CS4135
Software Architectures

Dr. Salim Saay
Associate professor, CSIS, University of Limerick
Funded Investigator at LERO and R@ISE
Salim.saay@ul.ie

2025

Recap

Teaching method

- The hybrid concept is composed of Lectures and project-based teaching.
- LAB Assessment Method
- Projects

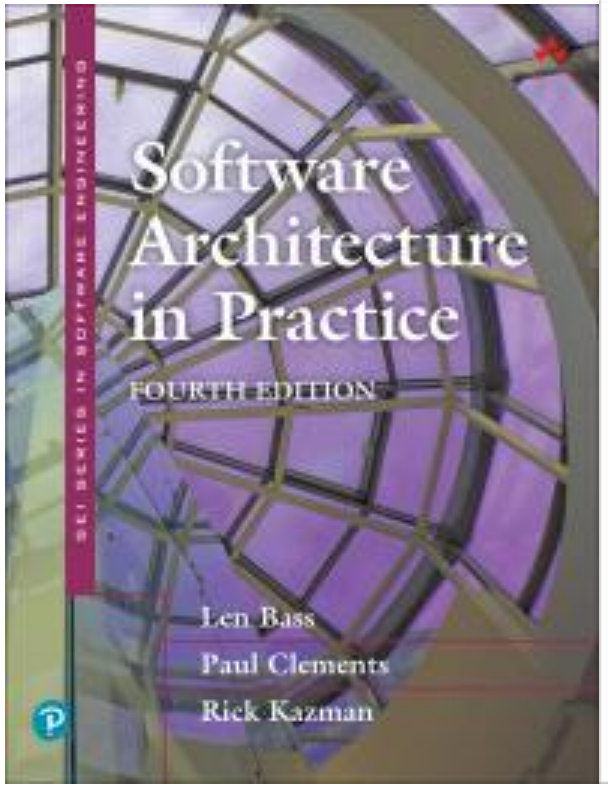
Timetable

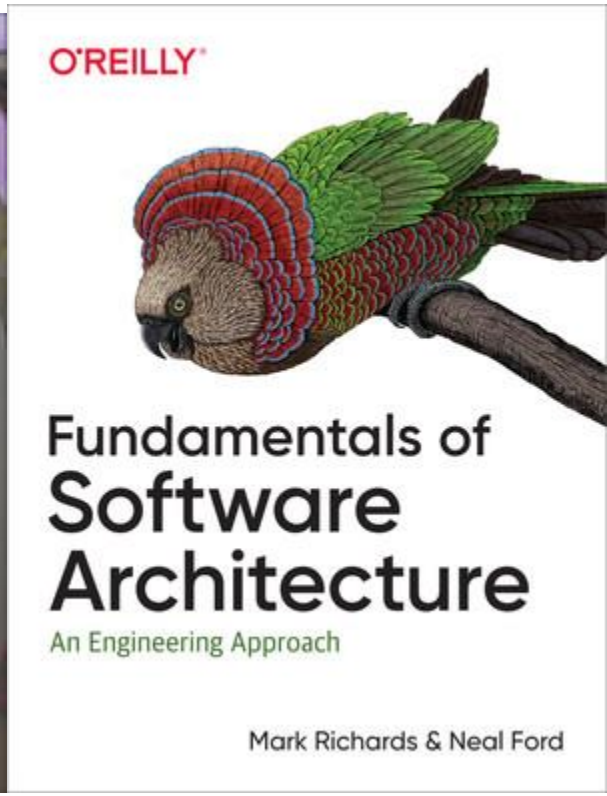
		Deadline
	Project Setup	13/02/2026
	Requirements and UML documentation	27/02/2026
	Strategic architecture development	10/04/2026
	Evaluation and Peer review	24/04/2026
	Deployment and documentation	8/05/2026
	Final Exam	

Contents

- The concept of Software Architecture and the role of the Architect
- Project Design

Part 1: Software Architecture

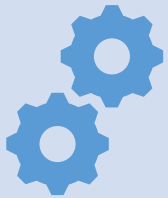




What is Software Architecture?

A high-level **structure** of a system, defining its **components**, their **relationships**, and how they **interact**

to meet **functional** and n**on-functiona**l requirements.



Architecture goes beyond code or technical components

it involves **making strategic decisions** that **set the direction** for the entire system and **align with broader business goals**.

Software Architecture definition

Since 2011, we have had an official definition of software architecture [ISO/IEC Standard 42010] “

- The **fundamental properties of a system** in its environment are embodied in its elements, relationships, and in the principles of its design and evolution”.

To understand the concept and definition of software architecture, need to know:

- **Structure**: what is the form of the interrelated elements
- **Behaviour**: what is the functionality of the interrelated elements?
- **Execution**: how the functionality provided by the interrelated elements is carried out

Software Architecture definition

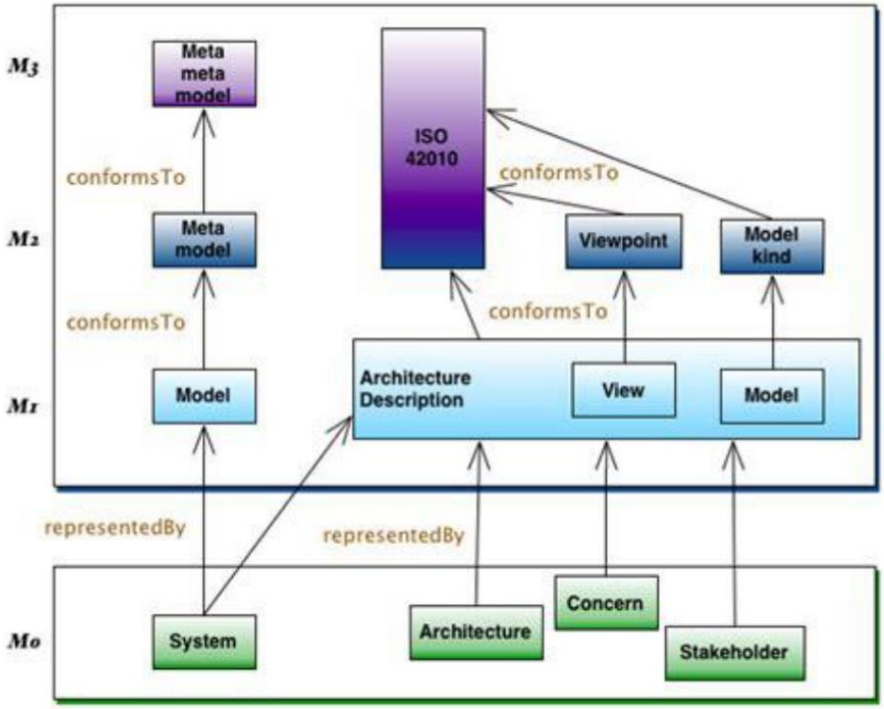
- Three **matters** in an architecture (ISO/IEC Standard 42010 definition)
  - **Elements**
  - **Relationships** between elements
  - **Principles** in the design and evolution of these interrelated elements

Conceptual model of an architecture description proposed by the ISO/IEC/IEEE 42010:2011 standard.

Conceptual model of an architecture

Layers in the Meta-Model Hierarchy

- **M3 - Meta-Meta Model**:
  - Represents the most abstract layer.
  - Defines the structure and rules for creating meta-models.
  - Ensures conformity of all underlying layers.
- **M2 - Meta Model**:
  - Defines the rules and templates for creating models (at M1).
  - Includes constructs such as:
    - **Viewpoint**: Specifies how concerns should be addressed in views.
    - **Model Kind**: Describes the type of model used (e.g., UML, ER diagrams).
  - Conforms to the meta-meta model.



