Introduction to Software Architecture Evaluation
Contents

• Introduction to Software Architecture Evaluation
  – Software Architectures
  – Architectural Patterns
  – Quality in Software Architectures
  – Architecture Evaluation

• Architecture Evaluation Methods
  – Architectural Tradeoff Analysis Method (ATAM)
  – Quality-Driven Architectural Improvement (QuADAI)
Architectures

- Architecture is roughly the prudent partitioning of a whole into parts, with specific relations among these parts.
  - This partitioning is what allows groups of people—often separated by organizational, geographical, and even time-zone boundaries—to work cooperatively and productively together to solve a much larger problem than any of them could solve individually.
Software Architecture Definition

• 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.

  [Bass, Clements, y Kazman, 1998]

• By “externally visible” properties, we are referring to those assumptions other components can make of a component, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.
Software Architecture

• Every software system intrinsically has an architecture even it is unknown or undocumented.

• A **software architecture** defines which role components play and how they interact with each other in a particular software application.

• When the software architecture is explicitly documented it provides valuable information about the software as *it is used as knowledge to support decisions* during the software development.

• Software systems are composed of many structures (commonly called views). No single view can appropriately represent the architecture.
  
  – A single view shows the architecture taking into account one specific aspect and can be used to analyze one specific problem.
Software architectures are usually described using different views:

- Modular
- Component & Connector
- Deployment
In the last years, several formal notations and languages for the description of software architectures have been proposed:

- **UML**: a software modeling standard than can also be used for the *description of software architectures*, even though its abstractions are not the best way for representing software architectures.

- **SysML**: although it is a system modeling language (hardware & software), SysML provides the abstractions required for specifying the different views of software architectures.

- **AADL**: is an architectural specification language that provides a textual and a graphical syntax for the description of software architectures.

  - It allows not only the specification of architectural views but also the modeling and analysis of other aspects related to the quality of software architectures (e.g., failure or latency time analysis).
Architectural Patterns 1/3

- **Architectural patterns**: describe *common solutions to recurrent problems* in the software architecture context.
  - Are a means for improving certain quality attributes (e.g., flexibility, modularity, reliability).
  - Specify how the system will deal with one aspect of its functionality, impacting directly on the quality attributes.

- There are examples for improving the maintainability of a system as the *layers pattern*.
Architectural Patterns 2/3

- Architectural patterns are usually described using templates
- The template for the description of an architectural pattern typically contains:
  - **Context:** The context in which the pattern is commonly applied.
  - **Problem:** Architectural problem that it solves.
  - **Solution** (pattern structure): How it changes the architecture
    - In some cases these changes are specified by showing the “before” and “after”
  - **Consequences:**
    - **Benefits:** What will be improved on the architecture after the application of the pattern
    - **Drawbacks:** Which problems can we have if we apply the pattern
Architectural Patterns 3/3

• **Context**
  – Applications that need to expose part of their functionality or consume services from other applications or have different abstraction levels

• **Problem**
  – In some systems we require portability to other platforms or exporting services from the application’s core to other client applications
  – In systems where the abstract concepts from a given domain have to be implemented in terms of a more specific domain.

• **Pattern Structure**

• **Consequences**
  – Improves the modifiability and adaptability
  – In some cases, a severe problem can occur when the behavior of a layer changes.
  – Layered architecture with too few layers does not exploit this pattern's potential for reusability, changeability and portability; too many layers introduce unnecessary complexity
Quality in Software Architecture 1/2

- Software architecture is a mechanism that allows us to ensure (or achieve) certain quality attributes such as performance, reliability or security.
- When evaluating an architecture, if we need to improve performance, we should:
  - Exploit the parallelism, decomposing the work in different processes that can run in parallel and cooperate among them
  - Improve the communication between processes, increasing the network bandwidth and the frequency of data access.
  - Be able to estimate latency and performance.
  - Identify bottlenecks.
- The quality attribute requirements are the main drives for architectural design.
- The architecture is also a tool for predicting the quality of the system under development.
The way in which a system achieves its quality attribute requirements depends on the architectural design decisions.

The strategies for attaining the quality attributes levels are mainly architectural, but it is important to understand that architecture alone cannot guaranty the functionality or quality:

- Wrong design decision or incorrect implementations can always undermine a good architectural design.
- The decisions taken all along the lifecycle (from the design to the codification and implementation) will affect to the system quality. Therefore Quality is not only function of the architectural design.
- For ensuring the quality of a system, a good architecture is required but it may not be enough.
Since the architecture allows the attainment of certain quality attributes (meaning the success or failure of a project), its evaluation in early stages is a crucial task in a software development project.

It is possible to verify if the architectural decisions are appropriate in early stages without waiting for the system to be developed and deployed.

– We can predict if a system will have the required quality attributes.

The goal is to determine the degree in which a software architecture or an architectural style satisfy the quality requirements.

Architectural evaluation has saved important amount of money when detecting that the system under development could not achieve the quality requirements which was supposed to in early stages of the development.
• We should be sure that the selected architecture for a project fits in with the project needs.
  – If the design decision determine the quality attributes of the system we should evaluate the decisions with regard to these quality attributes.

• The earlier we find a problem, the less it costs to fix it.
  – If we predict the impact a design decision has, and we choose the best option for our project, we will reduce the budget and obtain the software architecture we need.

• The mere fact of carrying out an architectural evaluation forces the architects and developer to document, analyze and understand the project’s architecture.
• In the last years, several methods for evaluating software architectures have been defined
  – **Scenario-based Software Architecture Evaluation Methods**
    • SAAM, ATAM, ALPSM and ALMA, SBAR, SALUTA, SAAMCS, ESAAMI, etc.
  – **Mathematical Model-based Software Architecture Evaluation**
    • Software Architecture-based Reliability Analysis (e.g., Cheung [30])
    • Software Architecture-based Performance Analysis (e.g., BIM, CM, ABI)
  – **Metrics-based Software Architecture Evaluation Methods**
    • **Early evaluation:** QuADAI
    • **Late evaluation:** TLC, FA, MNS, SSC

• In this lab exercises, we will be working with **ATAM** and **QuaDAI**.
  – The first method is a scenario-based method while the second one is a metrics-based architecture evaluation method.
References


Introduction to Software Architecture Evaluation