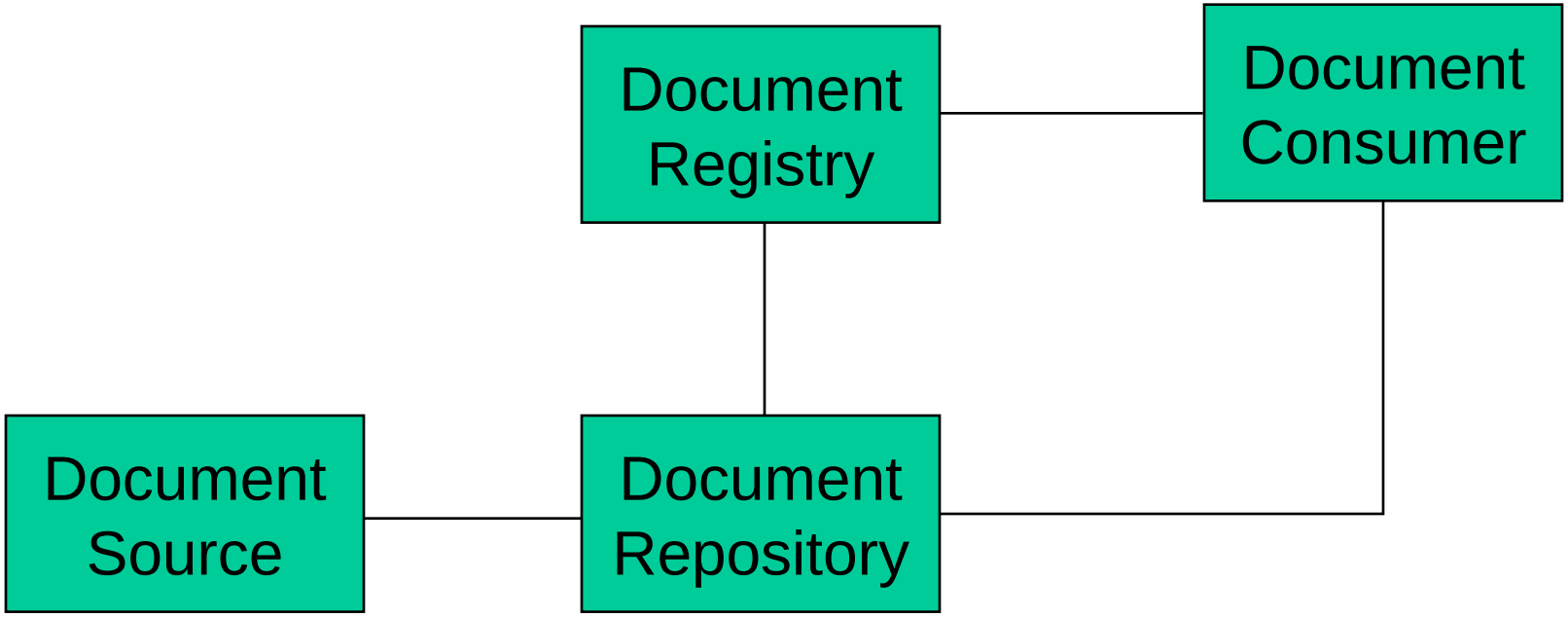
KATHOLIEKE UNIVERSITEIT
LEUVEN
DistriNet

Objectives



- What is Software Architecture?
- Why is Software Architecture important?
- How to Create Software Architecture?
- How to Evaluate a Software Architecture?

Is this an architecture?



Boxes and arrows

Definition of Software Architect

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them

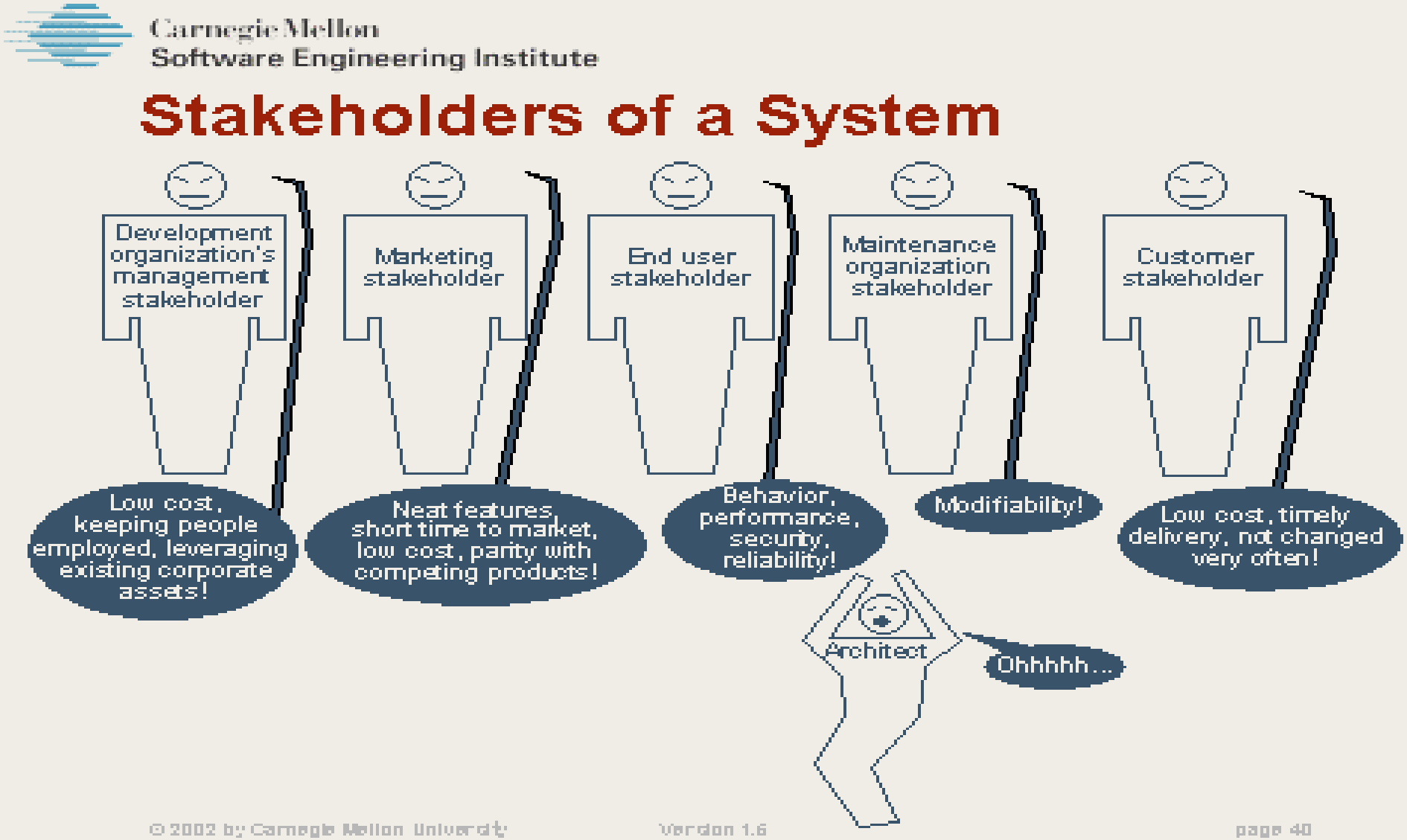
Other Definitions

"Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution"[IEEE 1471]

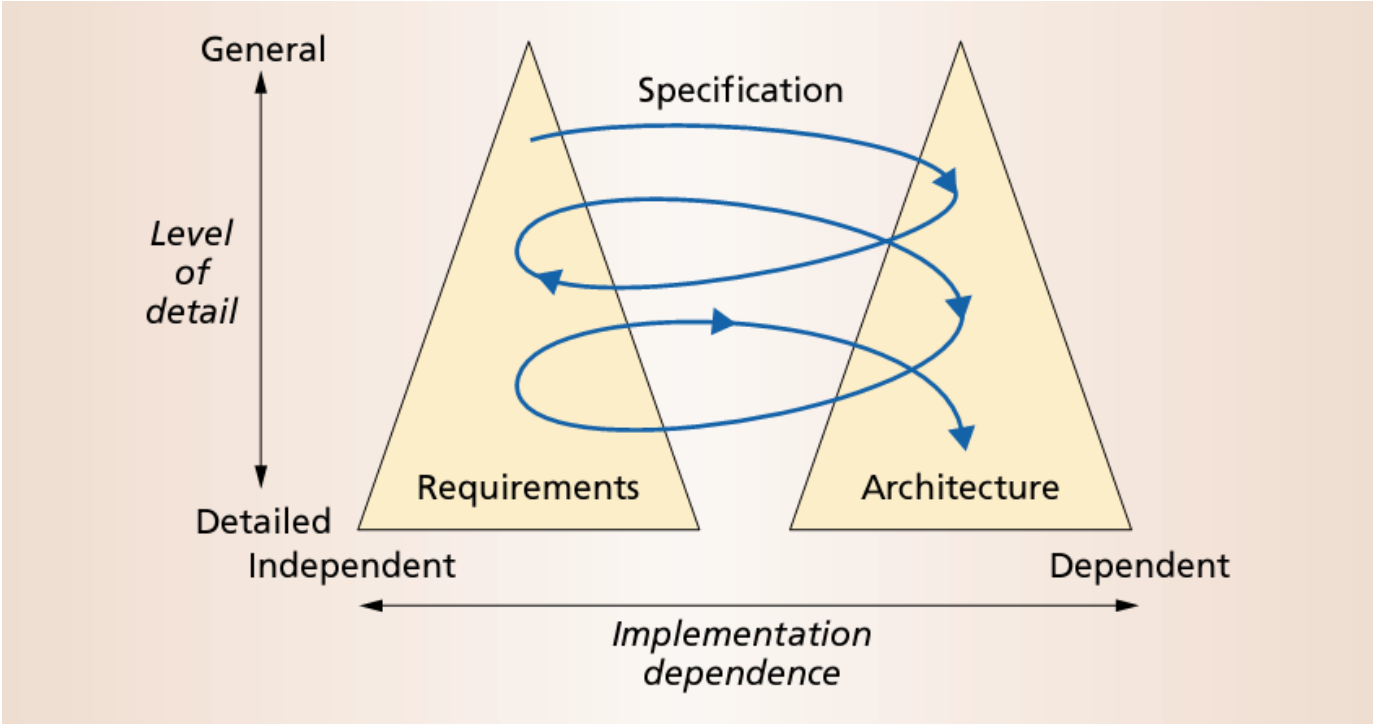
Maier, M. W., Emery, D., and Hilliard, R. 2004. ANSI/IEEE 1471 and systems engineering. *Syst. Eng.* 7, 3 (Sep. 2004), 257-270

Importance of architecture

Reconcile stakeholders



Importance of architecture
Impact on requirements



Twin Peaks

Creating software architectures

- Architectures are largely influenced by software qualities (non functional requirements)

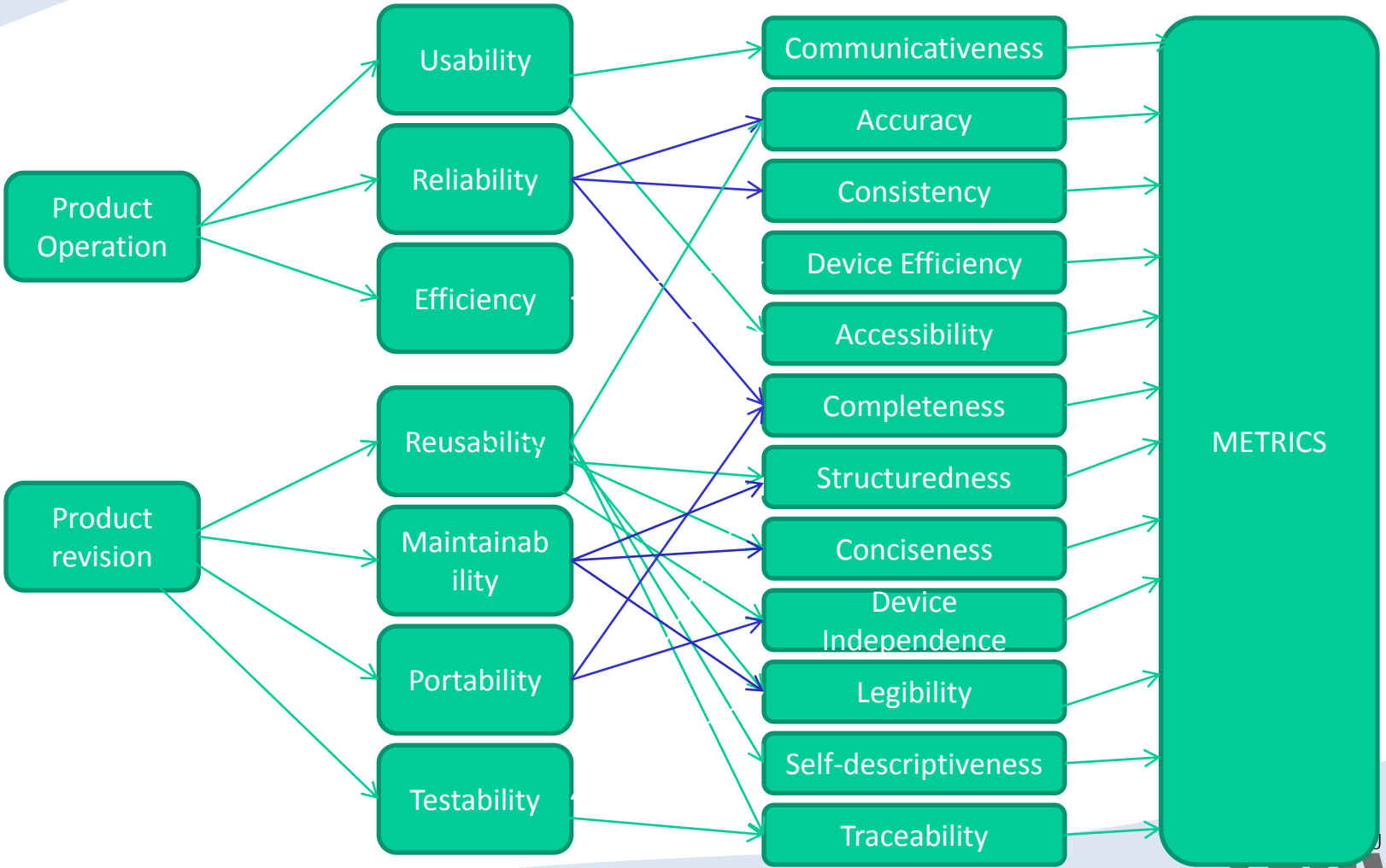
- Software qualities
  - Performance
  - Modifiability
  - Availability
  - Security

Creating SA
Quality Models

- How achieve software quality?
  - Understand what quality means: quality model
  - Verify that quality is achieved: measure

- Quality Model
  - ISO9126, Boehm, etc

Creating SA Quality Model



Creating SA
Attribute-driven design

- A recursive decomposition process where, at each stage, tactics and architectural patterns are chosen to satisfy a set of quality scenarios and then functionality is allocated to instantiate the module types provided by the pattern.