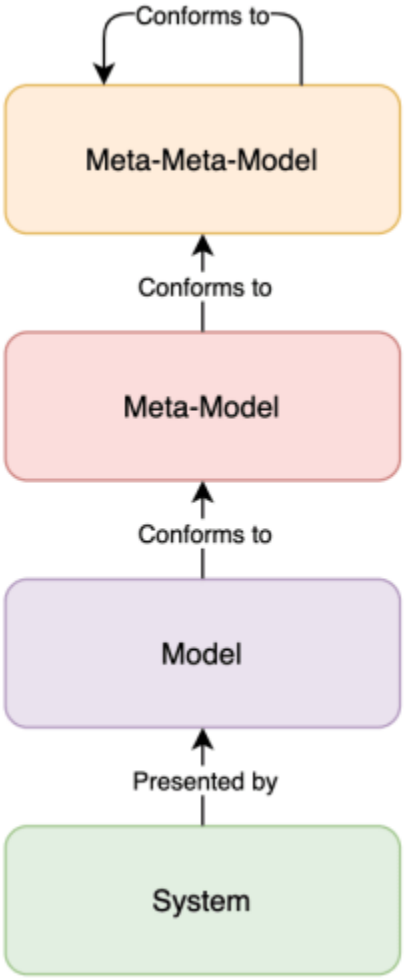
Conceptual model of an architecture

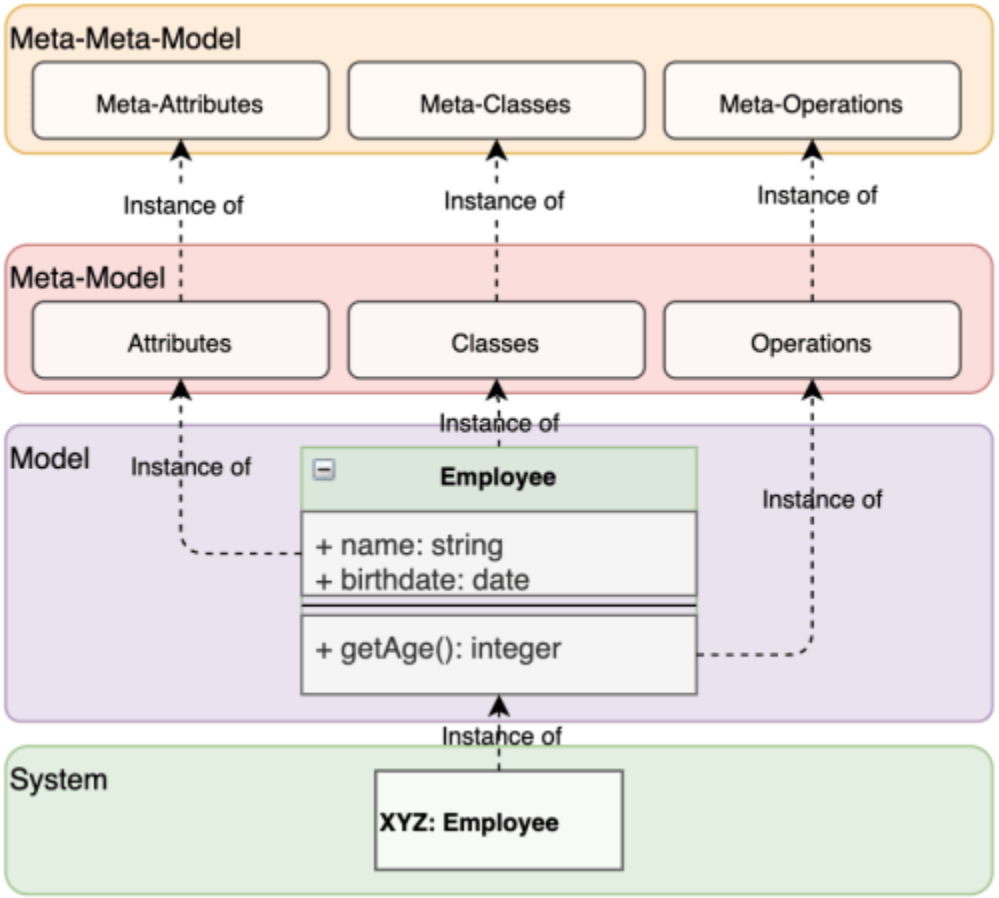
Layers in the Meta-Model Hierarchy

- **M1 - Model Layer**:
  - Represents the **Architecture Description**, which includes:
    - **View**: Depicts specific aspects of the system, guided by a viewpoint.
    - **Model**: Concrete representation of elements in the system (e.g., diagrams, tables).
  - These models conform to the rules defined in the meta-model layer.

- **M0 - System Representation Layer**:
- Represents the real-world elements being modelled, including:
  - **System**: The physical or software system being described.
  - **Architecture**: The system structure
  - **Concern**: Issues or interests related to the architecture.
  - **Stakeholder**: Individuals or groups with a vested interest in the system.
- Models at the M1 layer represent these elements.

Conceptual model of an architecture





Laws of Software Architecture

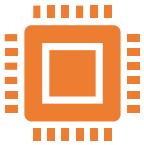
There are two primary **principles** you need to learn

- **Everything in Software Architecture is a Trade-off**:
  - Every architectural decision involves balancing **costs** and **benefits**, such as **performance** vs. **scalability** or **flexibility** vs. **simplicity**.
- **Why is More Important Than How**:
  - Understanding the reasons behind architectural decisions is critical for long-term success.
  - The “how” (implementation details) may change over time, but the “why” must remain aligned with system and business objectives.

Key Concepts of Software Architecture

- **Architecture as a Set of Structures:** categorised into three types:
  - **Module Structures:** Focus on implementation units (e.g., layers, classes, modules) and their static organisation.
  - **Component-and-Connector (C&C) Structures:** Highlight runtime behaviour and interactions (e.g., services, communication mechanisms).
  - **Allocation Structures:** Map software to physical environments (e.g., hardware, file systems, development teams).

Key Concepts of Software Architecture



Architecture as an Abstraction:

Implementation details to focus on **public interfaces** and **relationships** critical for reasoning about system attributes.

Behaviour as Part of Architecture:

An element's behaviour that **impacts other** components or the overall system is integral to the architecture beyond just box-and-line diagrams.

Every System Has an Architecture:

Whether documented or not, every software system inherently has an architecture.

However, its usefulness and clarity depend on proper documentation and dissemination.

Key concept of Software Architecture

Software architecture is an **essential activity** for the development of software systems

- Enabling reasoning about system properties
- **Early** in the development lifecycle.

The issue is on how to organise a system to simultaneously: **Trade-off**

- provide the required **functional** services,
- guarantee the required **quality of service**.

Architecture Characteristics

Software architecture is **concerned** with

The n**on-functional** aspects of a system, such as

- Scalability,
- Reliability,
- Maintainability,
- Security,

The **functional** requirements of the system.

- The structure of the system,
- The technologies to be used,
- The design patterns to be employed, and
- The trade-offs between various competing factors.

Qualities of a "Good" Architecture

- **Fit for Purpose:**
  - A good architecture is context-specific and aligned with the system's business and technical goals.
- **Process Recommendations:**
  - Lead by a small team for conceptual integrity.
  - Base architecture on prioritized quality attributes.
  - Use architectural views to support stakeholder communication.
- **Structural Recommendations:**
  - Apply principles of information hiding and separation of concerns.
  - Use standard patterns and tactics to achieve desired quality attributes.
  - Minimize dependencies on specific tools or commercial products.

Why Is Software Architecture Important?

- **Driving Architecture Characteristics:**
  - Architecture significantly influences whether a system can meet its required quality attributes (e.g., performance, security, scalability).
  - It shapes time-based behavior, resource usage, coupling, and dependency management to achieve these attributes.
- **Justifying and Managing Change:**
  - A well-designed architecture isolates potential changes within specific components, enabling adaptability.
  - Helps stakeholders assess the impact and feasibility of modifications.

Why Is Software Architecture Important?

- **Predicting System Qualities:**
  - By analyzing architectural structures, one can predict system qualities early in the lifecycle, such as modifiability, scalability, or reliability.
  - This early analysis reduces risks and guides design decisions.
- **Enhancing Stakeholder Communication:**
  - Architecture is a common language for diverse stakeholders, such as developers, testers, and business owners.
  - Documented architecture provides clarity and shared understanding, fostering collaboration.
- **Making and Capturing Fundamental Design Decisions:**
  - Architecture embodies the earliest and most critical design decisions, which are complex and expensive to change later.
  - These decisions form the foundation for all subsequent development work.