Creating SA
Quality attribute scenario

Source:
Developer

Stimulus:
Wishes to change the UI

Artifact:
Code

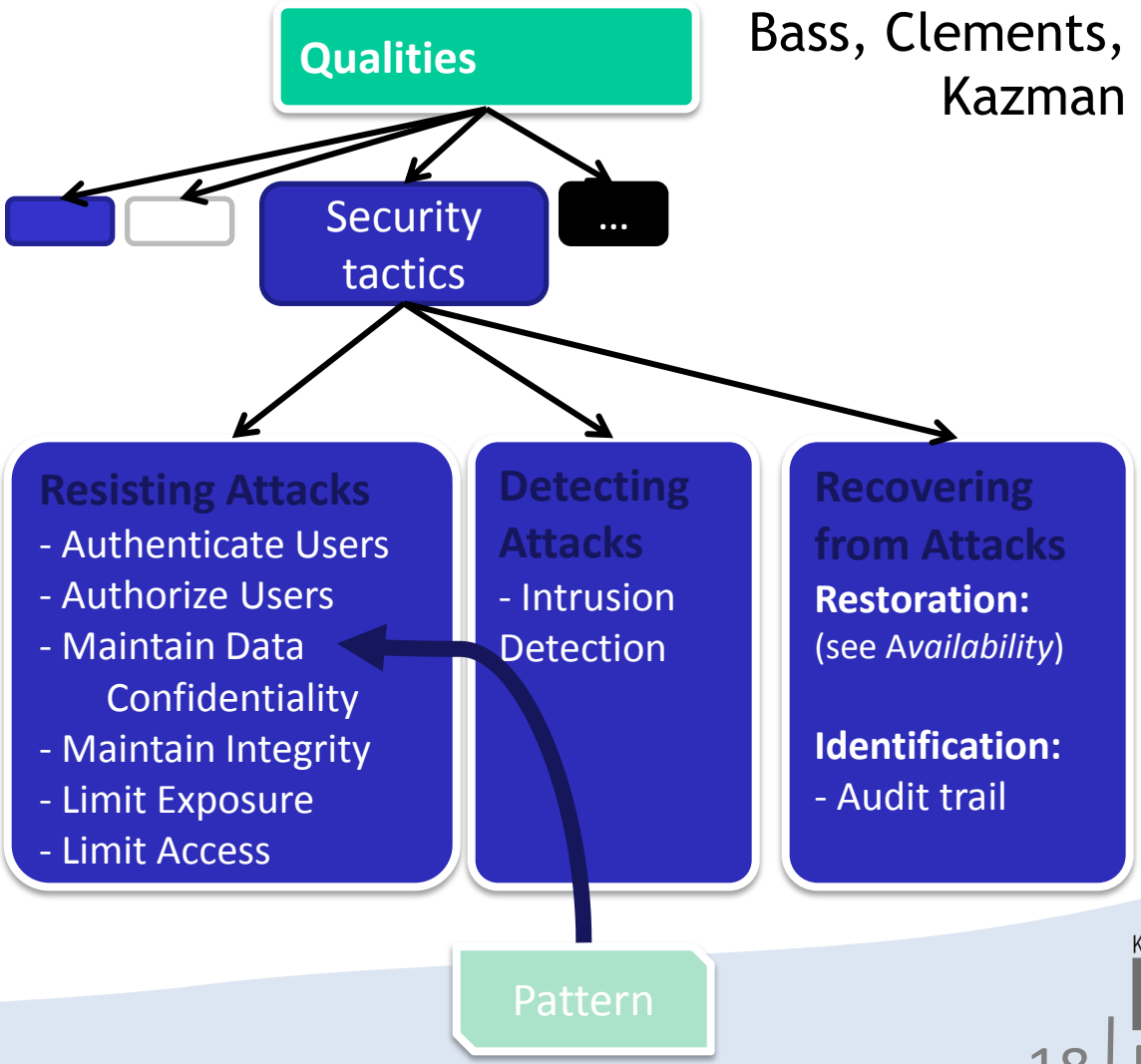
Environment:
At design time

Response:
Modification is made with no side effects

Response measure:
In 3 hours

Tactics to Control response

Creating SA
Tactics & patterns



Bass, Clements, Kazman

Creating SA
Algorithm

1. Choose the module to decompose
2. Refine the module
   a) Choose architectural drivers
   b) Choose architectural patterns (from strategy)
   c) Instantiate child modules and allocate functionality (from use cases). Document in multiple views
   d) Gap analysis
3. Repeat

Documenting SA
Architectural Views

- Views on human body ☺

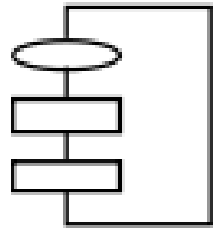
- An architectural view is a simplified description (abstraction) of a system
  - From a particular perspective
  - Covering particular concerns, and
  - Omitting entities that are not relevant to this perspective

Documenting SA
Architectural Views

- At least
  - Decomposition
  - Interaction
  - Deployment
- Mapping between views
  - Important
  - Hard

Documenting SA
Decomposition

Components



Module

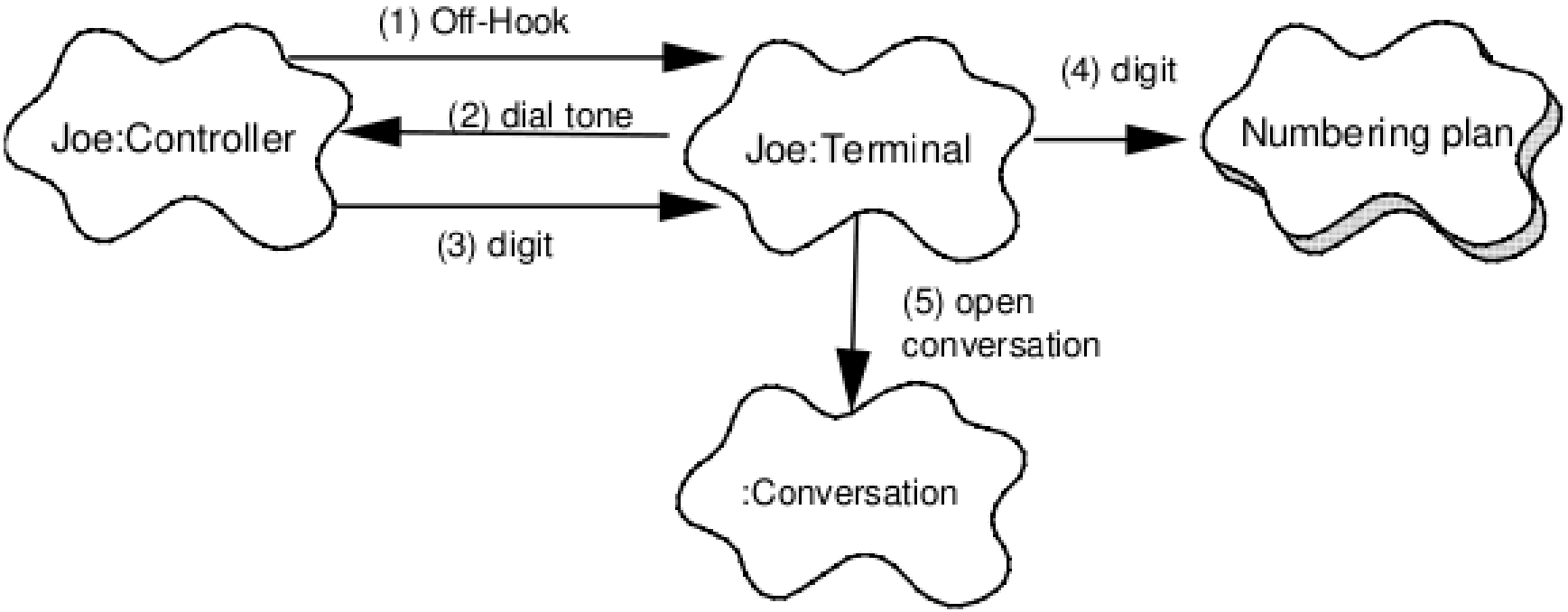
Subsystem

Layer

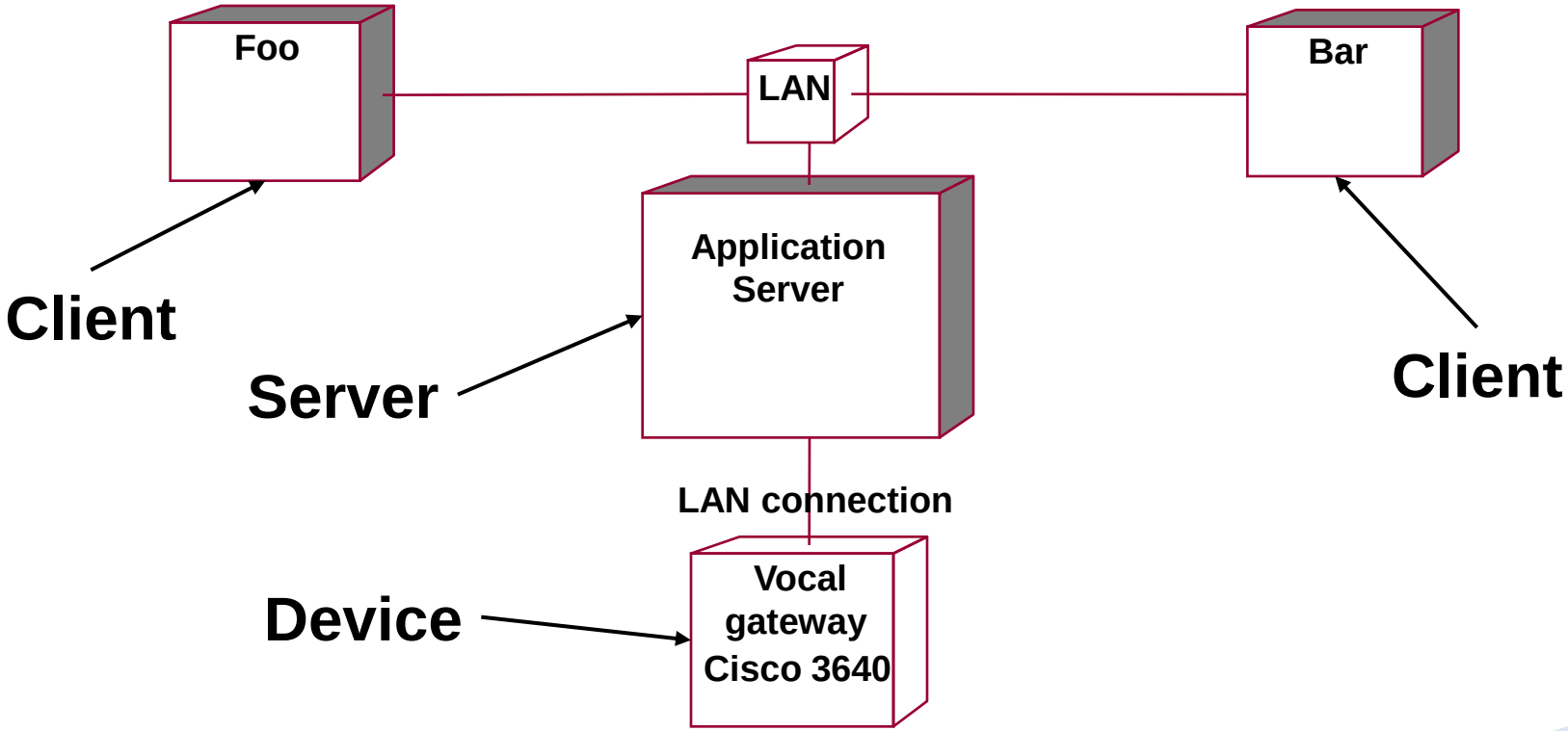
Connectors

Reference
Compilation dependency
(include, "with")

Documenting SA
Interaction

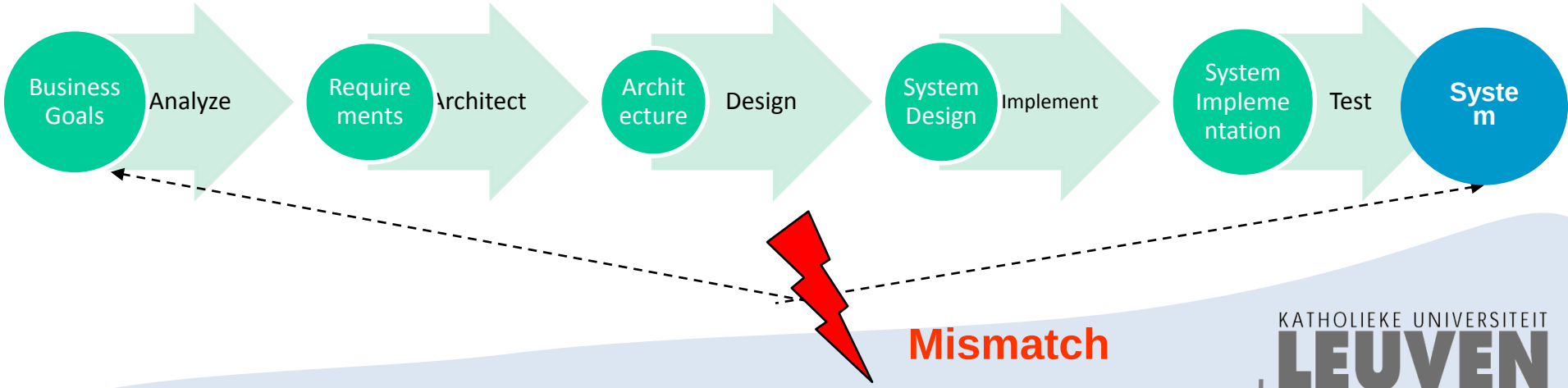


Documenting SA
Deployment

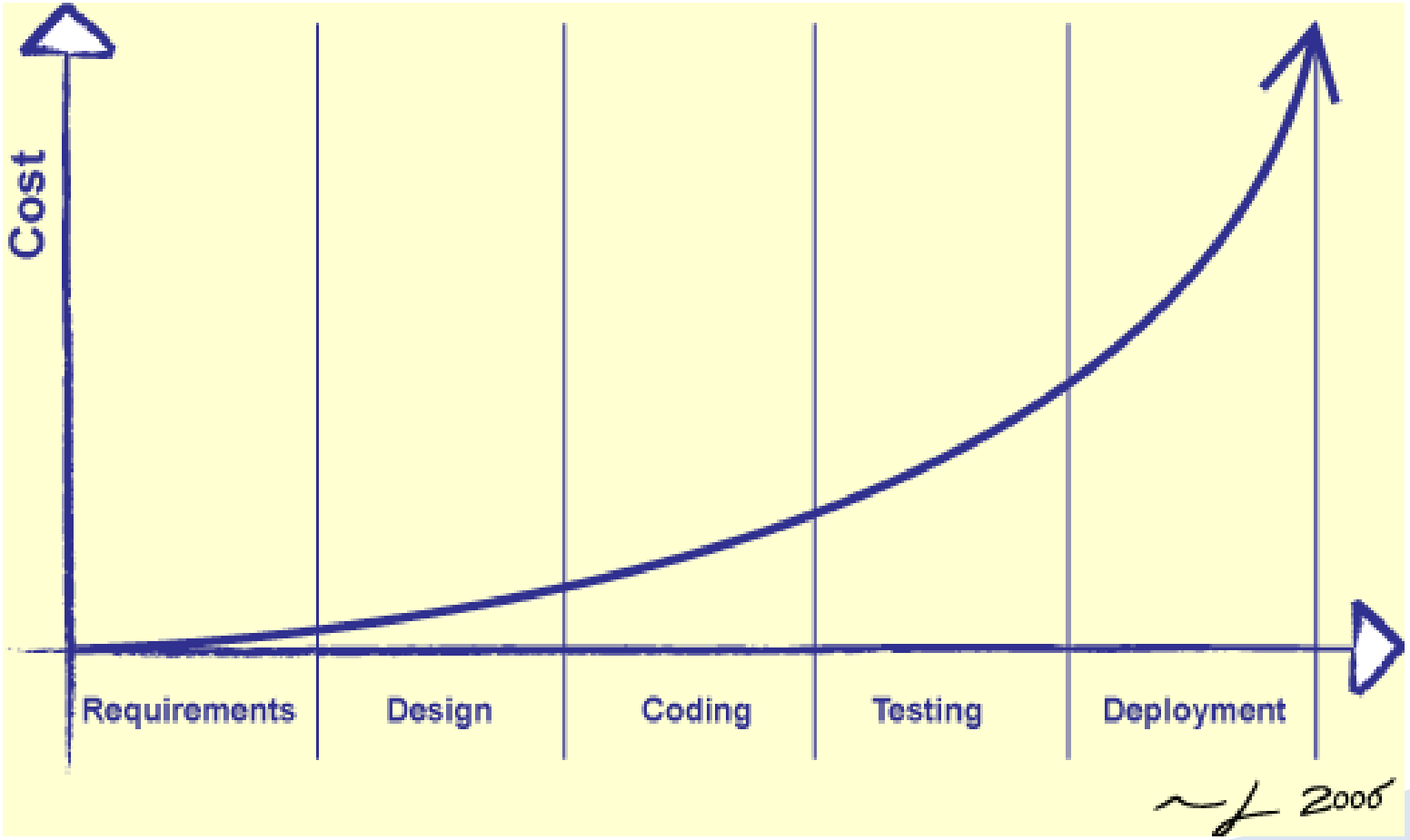


Evaluating SA
Motivation

- Creating the “right” system for a set of given requirements is still a general problem in software system development [SEI]

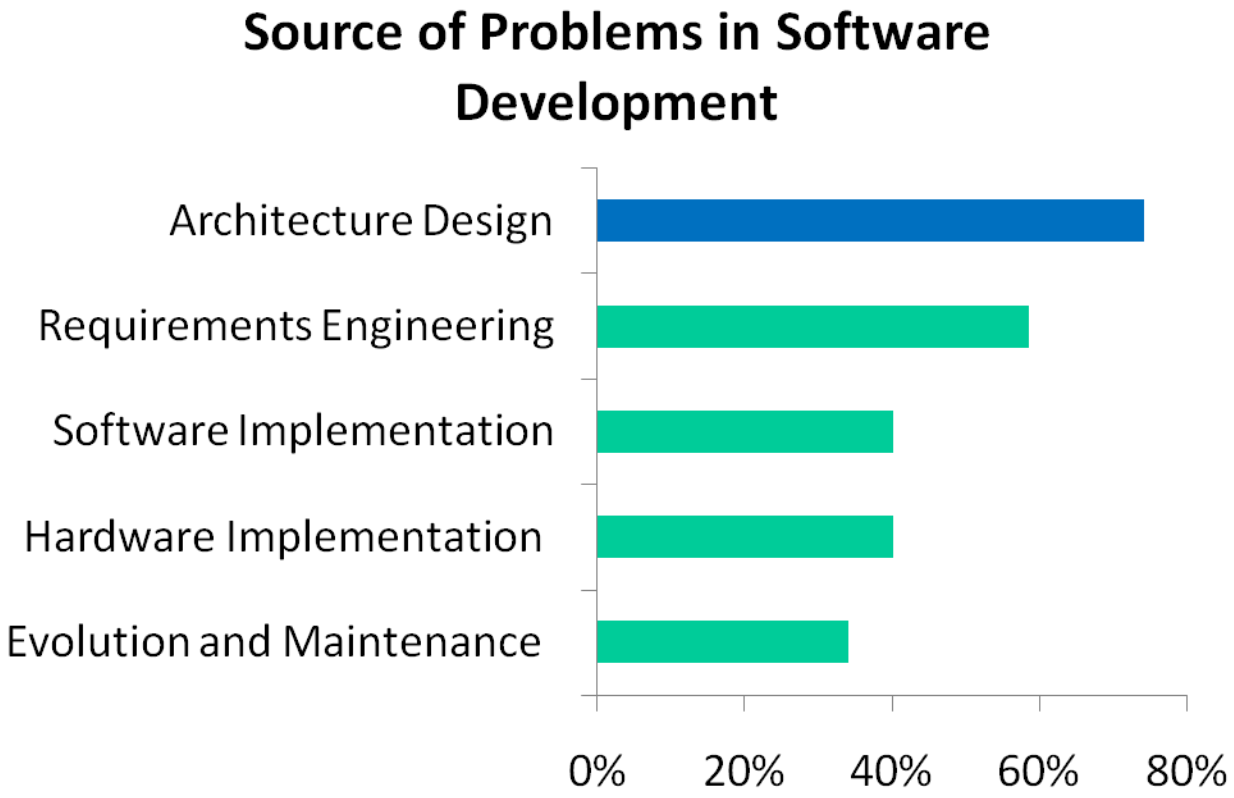


Evaluating SA
Boehm costs of change



Evaluating SA

Motivation



P. G. Neumann, *Computer-Related Risks*. Addison-Wesley, 1995

Evaluating SA
Output

- Is this architecture suitable for the system for which it was designed?
  - Resulting system will meet quality goals
  - System can be built using available resources
- Architectural risks
  - What are the weak points of the architecture?
- Architectural trade-offs
  - Choices are made explicit

Evaluating SA
Who's involved?

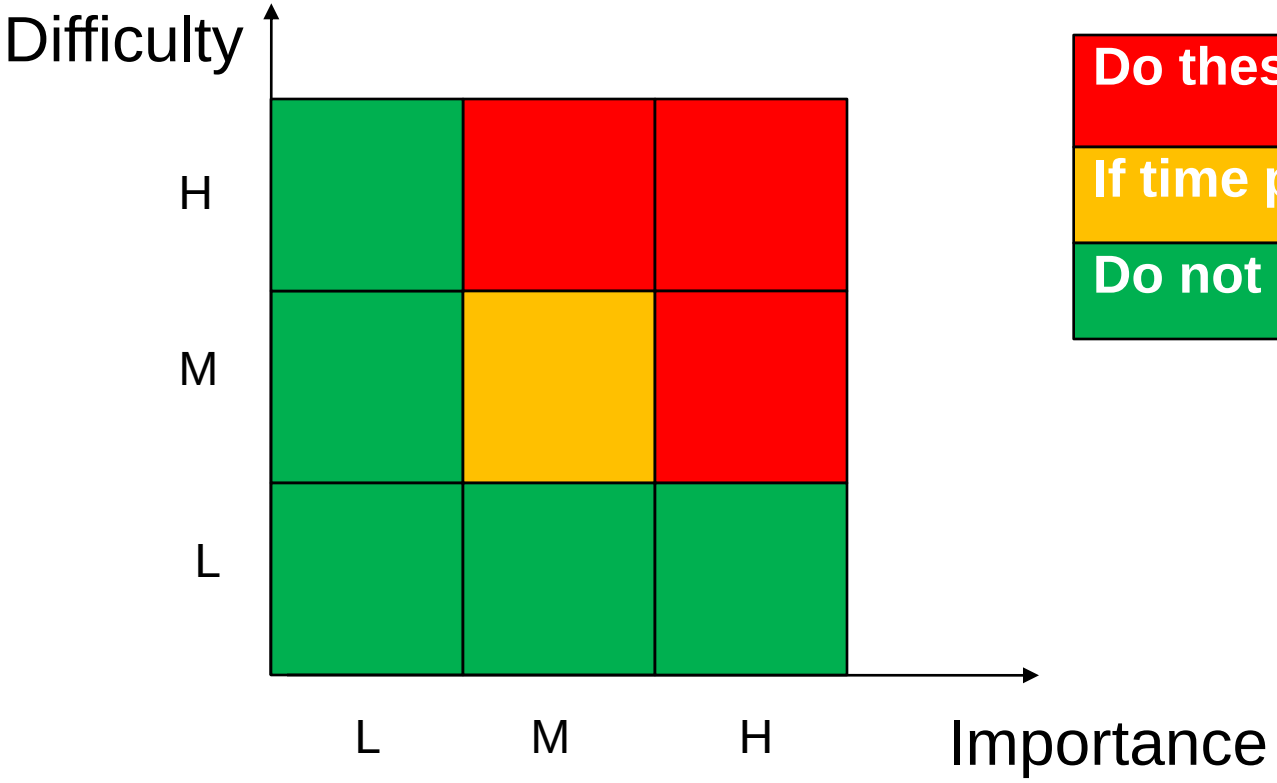
- Evaluation Team
  - Team leader
  - Evaluation leader
  - Scenario Scribe
  - Proceedings Scribe
  - Timekeeper
  - Questioner
- Customer Roles
  - Decision Maker
  - Software Architect
  - Other stakeholders

Evaluating SA
Architectural approaches

- Examples
  - Used a layered architecture
  - Use of distributed data
- I.e., architectural styles (patterns)
- Examples in security
  - Use of interception
  - Use of process separation
  - Use of single access point

Evaluating SA

Elicit and prioritize scenarios



Do these first

If time permits, do these

Do not do these

Evaluating SA
Analyze

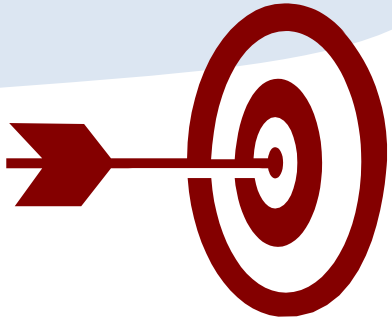
Scenario A8.1	Search, browse, and order submission is down less than 1 hour/week		
Attribute	Availability		
Architectural approaches	**Risk**	**Tradeoff**	**Nonrisk**
AD9 Backup copy of database on tape (not disk)	R9		

R9. Recovery from tape can take more than 1 hour in case of large amount of data

Act II
Security Architectures

KATHOLIEKE UNIVERSITEIT
LEUVEN
DistriNet

Objectives



- What Are Security Patterns?
- How to systematically bridge from security requirements (problem domain) to security-aware software architecture (solution domain)?

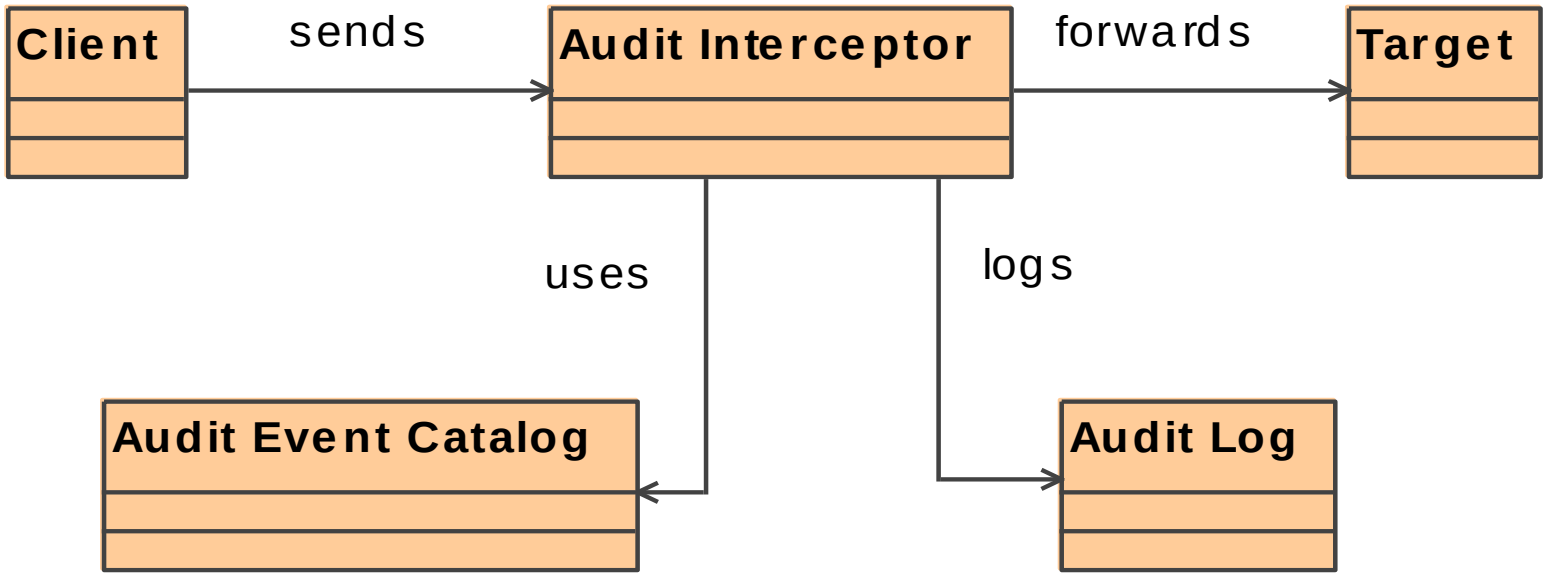
Security patterns

Well-known (and sound) solution for a recurring security problem, whose pros & cons are known in advance