Why Is Software Architecture Important?

- **Defining Constraints on Implementation:**
  - Architectural choices constrain how systems are implemented, ensuring consistency and adherence to quality goals.
  - Defines “rules of engagement” for development, such as design standards and coding conventions.
- **Influencing Organizational Structure:**
  - According to Conway’s Law, the architecture often mirrors the organizational structure that created it.
  - A clear architecture supports efficient team coordination and task division.
- **Enabling System Evolution:**
  - Architecture supports prototyping by offering a structure to build and test incremental system functionality.
  - Facilitates iterative development, especially in Agile or exploratory projects.

Why Is Software Architecture Important?

Supporting Cost and Schedule Estimation:

- Provides a basis for estimating development effort, resource allocation, and timelines.
- Helps project managers plan and manage risks associated with large-scale software development.

Foundation for Reusable Models:

- Architectures can be reused across product lines or projects, enabling economies of scale.
- Serves as a blueprint for building similar systems efficiently.

Focus on Component Composition:

- Encourages viewing the system as a set of interacting components rather than just a collection of code.
- Shifts focus from creating components to effectively combining them, improving modularity and maintainability.

Why Is Software Architecture Important?

- **Constraining Design Alternatives:**
  - Architecture narrows design choices, channeling creativity toward solving specific problems.
  - Reduces complexity by limiting unnecessary variability in implementation.
- **Basis for Training:**
  - Architectural documentation provides new team members with a structured understanding of the system.
  - Accelerates onboarding by offering a clear overview of system structure and goals.

Architectural Thinking

- **Architectural thinking** is a mindset that emphasizes strategic decision-making, forward-thinking, and balancing trade-offs.
- Architects must look **beyond immediate technical concerns** and consider the long-term impact of their choices on the system's evolution and alignment with business goals.
- **Communication** is another critical aspect of architectural thinking. Architects must explain and justify their decisions to technical teams and business stakeholders, ensuring everyone understands the rationale behind the choices and how they align with broader business goals.

Architectural Thinking

Vital elements of architectural thinking include:

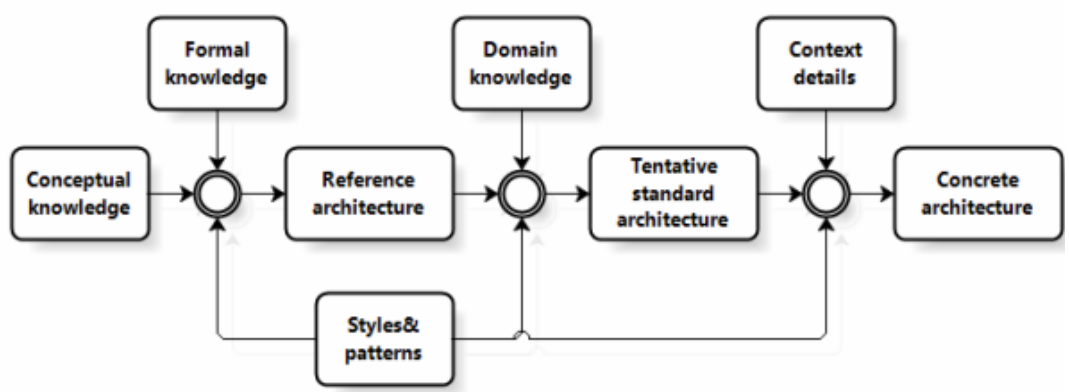
- **Managing complexity**: Simplifying the system's design where possible while ensuring it can handle future requirements.
- **Future-proofing decisions**: Architects must make decisions anticipating the system's need to evolve, ensuring flexibility and adaptability to changing technologies and business needs.
- **Navigating trade-offs**: Every architectural decision comes with costs and benefits, and architects must weigh these trade-offs. For example, prioritizing scalability may introduce complexity, while optimizing for simplicity might limit flexibility.

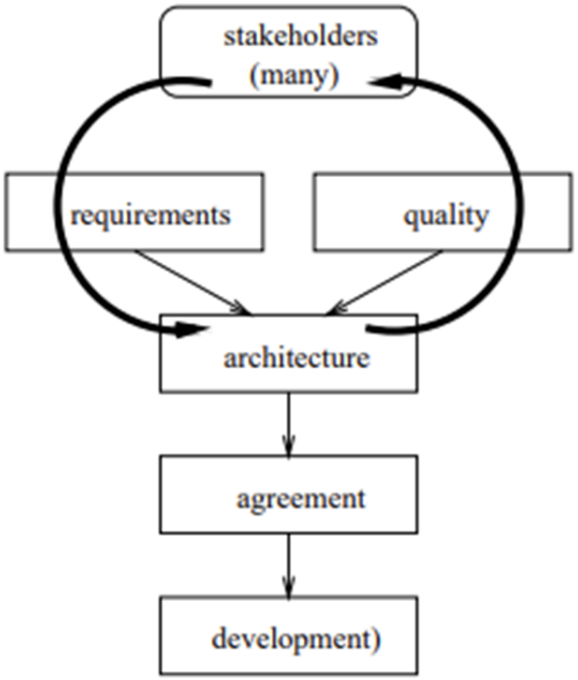
Software architect

- When you work as an **architect**, you need to consider different components of the system and need to know how they work together.
  - Software architect needs to focus on the **big picture and** the whole system.
  - **While software developers** need to understand deeply **one or two components**,
  - Architects need to understand **many components** but at a high level
  - The responsibility of an architect is beyond the design and architect but is also responsible for the **performance** of the system.
  - Architect is an **evaluator** and needs to provide continuous feedback.
  - Architects need to **continuously learn new things**
  - Architects need to **review the plan** of software development and guide developers.

Software architect

Developing software architecture requires broader knowledge

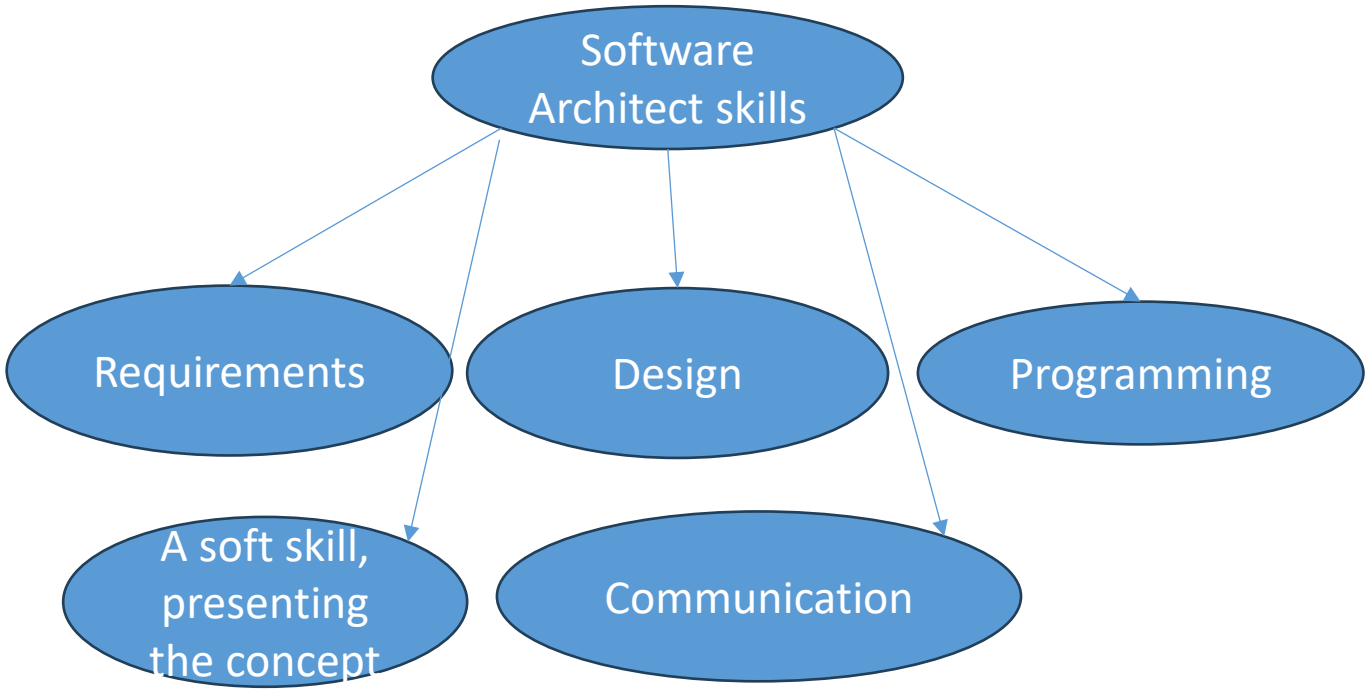




Software architect

Software architecture is not just putting together existing packages as if they were building blocks.

In programming, you learn simplicity, maintainability and testability



Software architect

Traditional architects

Modern architects

Feature	Traditional Software Architect	Modern Software Architect
Decision-Making	Centralized, top-down	Collaborative, agile-driven
Development Approach	Often Waterfall	Agile, DevOps integration
Documentation	Extensive, rigid documentation	Lightweight, iterative documentation
Technology Adoption	Focus on stable, proven tech	Embraces emerging trends like microservices, cloud, and AI
Flexibility	Less adaptable to change	More iterative and flexible

Architecture vs Design

Another critical concept to understand is the boundary where architecture ends and design begins:

- **Architecture** involves making high-level, strategic decisions that determine the long-term direction of the system and are more difficult to change. Essentially, architects define the components (the system's fundamental building blocks), their scope of responsibilities, and the relationships between them.
- **Design** focuses on tactical decisions, addressing the finer details of implementation that are more flexible and can evolve as the project progresses. Essentially, design answers how a specific component or the relationship between components should be implemented.

Part 2: Project Design

Identification of the project

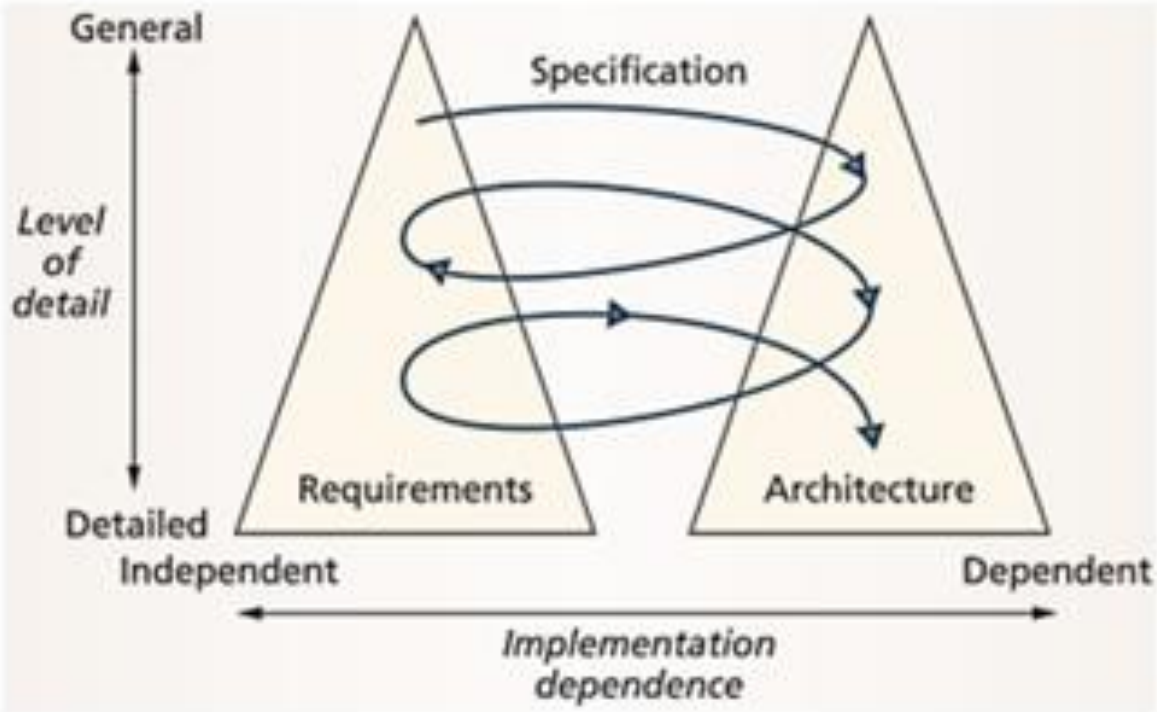
- Is this a new application?
  - Does the application have enough description?
  - Are the requirements listed?
- Is this an existing application?
  - Does it need extension?
  - Evolve
  - Required structure change?
  - ….

Definition of Architectural Requirements

The definition of architectural requirements aims to meet the following goals:

- Describe a **necessary change** to components in an architecture
- Include the **reasoning** or motivations behind the change.
- Outline the **available options** for future architectures that address all concerns.
- Explain the **benefits**, value, risks, costs, opportunities, constraints, and future options associated with each alternative.
- Outline any **alternative routes** to close the gaps and get from the current to the target architecture.

Relationship between Software Requirements and Architectural Activities in Agile Environments



The twin peaks model showing the interplay of requirements and architecture.

Identification of Software Architecture Styles, alternatives

- Alternative architecture solutions may be proposed and analyzed to identify an optimal solution for the product
- The identification of software architecture styles aims to precise the associated elements, forms, and rationales:
  - **Elements**: There are three classes of software elements, namely processing elements, data elements, and connecting elements.
  - **Forms**: The architectural form consists of weighted properties and relationships. Properties define the constraints on the software elements to the degree desired by the architect.
  - **Rational**: The rationale explains the different architectural decisions and choices; for example, why a particular architectural style or element or form was chosen.

Software Architecture patterns

Categories of Software Architecture Patterns

Application Landscape	Application Structure	User Interface
▪ Monolith ▪ N-Tier ▪ **Services-oriented** ▪ **Microservices** ▪ Serverless ▪ Peer-to-Peer	▪ Layered ▪ Microkernel ▪ **Query based** ▪ **Event driven** ▪ Plugin	▪ Model-view-control ▪ Model-view-presenter ▪ Model-view-view-model