- A (security) pattern describes… [Doug Lea]
  - a single kind of (security) problem
  - the solution as a constructible software entity
  - design steps for constructing the solution
- Potential helpful tools to implement security

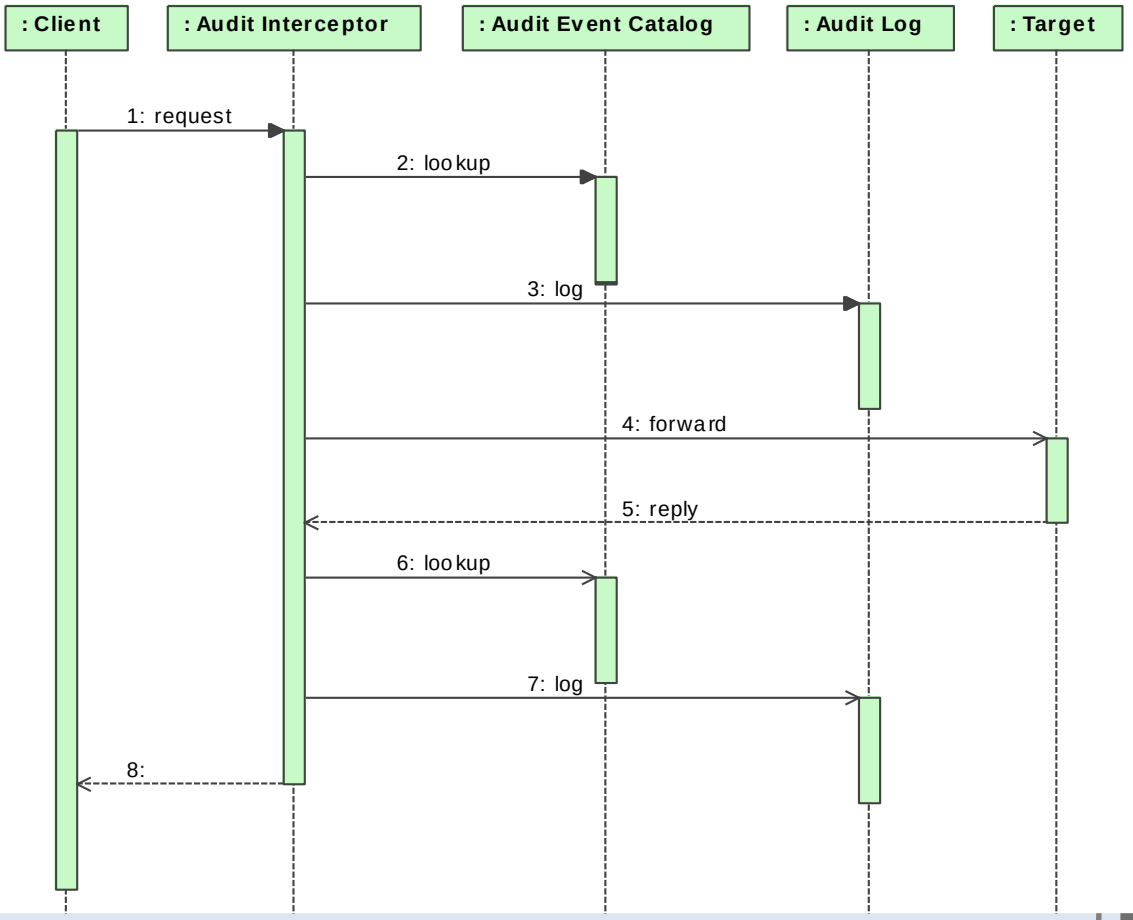
Example: Audit Interceptor

- Stucture



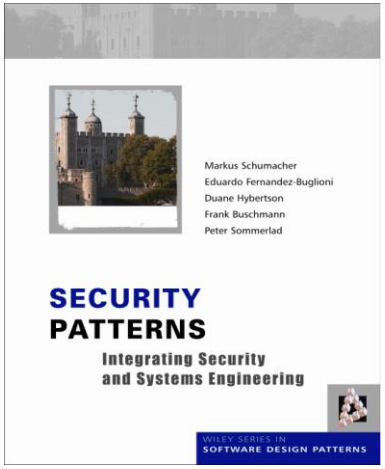
Example: Audit Interceptor

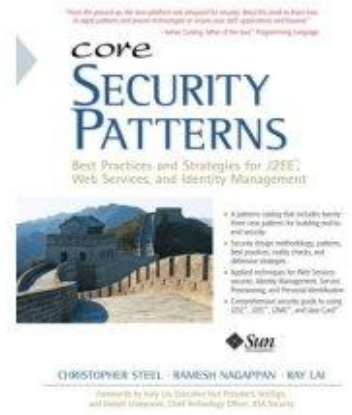
- Sequence Diagram



Existing inventories

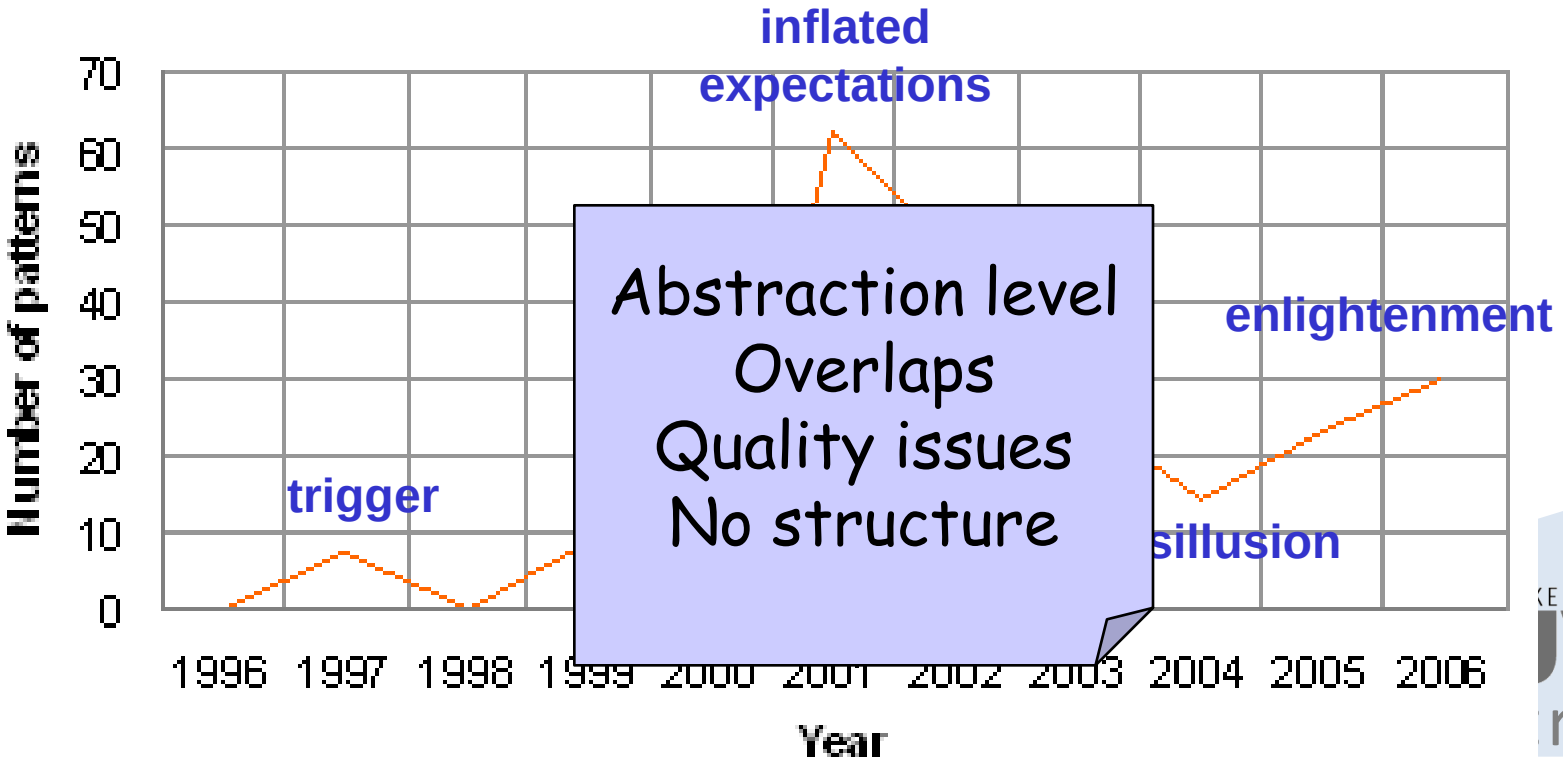
- Markus Schumacher, et al, Security Patterns: Integrating Security and Systems Engineering
- Christopher Steel, et al, Applied J2EE Security Patterns: Architectural Patterns and Best Practices





Security patterns landscape
Data set

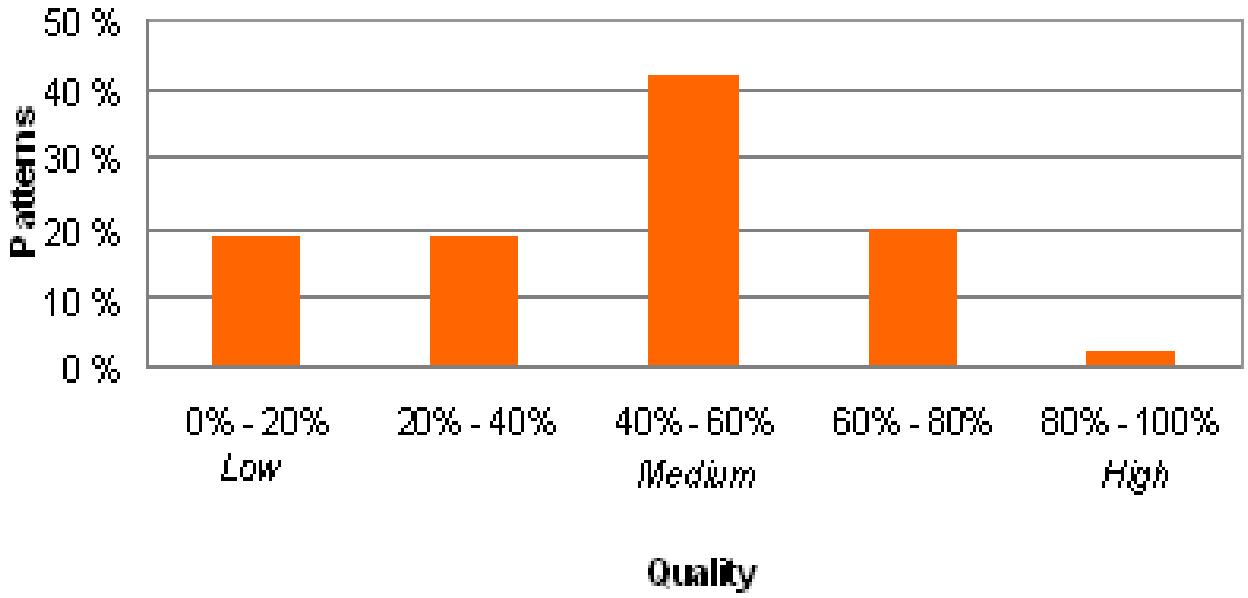
- 38 publications
- 218 patterns
- 1996-2006



Security patterns landscape Quality

- Grade pattern elements
  - Problem
  - Structure
  - Behavior
  - Consequenc
  - Example

$$Q = \Sigma\, w_i \, \frac{s_i}{max}$$



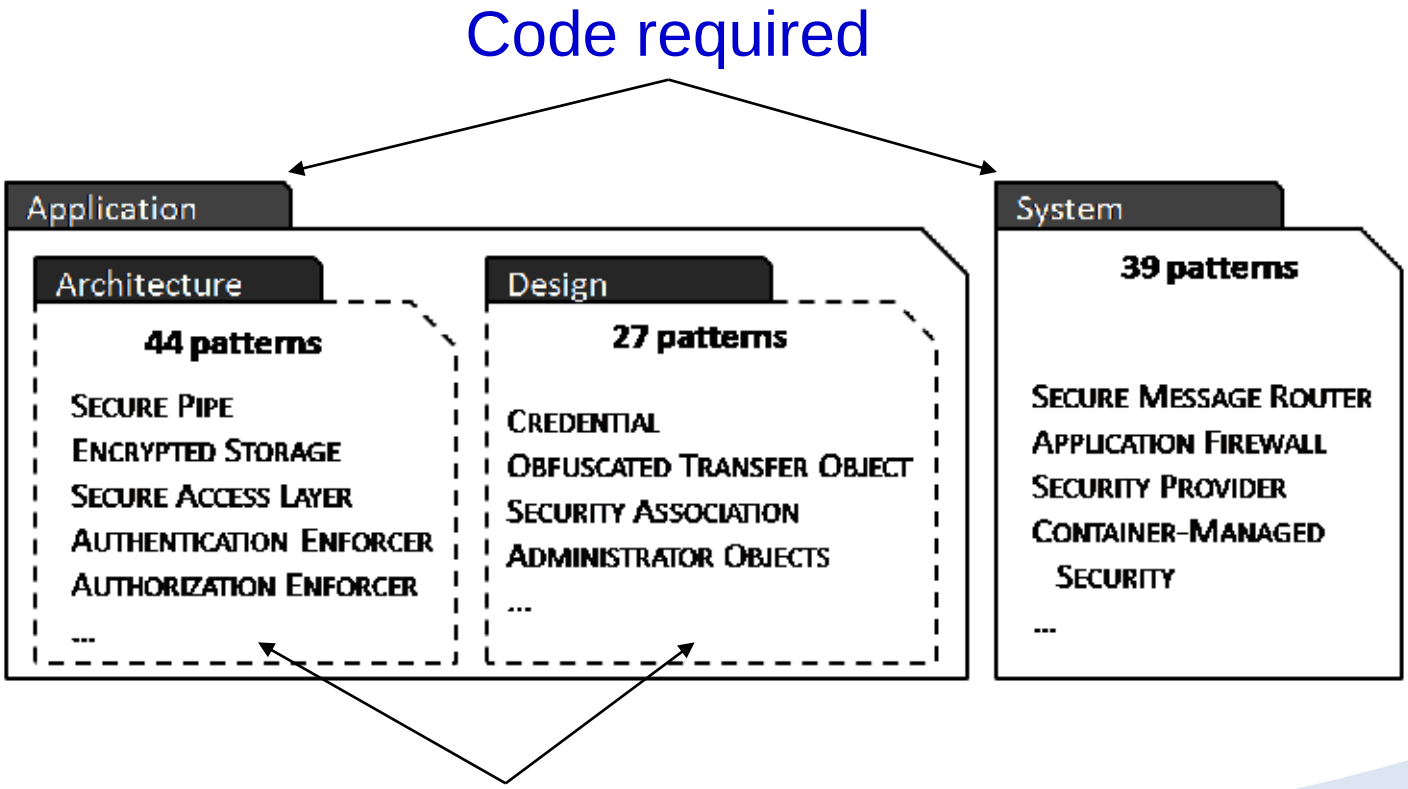
Problems & our approach

- Quality & quantity:
  - Not all published patterns are actual patterns
  - Overlapping/duplicate descriptions
  - Descriptions are lacking in detail
  - Essentially: too many unstructured patterns
- How to choose and implement the right pattern?
  - … reading them all?
    - done that, not recommendable ;)
- Our approach:
  - Collect good patterns in a structured inventory
  - Integrate selection in software engineering process

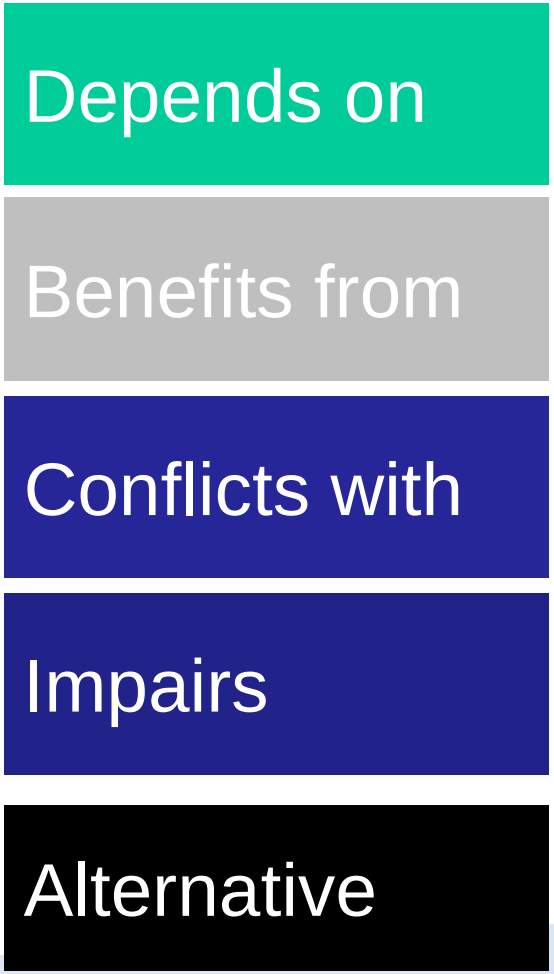
Security patterns catalog Overview

- Abstraction level
  - Categorization
- Quality
  - Template
- Overlaps
  - Grouping
- No structure
  - Inter-pattern relations
- Support for methodology
  - Security objectives
  - Trade-off labels

Security patterns catalog Categorization



Security patterns catalog
Relations

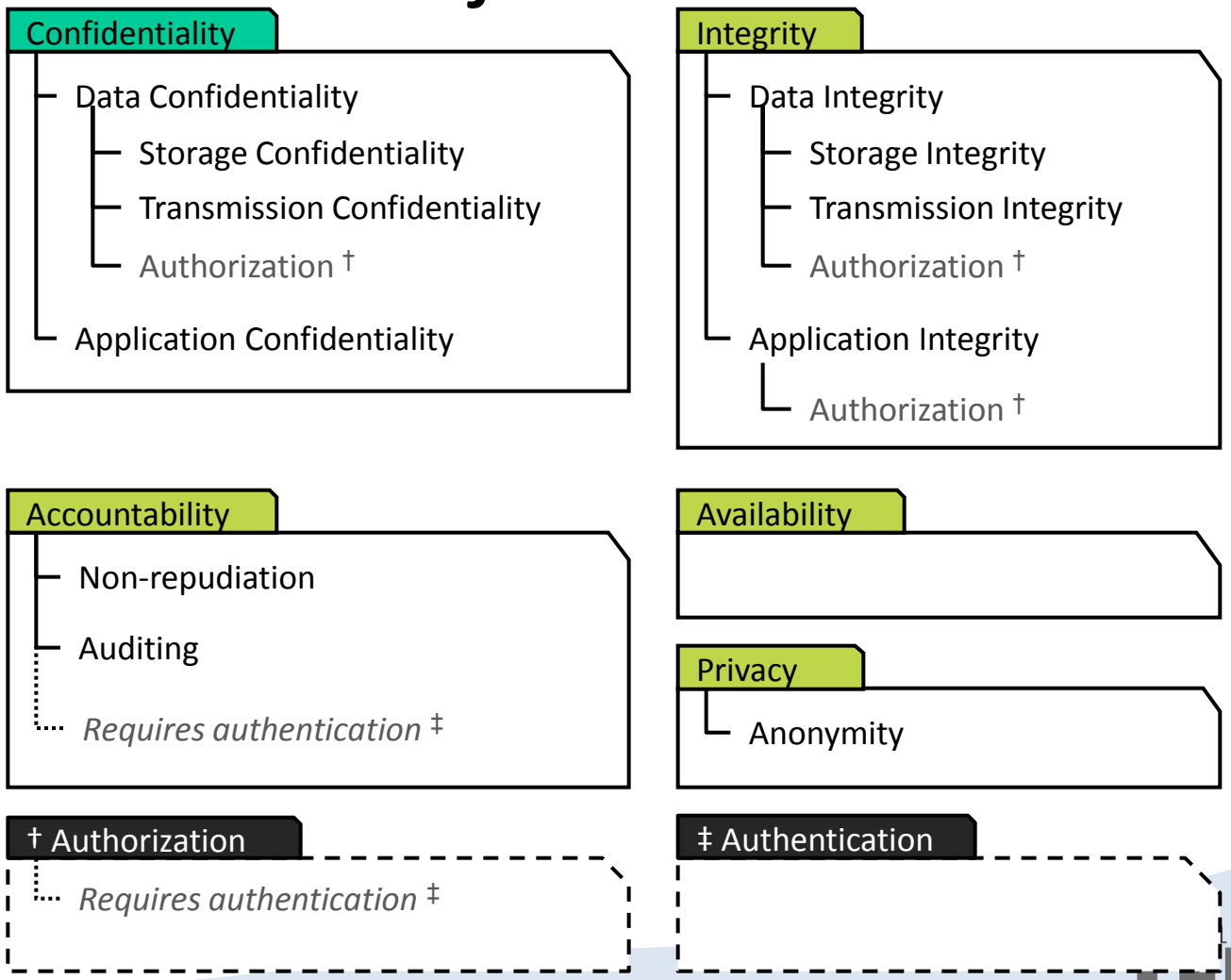


Security patterns catalog
Relations – In practice

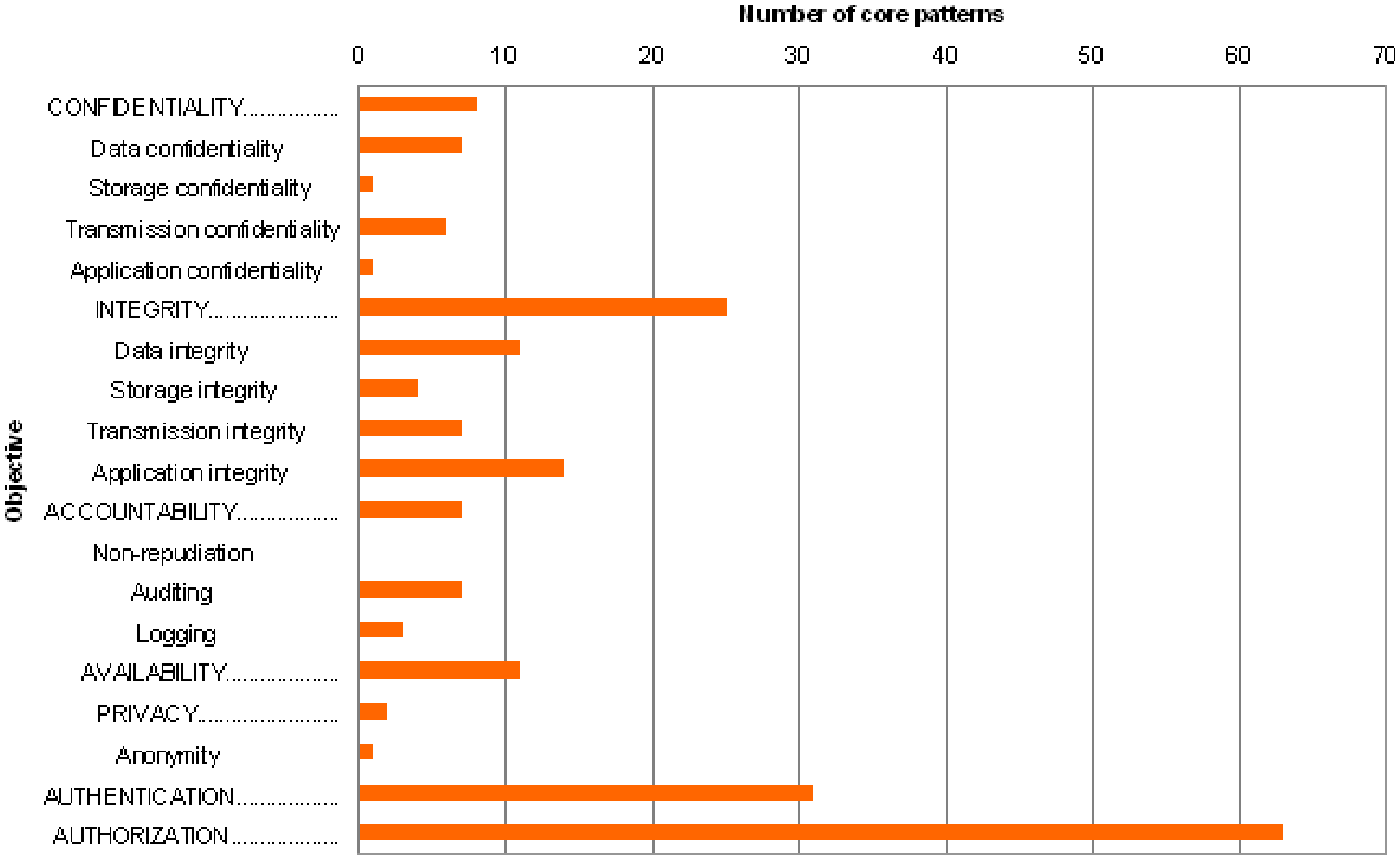
	System	Firewall	Single Access Point	*Appl. Architecture*	Authent. Enforcer	Authoriz. Enforcer	Secure Logger	*Applic. Design*	Security Association	Limited View	Full View w/ Errors	Session
System												
Demilitarized Zone		D										
Secure Pipe									B			
Load Balancer												I
Audit Interceptor							D					
Application Architecture												
Authentication Enforcer			B			B						
Application Design												
Limited View											A,C	
Full View with Errors										A,C		

UNIVERSITEIT VEN Net

Security patterns catalog
Objectives



Security patterns catalog
Objectives – In practice



Security patterns catalog

Trade-off labels

ISO 9126
- Dependability
- Portability
- Maintainability
- Performance
- Usability

Security Objectives
- Confidentiality
- Integrity
- Accountability
- Availability

CC
- Manageability
- Auditability

Business
- Cost

Denote strong points and weaknesses, e.g. Audit Interceptor:

- Performance

+ Accountability

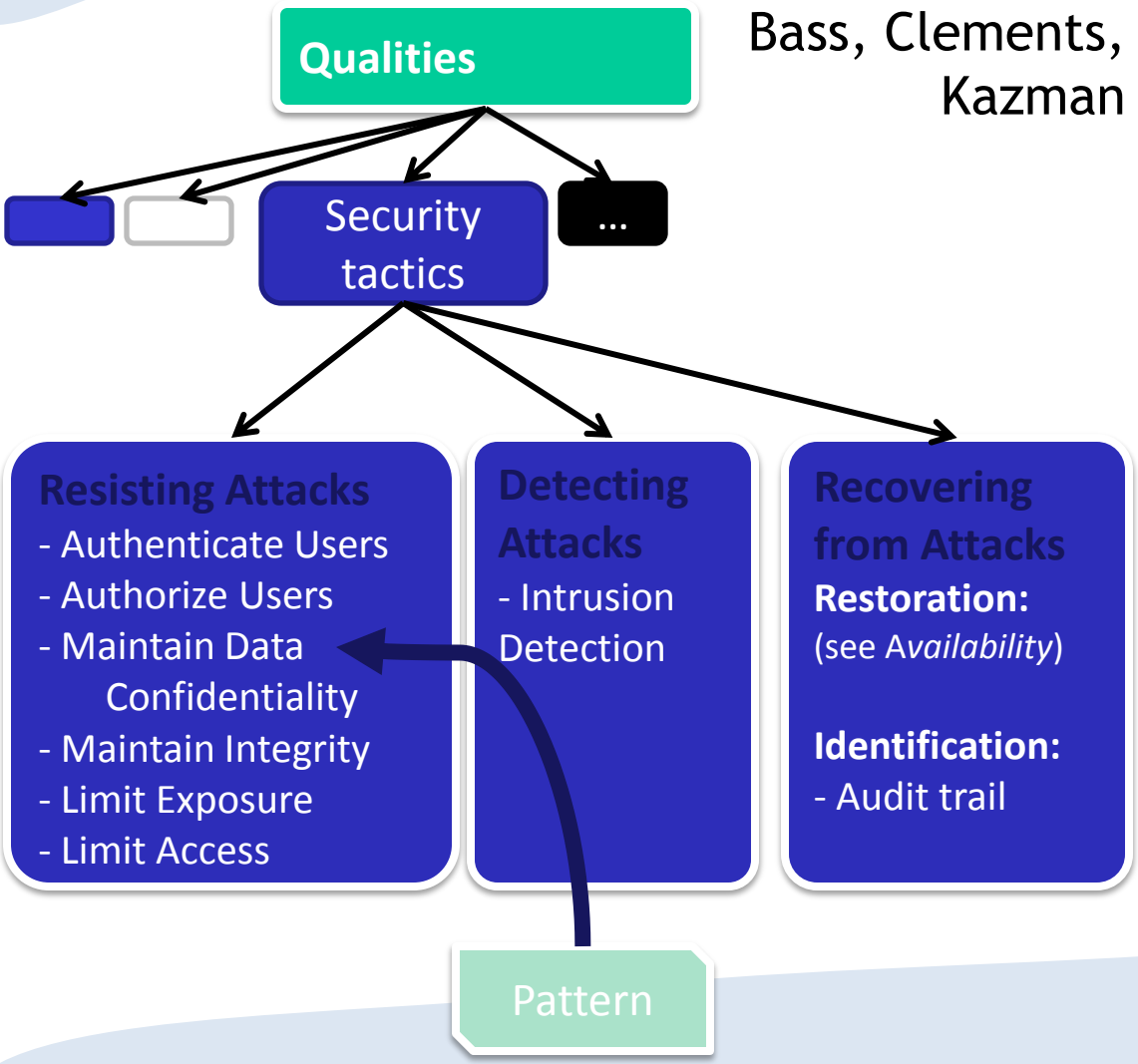
Security patterns catalog
Bringing it together