Some commonly used software architecture patterns

Software architectural patterns in practice: an empirical study

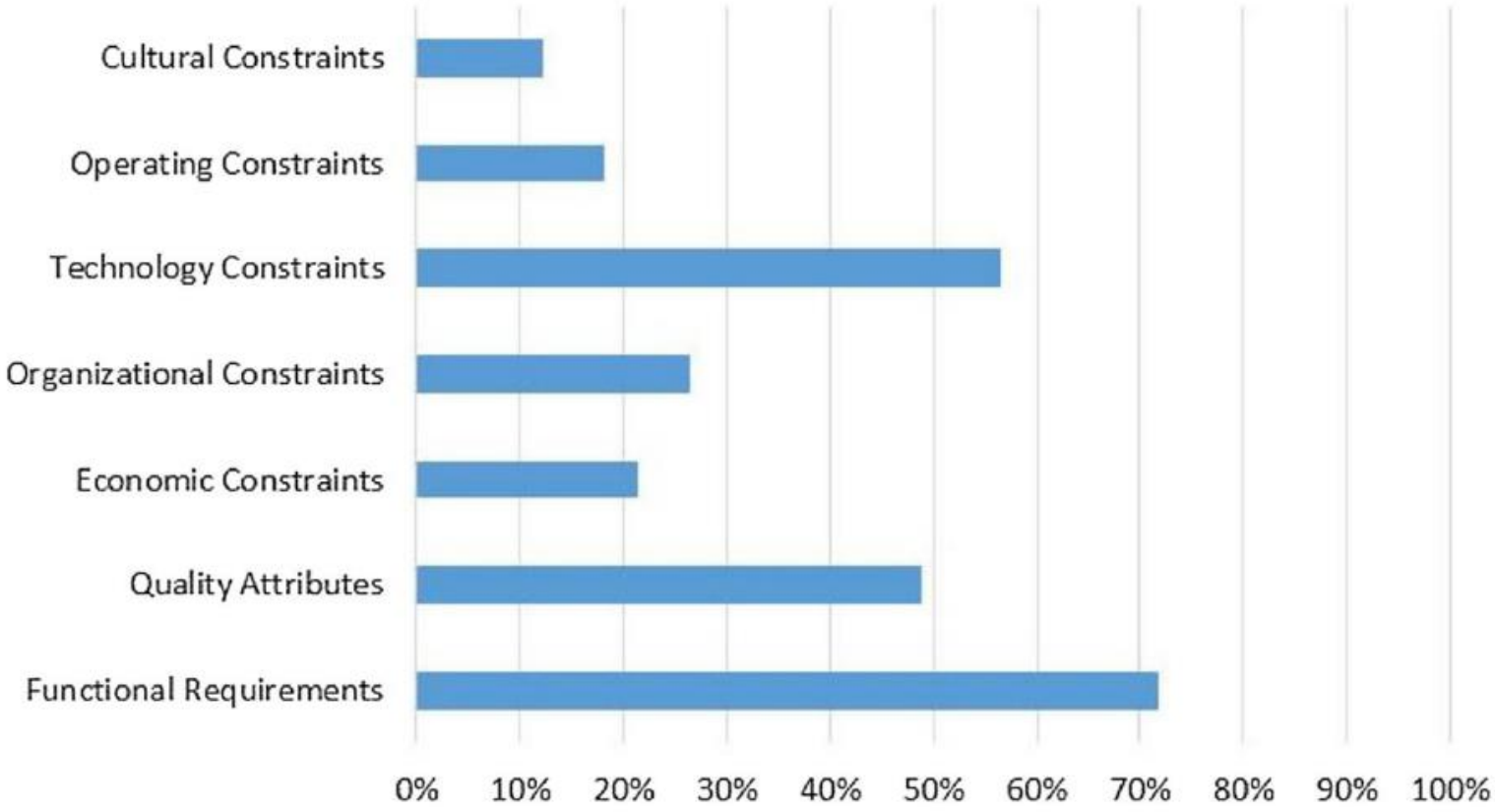
Table 1 Relevant quality requirements per project domains: a value in the table presents% of projects from corresponding domain while concerned about corresponding quality as relevant

	Aerospace (%)	Defense (%)	Education (%)	Finance (%)	Gaming (%)	Government (%)	HR (%)	Marketing (%)	Medical systems (%)	Sales (%)	Utilities (%)
Quality not considered	0	0	22	11	19	14	8	6	3	11	9
Availability	63	25	39	56	43	50	42	38	13	47	59
Interoperability	25	25	17	19	33	32	33	19	40	15	18
Modifiability	100	100	52	53	62	55	83	63	60	56	59
Performance	50	50	39	69	67	64	75	56	40	49	55
Security	50	50	30	44	29	23	50	44	20	29	23
Testability	63	25	22	39	19	27	42	25	20	25	32
Usability	50	50	52	53	43	45	67	56	67	38	36
Other	0	0	0	2	0	0	0	0	13	0	4

Kassab, M., Mazzara, M., Lee, J., & Succi, G. (2018). Software architectural patterns in practice: an empirical study. *Innovations in Systems and Software Engineering*, *14*, 263-271.

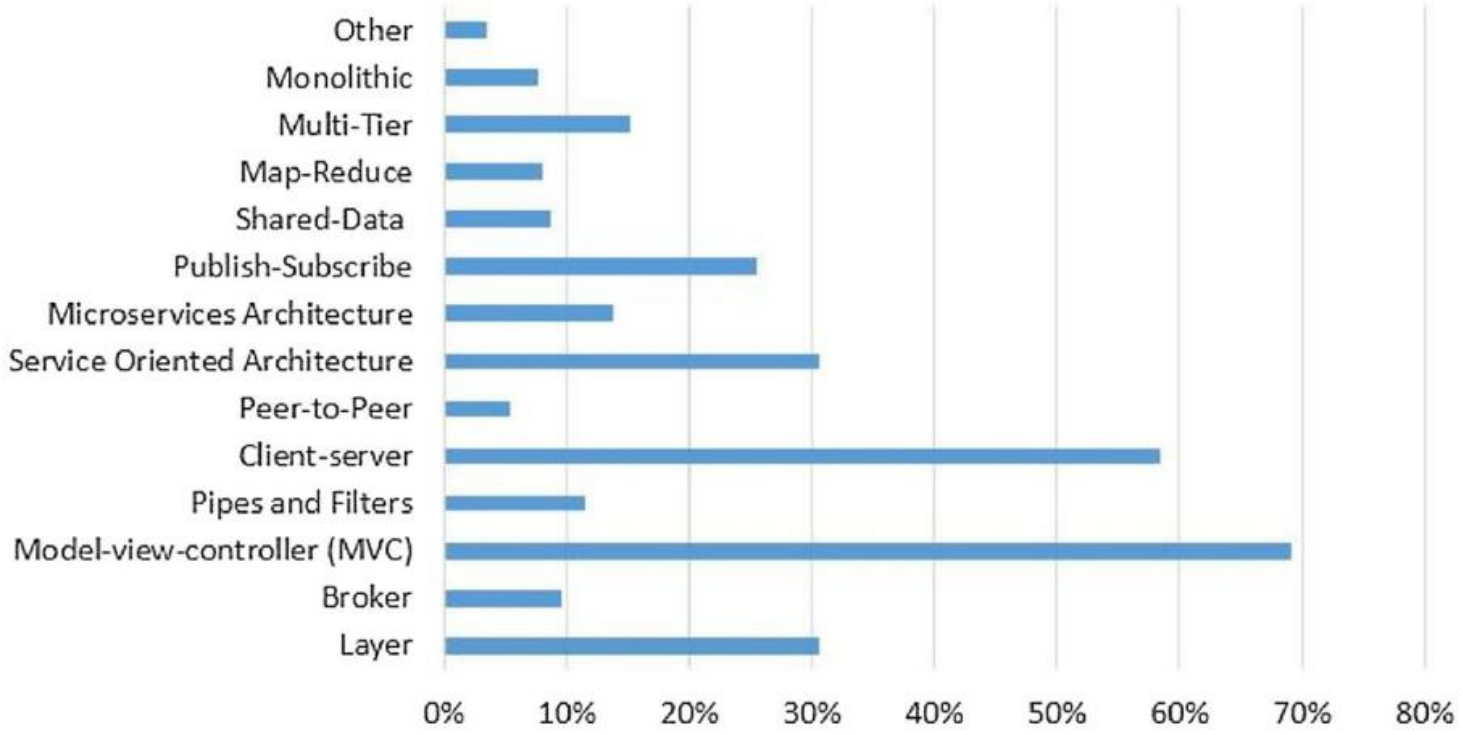
Software architectural patterns in practice: an empirical study

Which **criteria** were the most important in choosing the selected architectural styles (patterns) for the project?



Software architectural patterns in practice: an empirical study.