Pattern Name
Intent
Also known as (optional)
Applicability
Security objective
Labels
Relationships
- *Dependencies*
- *Impairments*
- *Conflicts*
- *Benefits*
- *Alternatives*

1. Problem
   - *Forces*
2. Example
3. Solution
   - *Structure*
   - *Dynamics*
   - *Participants*
   - *Collaborations*
4. Implementation (optional)
5. Pitfalls (optional)
6. Consequences
7. Related patterns
8. Known uses

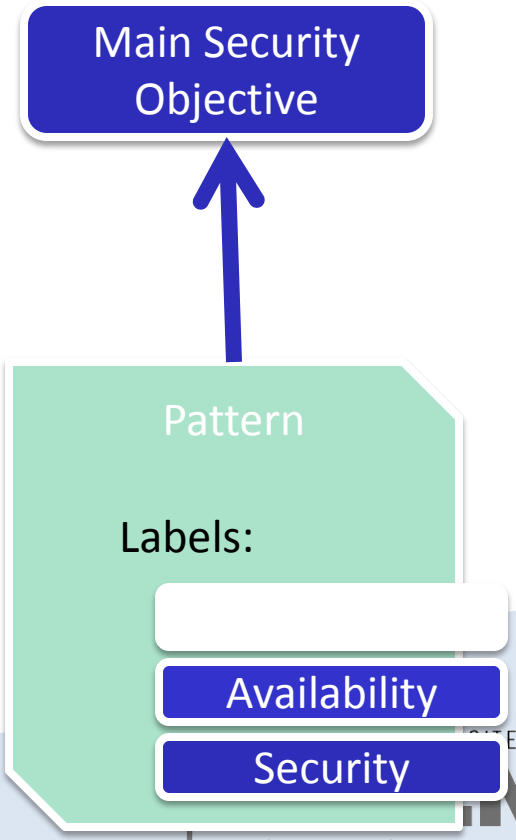
- Purpose: uniformly describing patterns
- Ensures that all relevant data is included
- Summarizes information for quick selection

Attribute-driven design

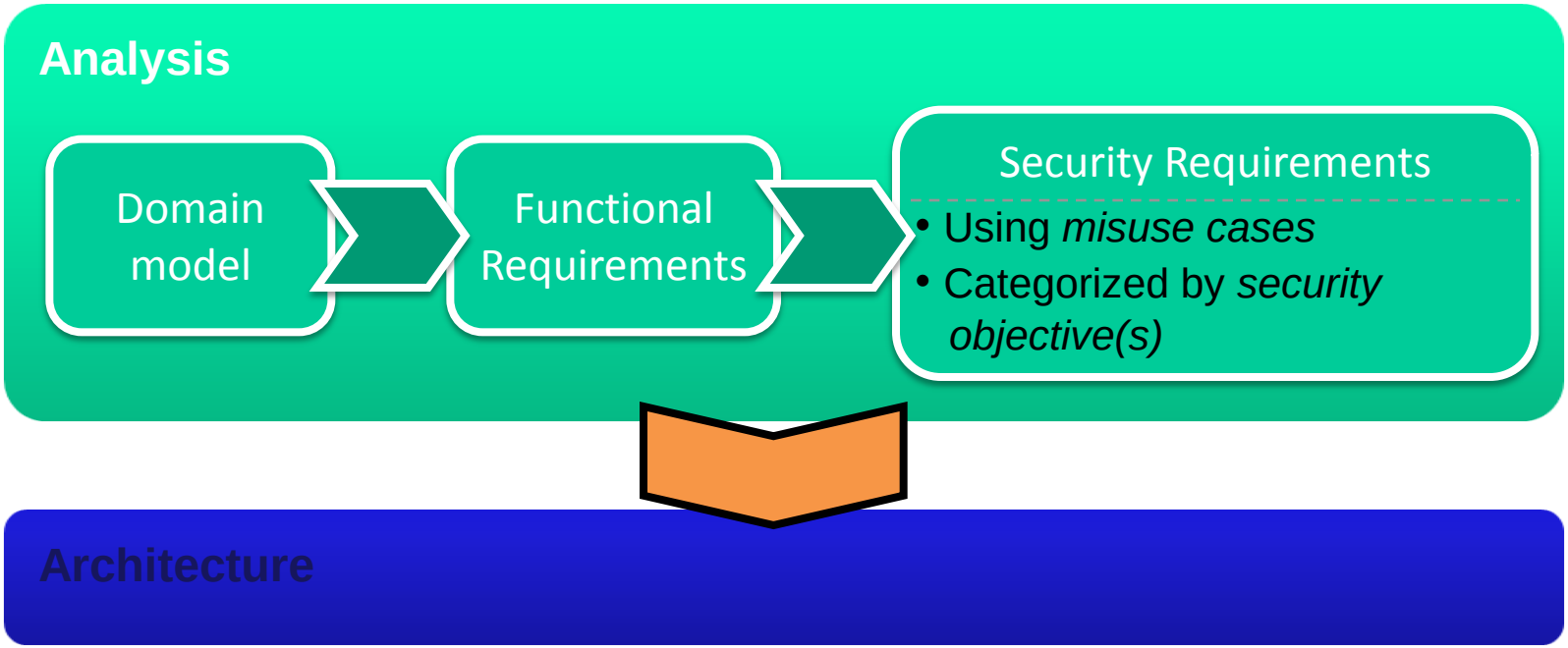


Bass, Clements, Kazman

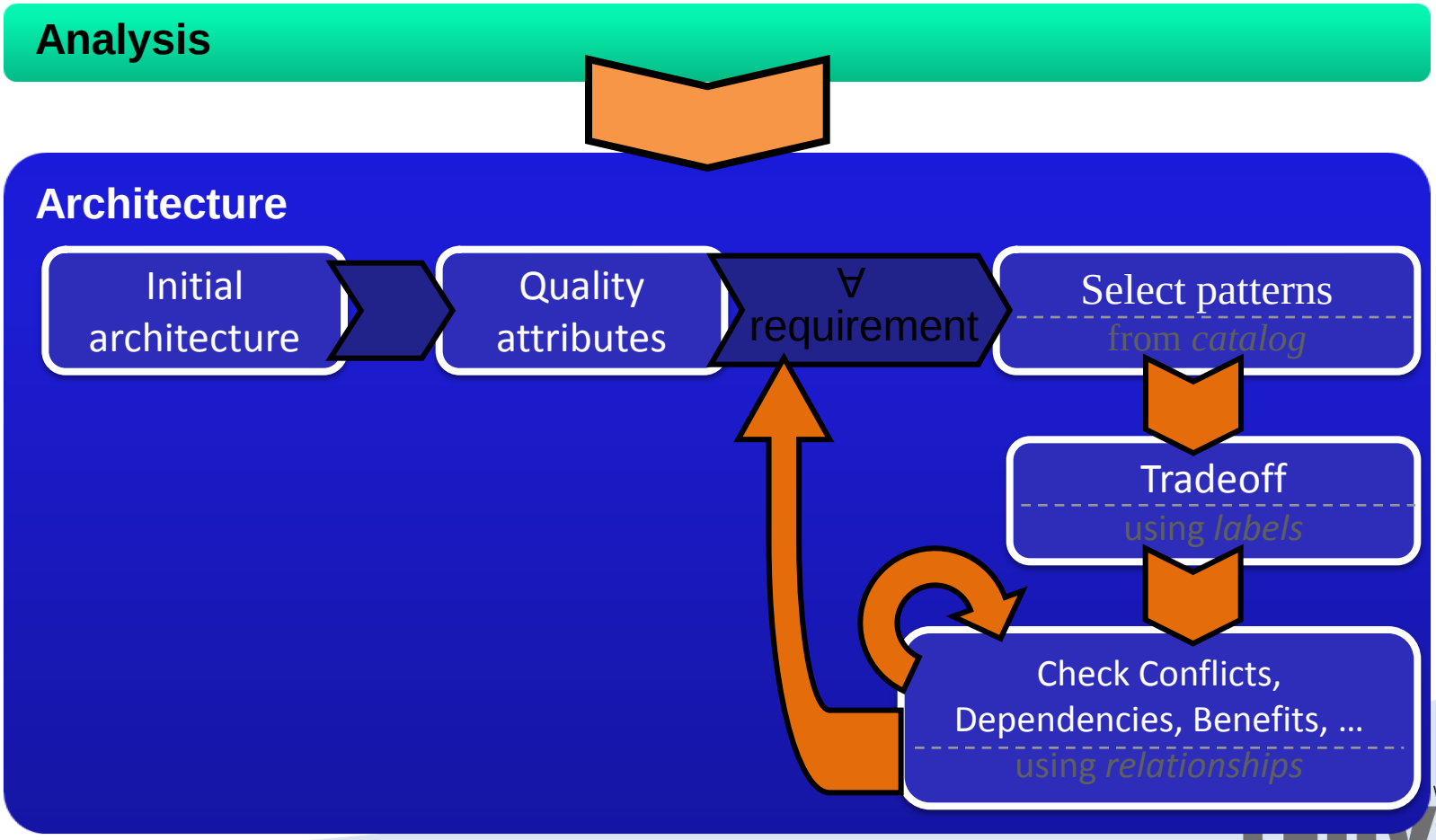
Our approach



Methodology
Analysis



Methodology
Architecture (inspired by ADD)



Methodology
Experimentation

	Functional components	MUCs	Patterns	Extra components
Calendar	2	5	5	2
ATM	5	8	9	10
E-health	7	92	13	10

Digital Publication System: new experiment this year, with students (including evaluation)