- **A survey in 2018** shows the following. Survey completed at a conference.
- participants from 39 countries, **126 of 809 participants** completed the survey



https://doi.org/10.1007/s11334-018-0319-4

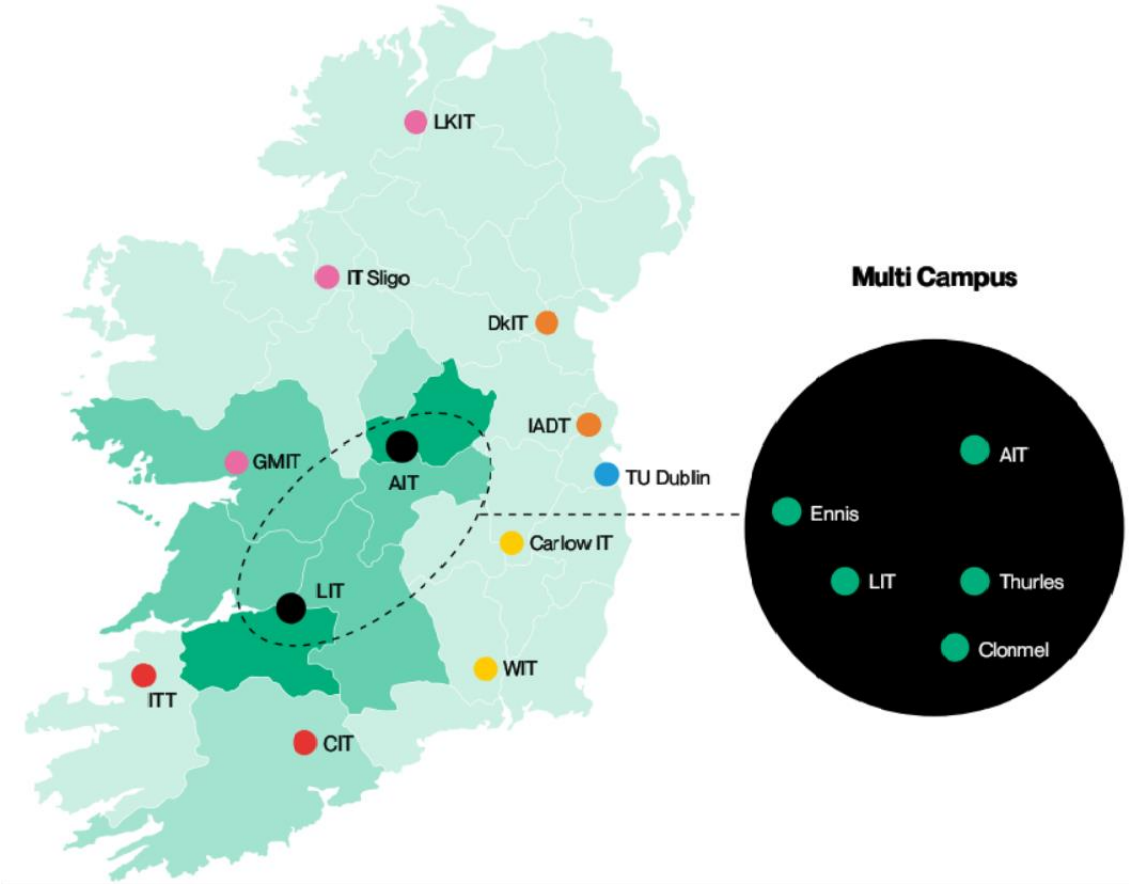
Software Architecture patterns

- Here are some details of commonly used software architecture patterns:
- **Model-View-Controller** (MVC) - pattern separates an application into three interconnected parts:
  - the **Model (Data, logic)**, the **View (interface)**, and the **Controller (**handle user input /output).
- **Microservices** - pattern **breaks down** a large application into a collection of **smaller**, **independent** services
  - Services communicate with each other through defined **APIs**.
  - Each microservice is responsible for a specific business capability
  - Each server can be developed, deployed, and scaled independently.
- **Layered Architecture** - pattern separates an application into distinct **layers of functionality**
  - Each layer is responsible for a specific set of tasks.
  - The layers are arranged in a hierarchical way
  - Each layer communicates with the layer directly **above** or **below** it.

Software Architecture patterns

- **Event-Driven Architecture** - Pattern is based on decoupling an application's components
  - allowing them to communicate through events.
  - When **an event occurs**, it triggers one or more actions handled by different system components.
- **Domain-Driven Design (DDD)** - Pattern focuses on **modelling** an application around the core business domain.
  - The domain model is a representation of the key concepts, rules, and relationships that exist within the business domain.
  - The application is then built around this domain model.
  - Design the architecture exactly how a customer need
- These are just a few examples of software architecture patterns. There are many other patterns that are commonly used, each with its own strengths and weaknesses.

Example of a project in real cases



Saay, TM Salim, and Tiziana Margaria. "XMDD as Key Enabling Technology for Integration and Organizational Collaboration: Application to E-Learning Based on NRENs." Advances in Science, Technology and Engineering Systems Journal 6, no. 3 (2021): 213-230.

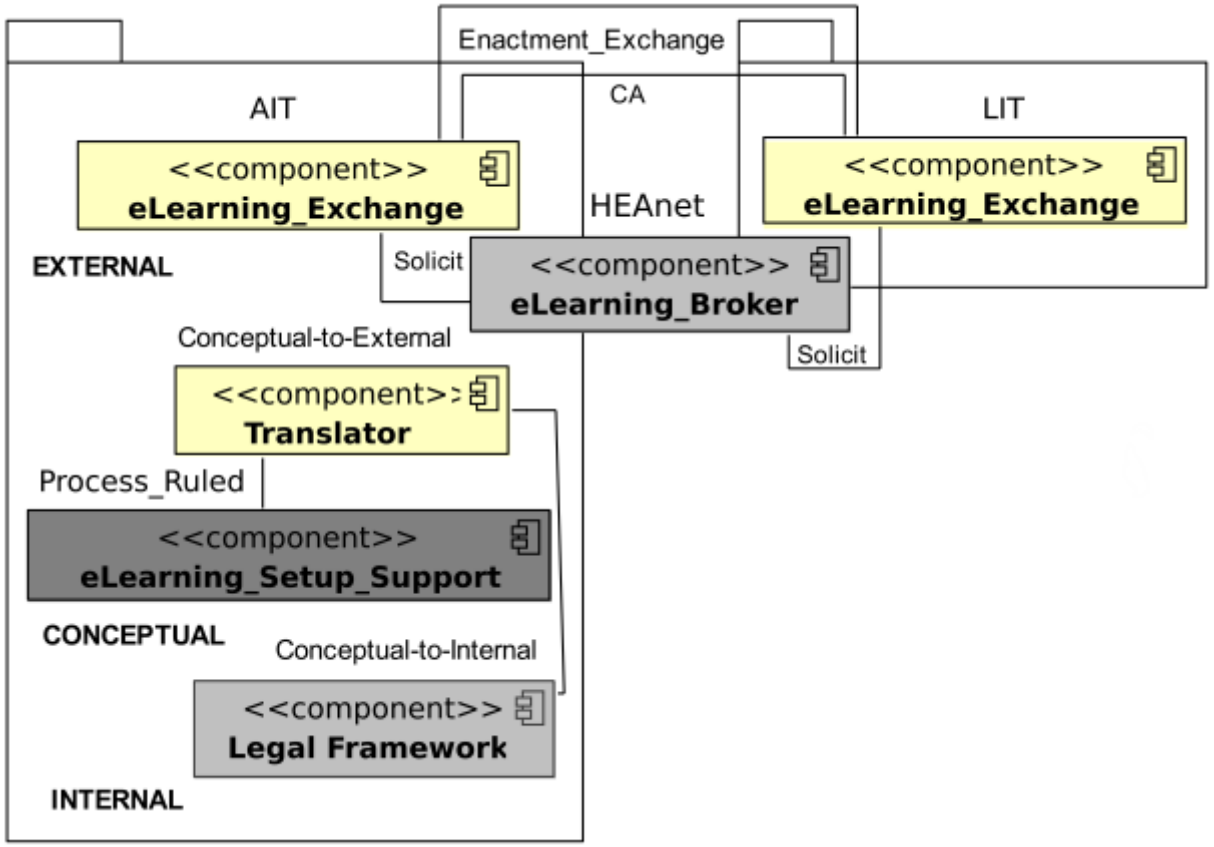
Example of a project in real cases

Table 1: Analysis of Architecture Patterns based on Quality Attributes [44, 42, 43, 41, 45, 46, 47, 48, 49]

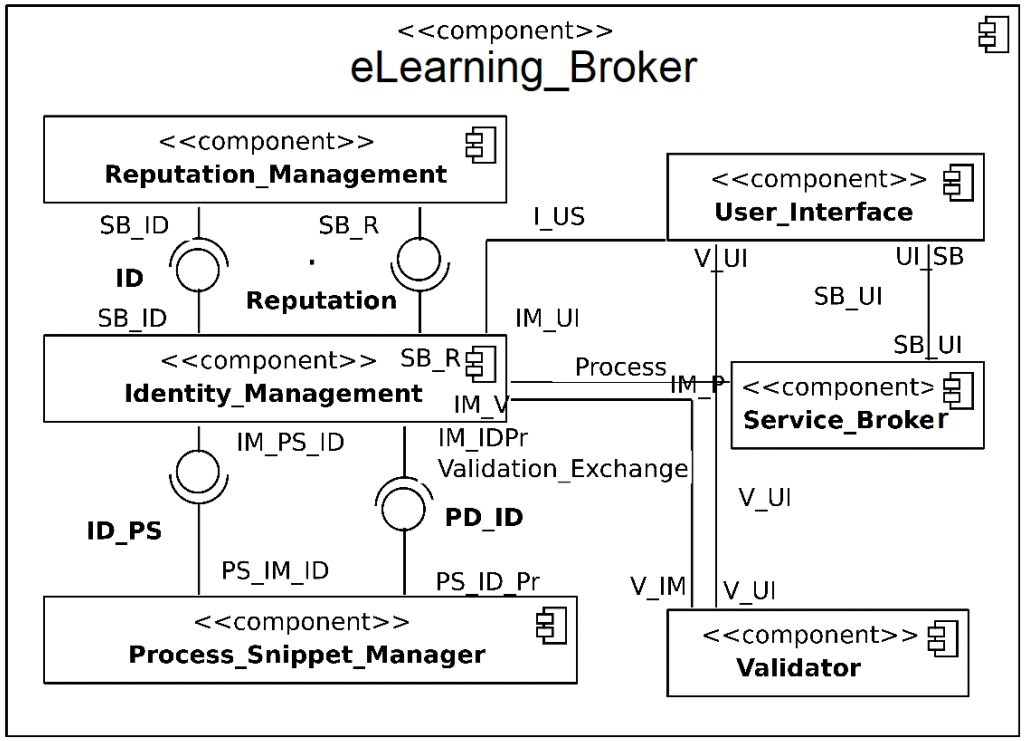
Architecture Pattern	Agility	Deployment	Testability	Performance	Scalability
Layered Architecture	*	*	***	*	*
Event-Driven Architecture	***	***	*	***	***
Plug-in Architecture	***	***	***	***	*
Microservice Architecture	***	***	***	*	***
E-Learning Broker Architecture	**	***	***	***	***

Saay, TM Salim, and Tiziana Margaria. "XMDD as Key Enabling Technology for Integration and Organizational Collaboration: Application to E-Learning Based on NRENs." Advances in Science, Technology and Engineering Systems Journal 6, no. 3 (2021): 213-230.