Final rehearsal

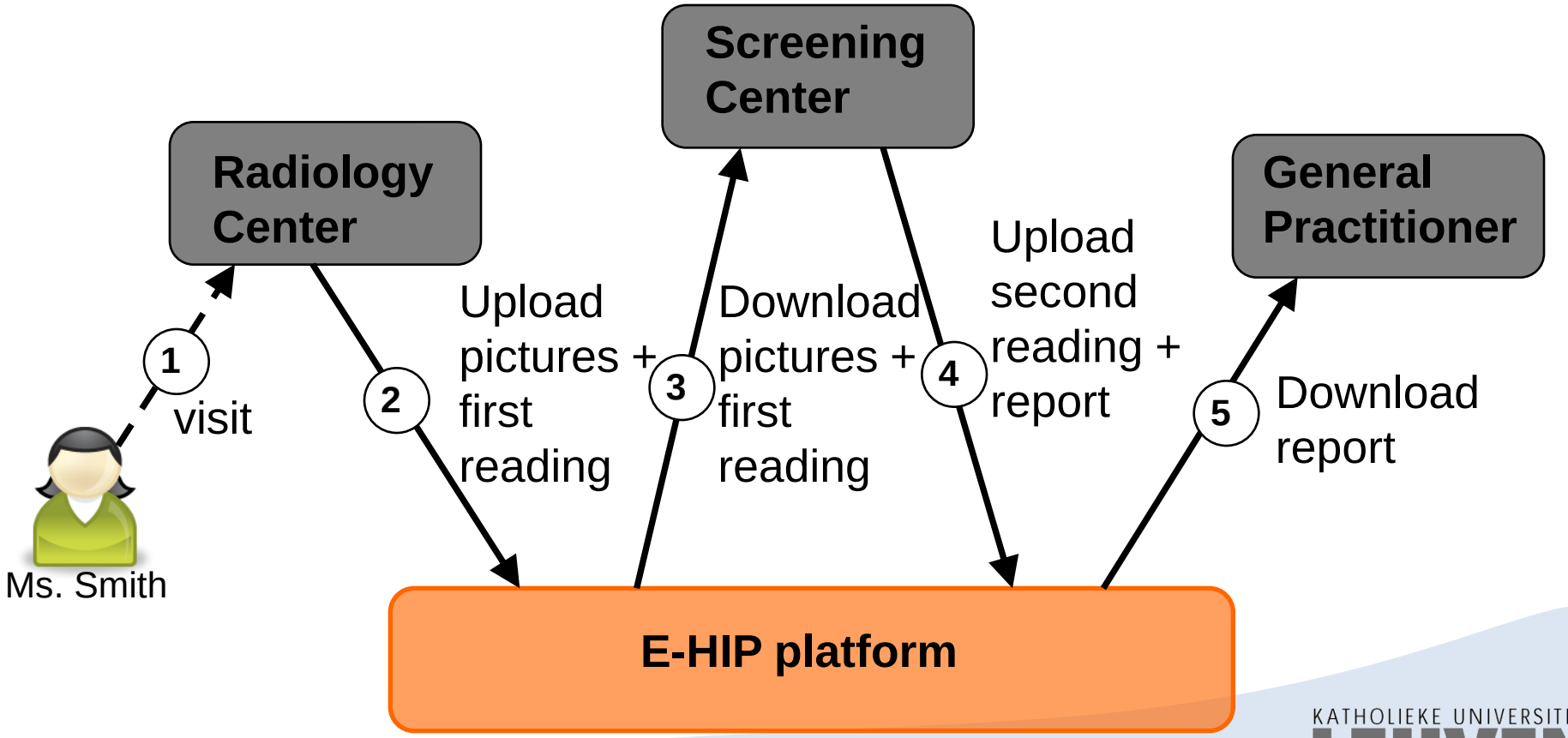
Case study

KATHOLIEKE UNIVERSITEIT
LEUVEN
DistriNet

E-Health Information Platforms

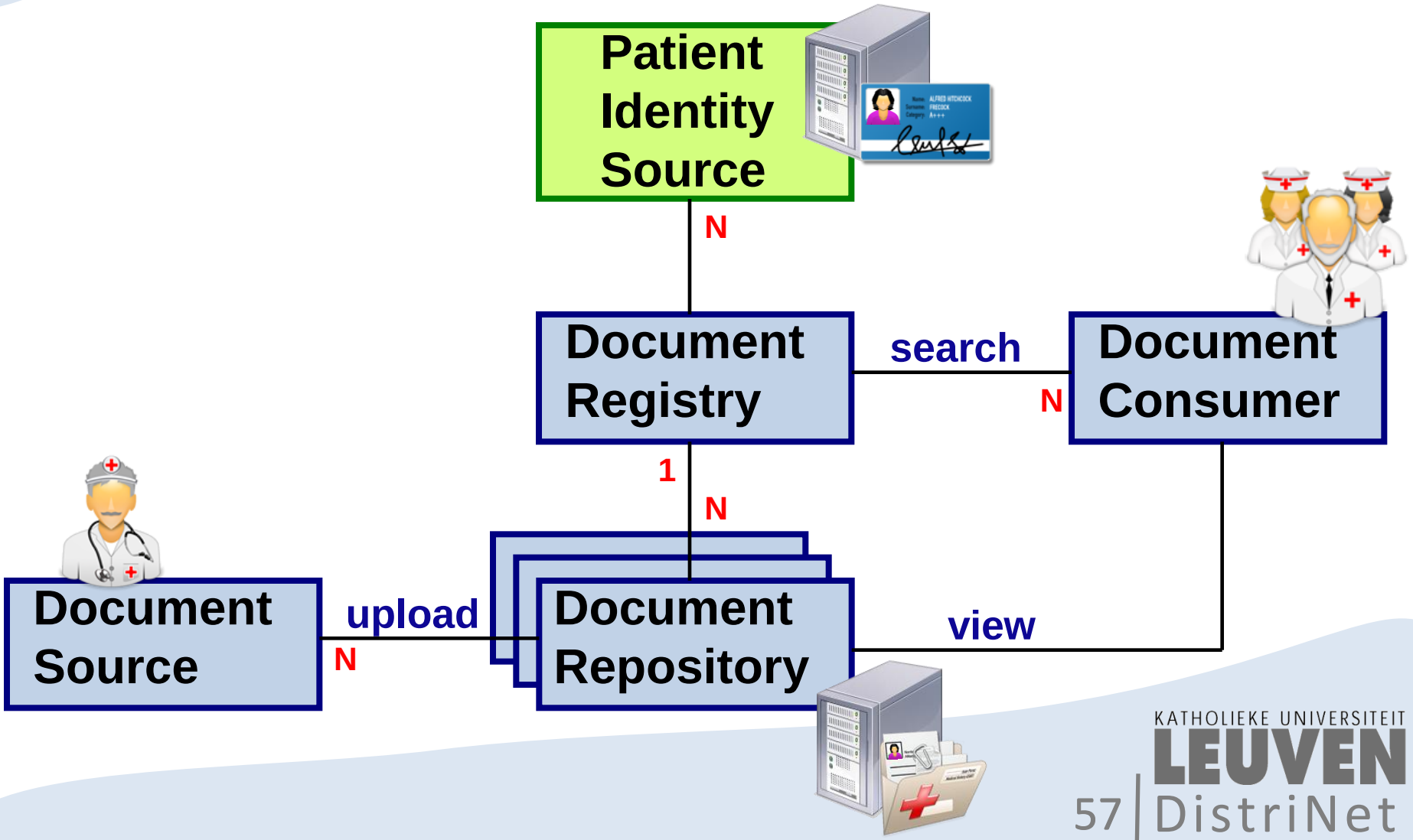
- Distributed health-care providers in Flanders
  - Hospitals, general practitioners, others
  - Large amount of data and proprietary systems
- Federated IT infrastructure
  - Enables smooth collaboration
  - Patient-centric
  - Access to data anytime, anywhere

E-HIP: example scenario
Mammo screening



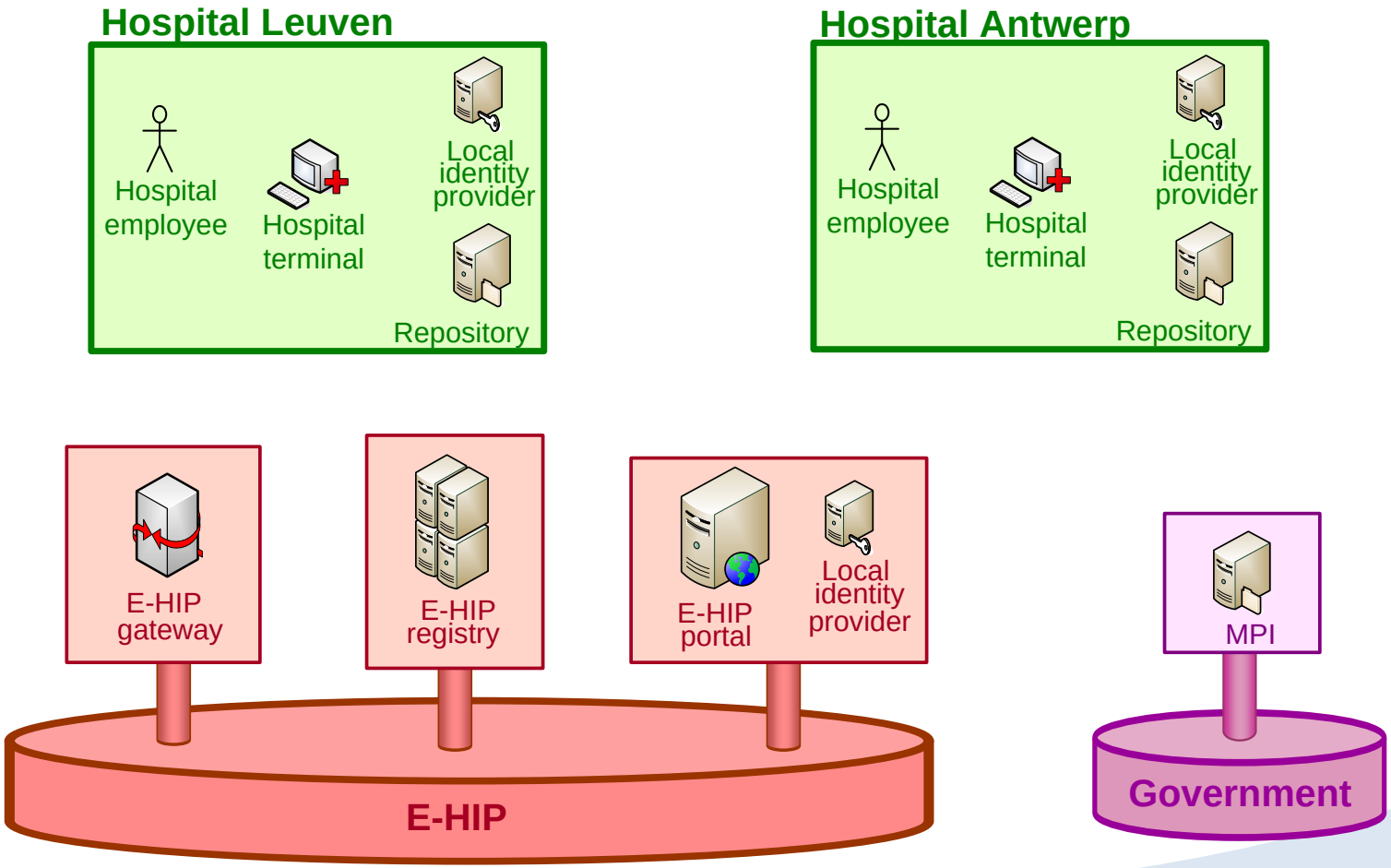
IHE-XDS

Reference model



Methodology

Start with initial architecture



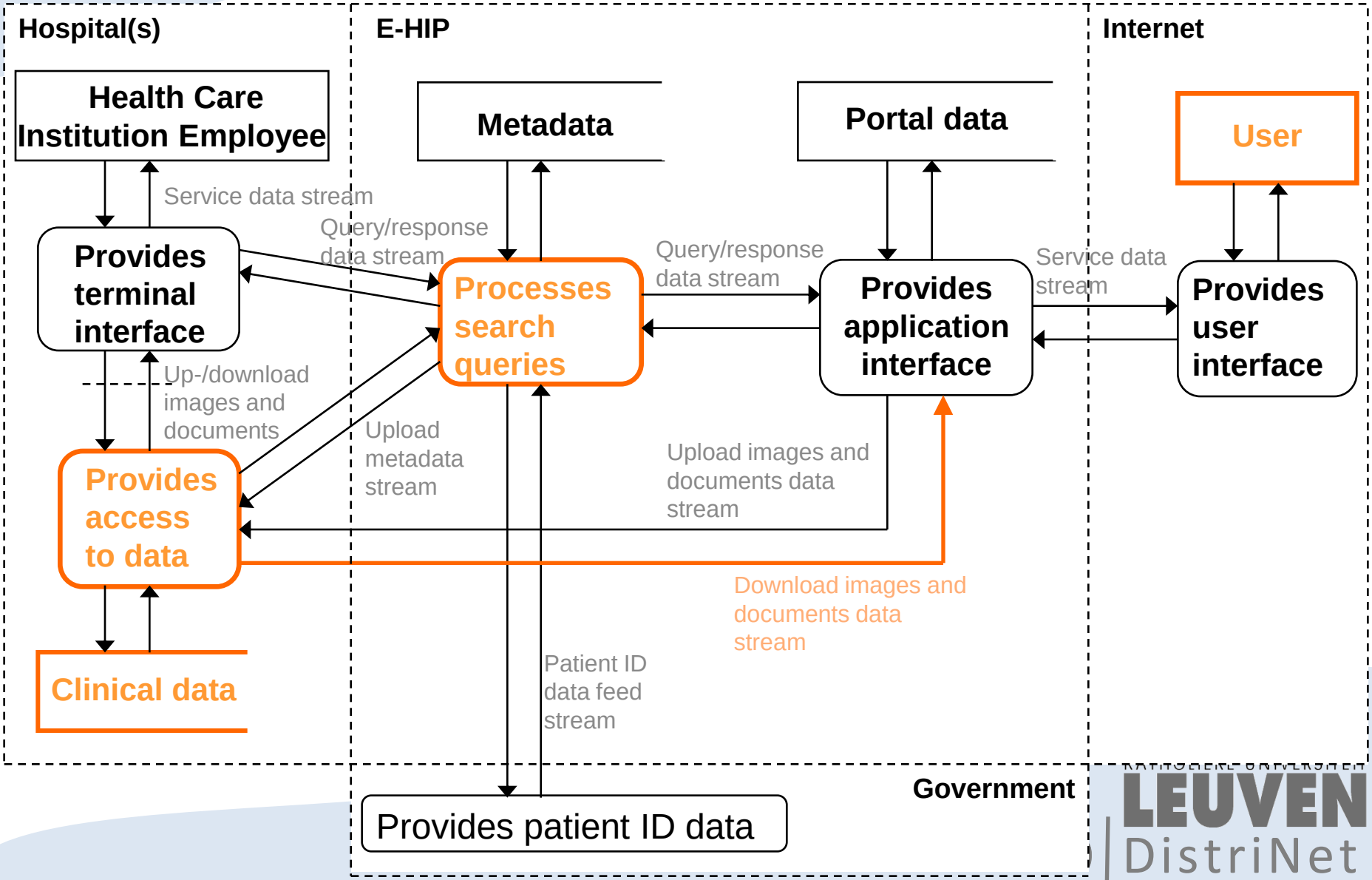
Security analysis
Architecture level

- Threat modeling using STRIDE
  1. Model architecture as Data Flow Diagram (DFD)
  2. Determine threats by using STRIDE

  - **S**poofing
  - **T**ampering
  - **R**epudiation
  - **I**nformation disclosure
  - **D**enial of service
  - **E**levation of privilege

M. Howard and S. Lipner, The Security Development Lifecycle. Microsoft Press, 2006.

DFD


Hospital(s)
E-HIP
Internet
Health Care Institution Employee
Metadata
Portal data
User
Service data stream
Query/response data stream
Provides terminal interface
Processes search queries
Query/response data stream
Provides application interface
Service data stream
Provides user interface
Up-/download images and documents
Upload metadata stream
Upload images and documents data stream
Provides access to data
Download images and documents data stream
Clinical data
Patient ID data feed stream
Government
Provides patient ID data

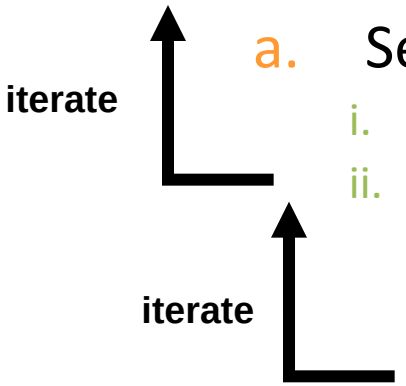
Security analysis
Results

- 86 MUCs
- Security assumptions, architectural similarities
  - No-deletion policy
  - Reuse solution for repository (data) to registry (meta)
- 14 MUCs left

- Gap analysis (business level misuse cases)
  - Consider how XDS/EHIP functionality can be misused
- 6 additional MUCs

Memo

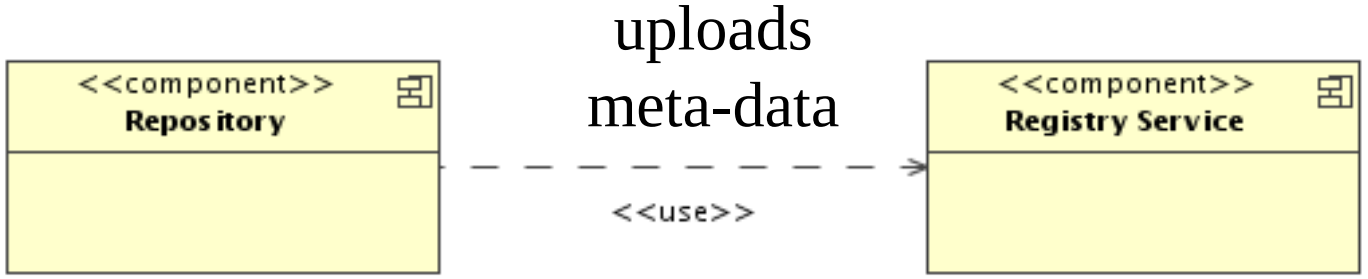
1. Start with initial architecture
2. Tag MUC's with security objective(s)
3. Prioritize security objectives
4. Select security objective from prioritized list
   a. Select pattern associated with objective
      i. Trade-off based on quality labels
      ii. Take into account benefits, dependencies, impairments and conflicts



Labeling MUCs

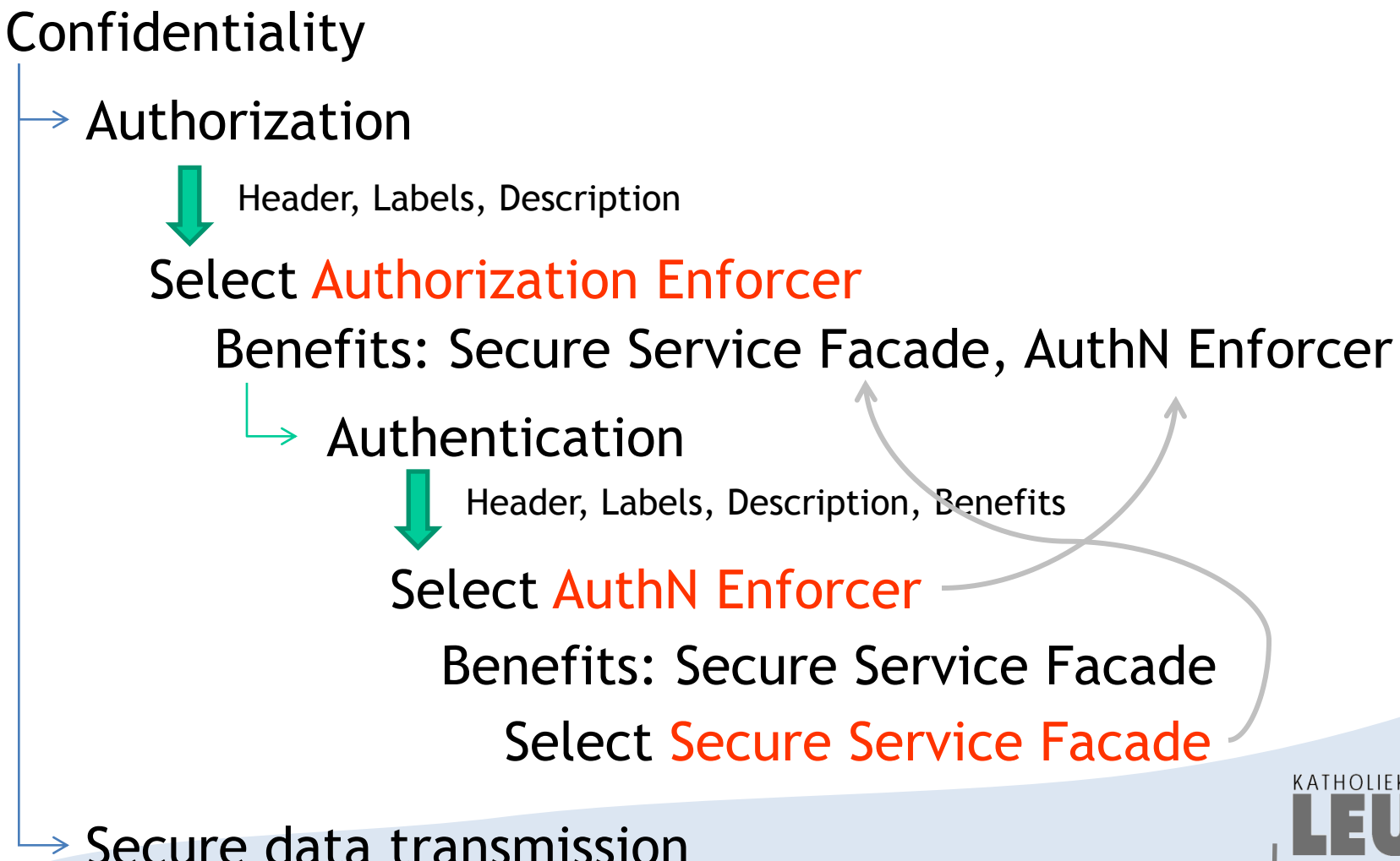
Threat	Mitigation Feature
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation
Information Disclosure	Confidentiality
Denial of Service	Availability
Elevation of Privilege	Authorization

Initial architecture

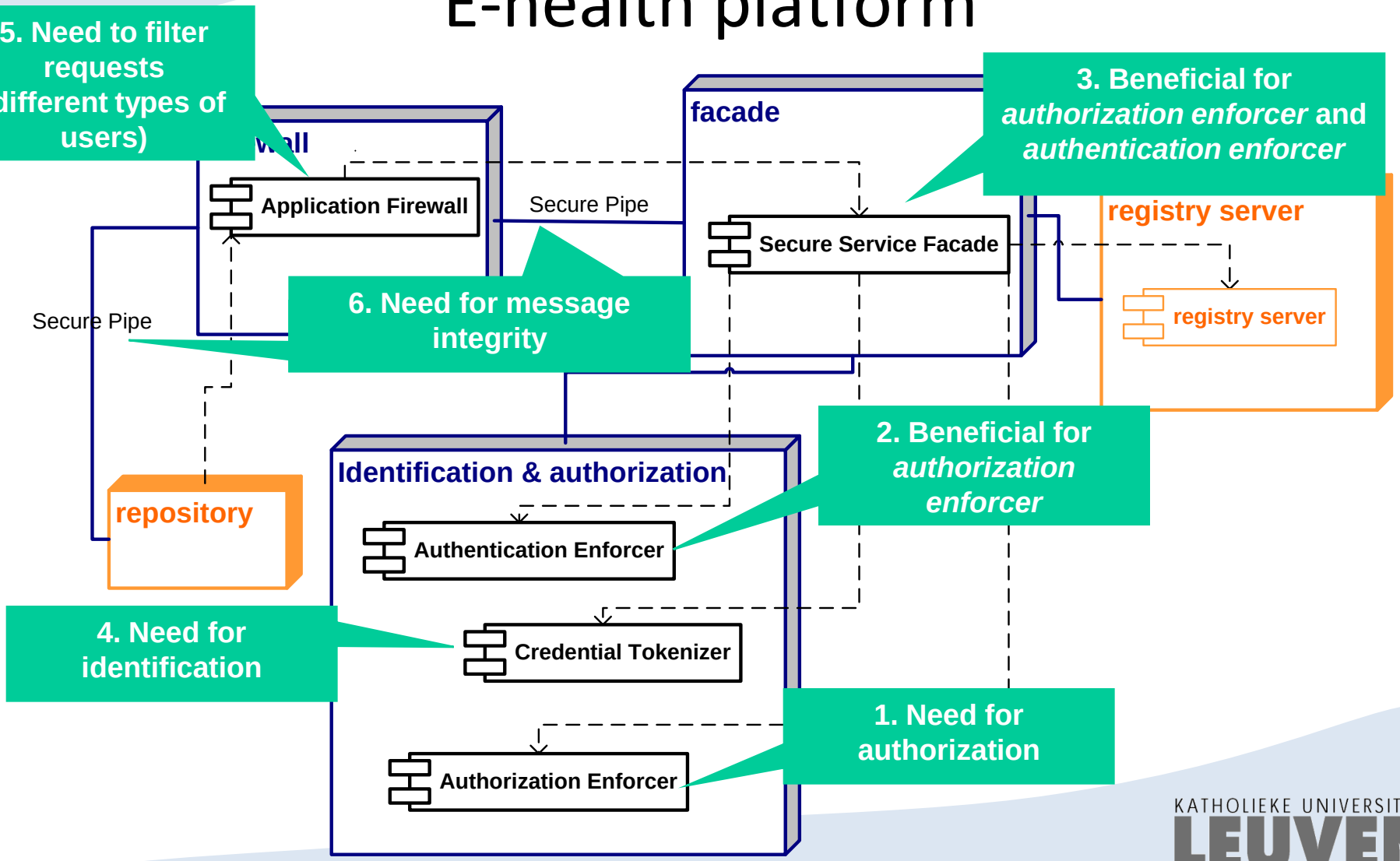


- Important qualities: manageability and auditing
- First security objective: confidentiality
  - Is composed of controlled access and secure data transmission
  - We start with controlled access

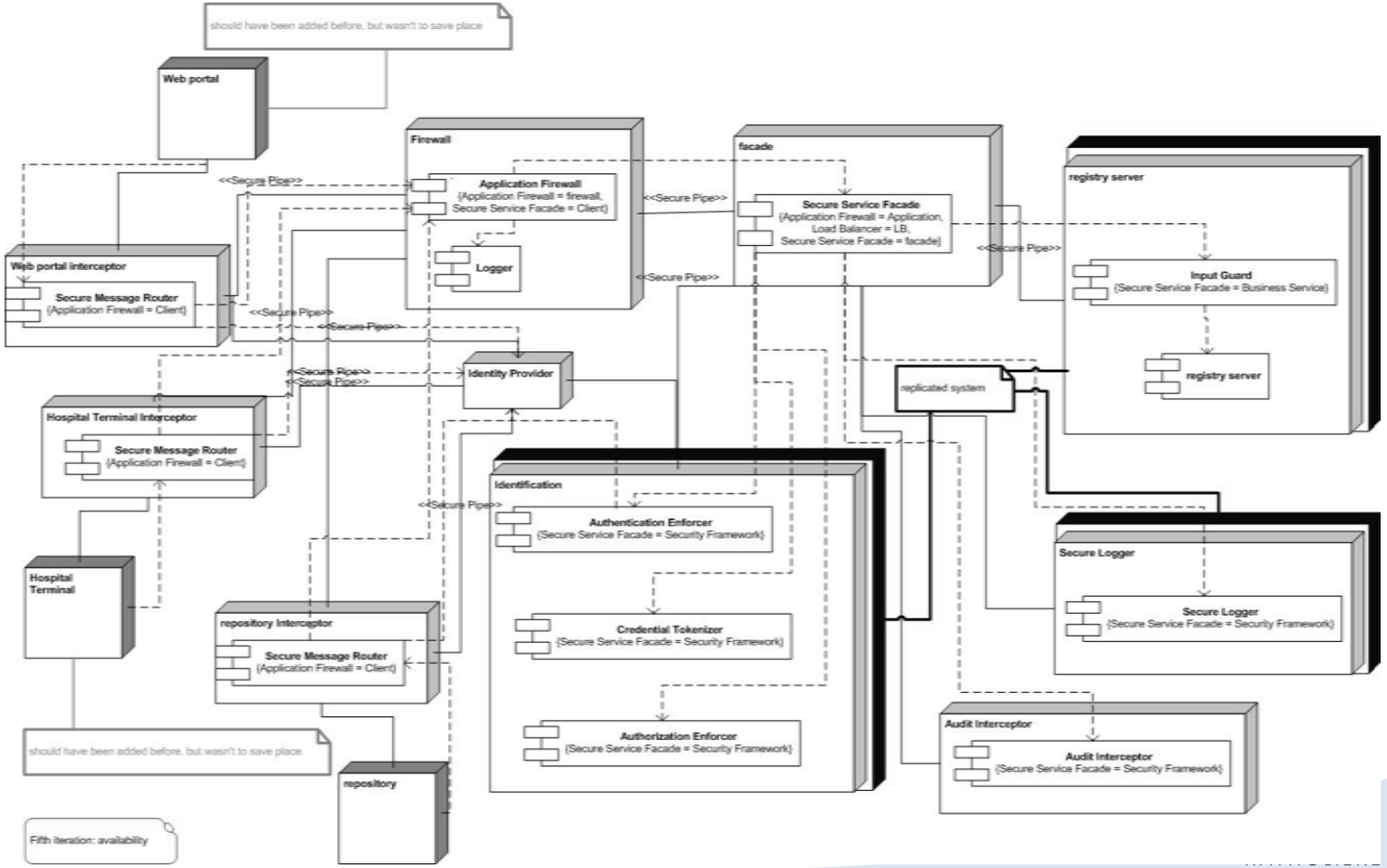
Example
E-health platform



Example
E-health platform



E-Health platform
Final architecture



SECAPPDEV 2008
Security Architectures

Riccardo Scandariato

Wouter Joosen

KATHOLIEKE UNIVERSITEIT
LEUVEN
DistriNet

For further reading

- Software Architecture
  - [SEI] Bass, L. Clements, P. and Kazman, R. 2003 *Software Architecture in Practices*. Addison-Wesley, 2003
  - [Shaw] M. Shaw, and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996
  - [TwinPeaks] B. Nuseibeh, Weaving Together Requirements and Architectures. Computer 34:3, March 2001, pp. 115-117.
- Documenting Software Architecture
  - [Doc] Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., and Little, R. 2002 *Documenting Software Architectures: Views and Beyond*. Pearson Education.
  - [Views] Kruchten, P. "The 4+1 View Model of Architecture," *IEEE Software* 12(6), 1995
  - [Notations] N. Medvidovic, and R.N. Taylor, A Classification and Comparison Framework for Software Architecture Description Languages. Technical Report UCI-ICS-97-02, University of California, Irvine, January 1997

For further reading

- Architecture Evaluation
  - [Survey] Dobrica, L.; Niemela, E., "A survey on software architecture analysis methods," Transactions on Software Engineering , vol.28, no.7, pp. 638-653, Jul 2002
  - [ATAM] P. Clements, R. Kazman, M. Klein “Evaluating Software Architectures”, Addison-Wesley, 2002
- Security patterns
  - [Analysis] Thomas Heyman, Koen Yskout, Riccardo Scandariato, Wouter Joosen, An Analysis of the security patterns landscape, IEEE Workshop on Software Engineering for Secure Systems (SESS), Minneapolis, MN, USA, May 2007
  - [Catalog] Koen Yskout, Thomas Heyman, Riccardo Scandariato, Wouter Joosen, A system of security patterns, K.U. Leuven Technical Report CW469, December 2006
  - [Methodology] Koen Yskout, Thomas Heyman, Riccardo Scandariato, Wouter Joosen, Security patterns: 10 years later, draft paper