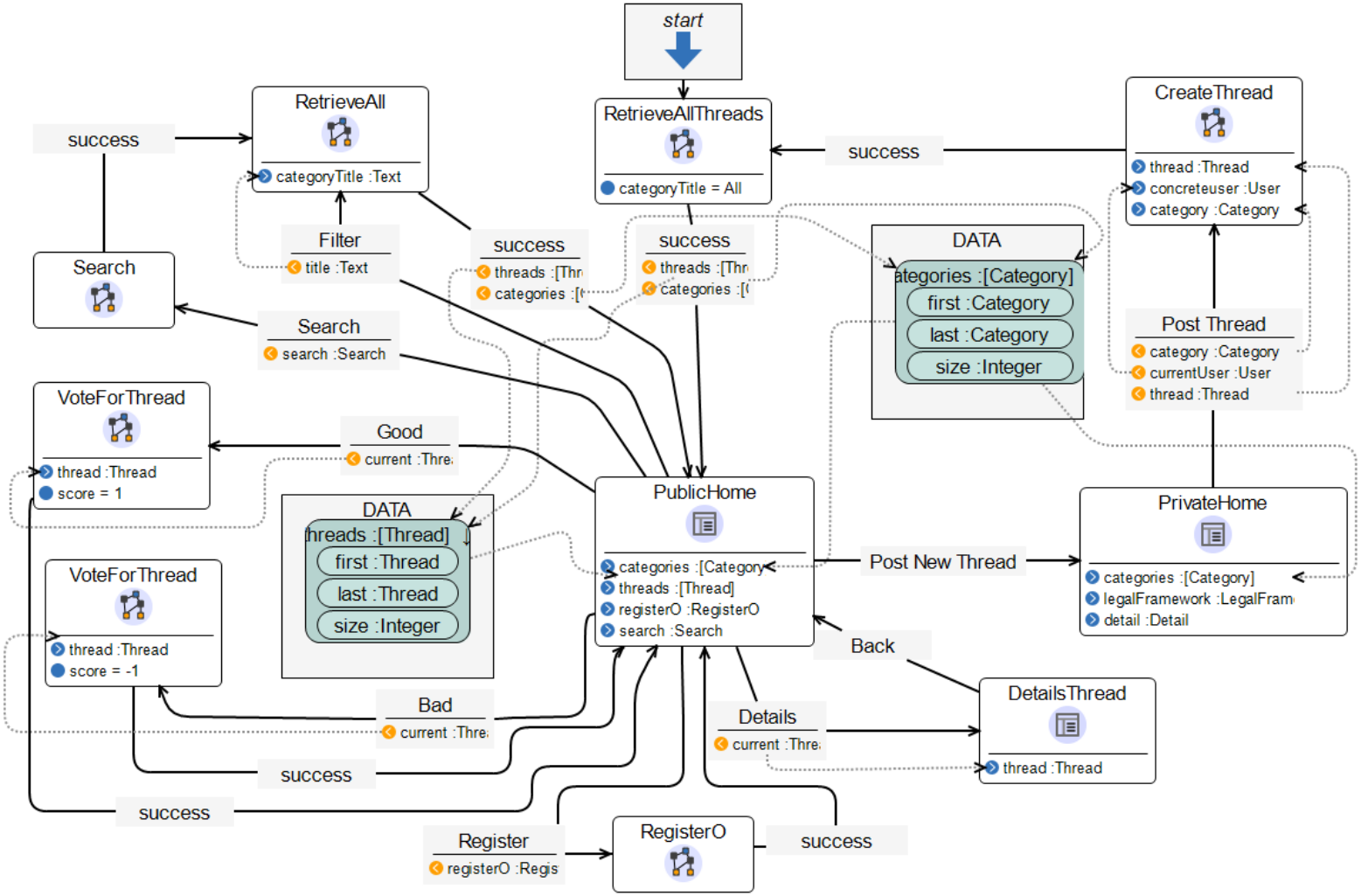
Example of a project in real cases



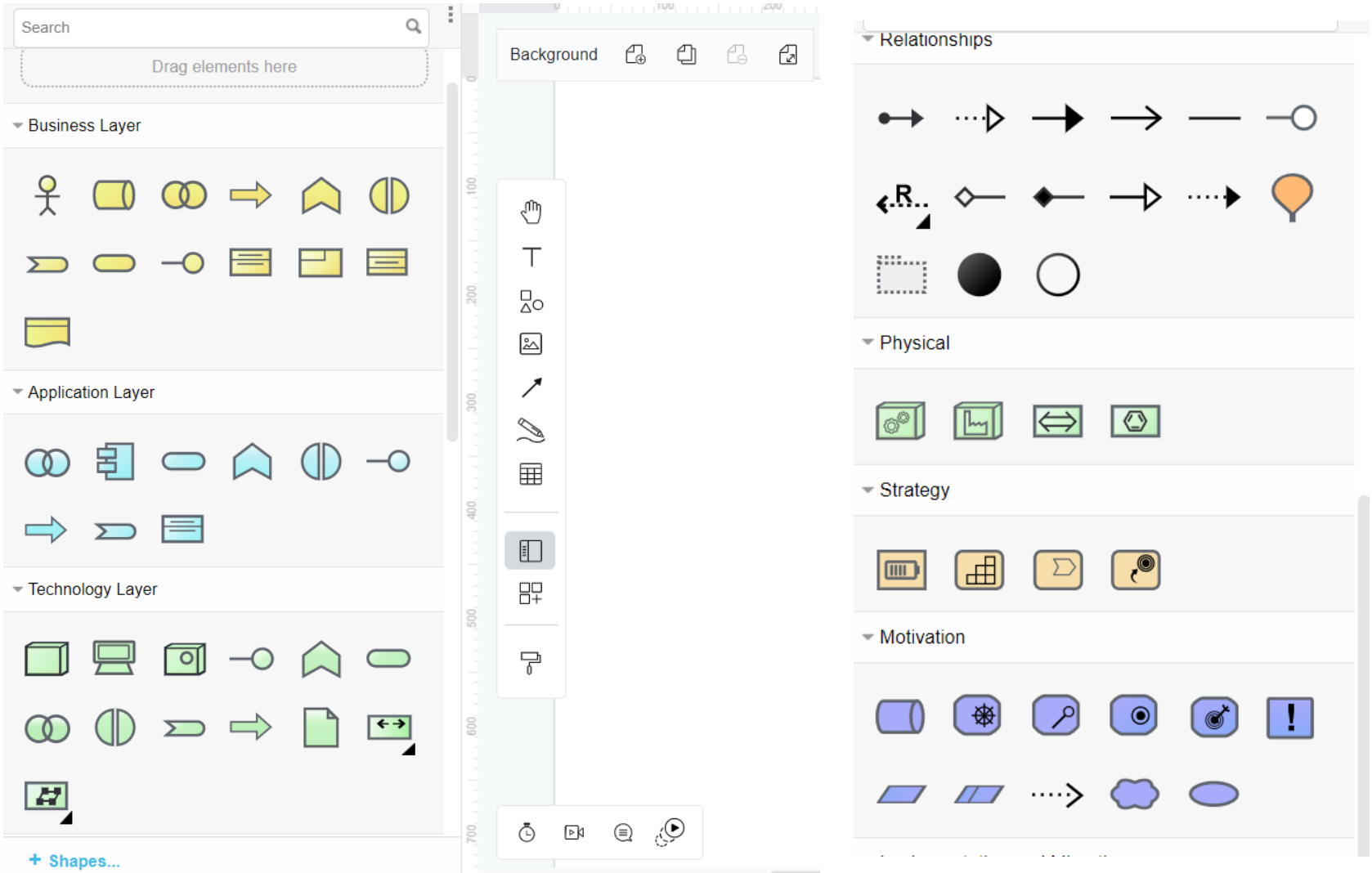
Decomposition



Prototype


start
RetrieveAll
categoryTitle :Text
RetrieveAllThreads
categoryTitle = All
CreateThread
thread :Thread
concreteuser :User
category :Category
success
Filter
title :Text
Search
search :Search
DATA
categories :[Category]
first :Category
last :Category
size :Integer
Post Thread
category :Category
currentUser :User
thread :Thread
VoteForThread
thread :Thread
score = 1
Good
current :Thre
PublicHome
categories :[Category]
threads :[Thread]
registerO :RegisterO
search :Search
PrivateHome
categories :[Category]
legalFramework :LegalFram
detail :Detail
Post New Thread
Back
Bad
current :Thre
Details
DetailsThread
thread :Thread
Register
registerO :Regis
RegisterO

Practice: [Untitled | Visual Paradigm Online (visual-paradigm.com)]



Language for Modeling Software Architecture

- UML (Unified Modeling Language) Visual Paradigm Online (visual-paradigm.com)
- SysML (Systems Modeling Language), a specialised type of UML that has more details.
- ArchiMate Untitled | Visual Paradigm Online (visual-paradigm.com)
- BPMN (Business Process Model and Notation) https://www.bpmn.org/
- ER (Entity-Relationship) Untitled | Visual Paradigm Online (visual-paradigm.com)
- SDL (Specification and Description Language) Untitled | Visual Paradigm Online (visual-paradigm.com)
- Petri Nets http://petrinet.org/

Learning ArchiMate: https://www.youtube.com/watch?v=CJ9oWiUMvCU

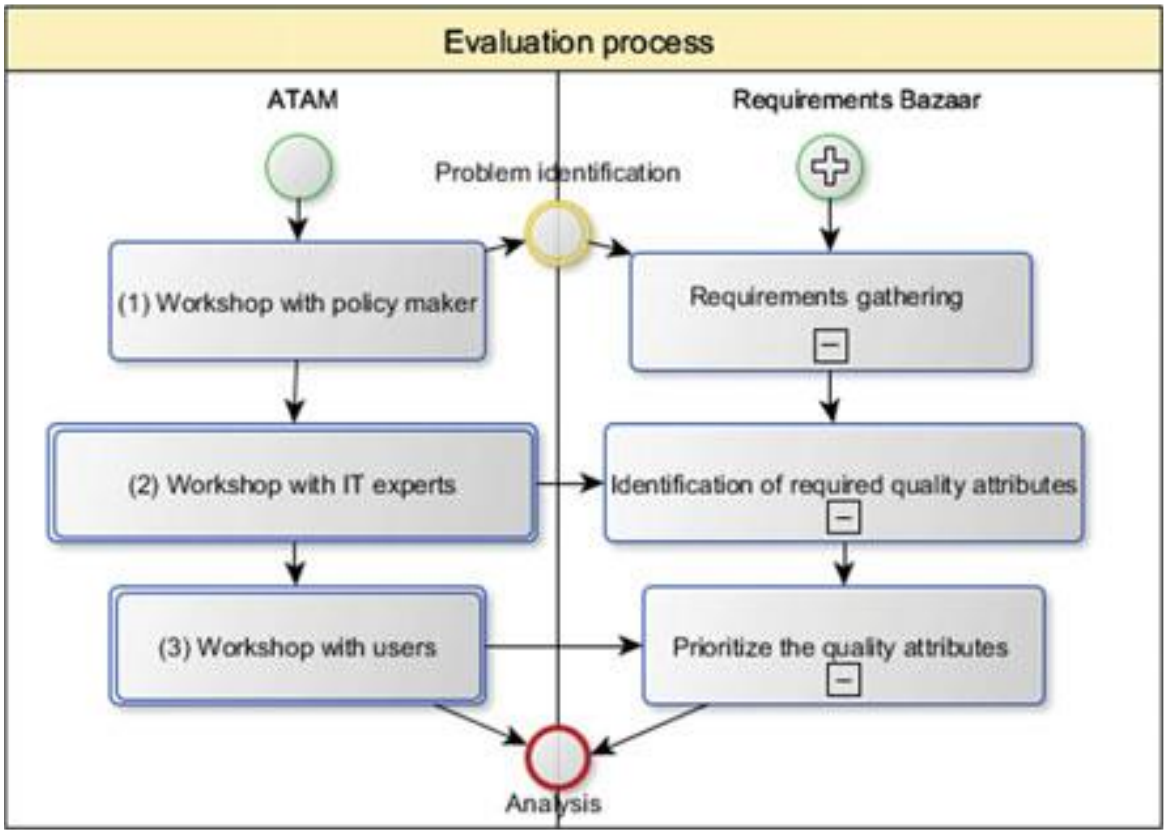
Conclusion

Agile and adapting agile environment

Requirements gathering

Next- Evaluation

Software Architecture Evaluation



Saay, Salim, and Alex Norta. "Designing a scalable socio-technical method for evaluating large e-governance systems." In Advanced Computational and Communication Paradigms: Proceedings of International Conference on ICACCP 2017, Volume 1, pp. 571-580. Springer Singapore, 2018.