Electronics and
Computer Science

University of Southampton

Software Architecture

COMP6226: Software Modelling Tools and Techniques for Critical Systems

Dr A. Rezazadeh (Reza)
Email: ra3@ecs.soton.ac.uk or ar4k06@soton.ac.uk

October 24

University of Southampton

Overview

- What Is Software Architecture?
- Why is software architecture important?
- The software architect role – Making architecture work through collaboration
- What Makes a “Good” Architecture?
- Why Architectural Design?
- Architecture and Abstraction
- Architecture and Modularity
- The Role of Architecture – Managing Complexity
- The Role of Architecture – Cohesion
- The Role of Architecture –Coupling

University of Southampton

What Is Software Architecture?

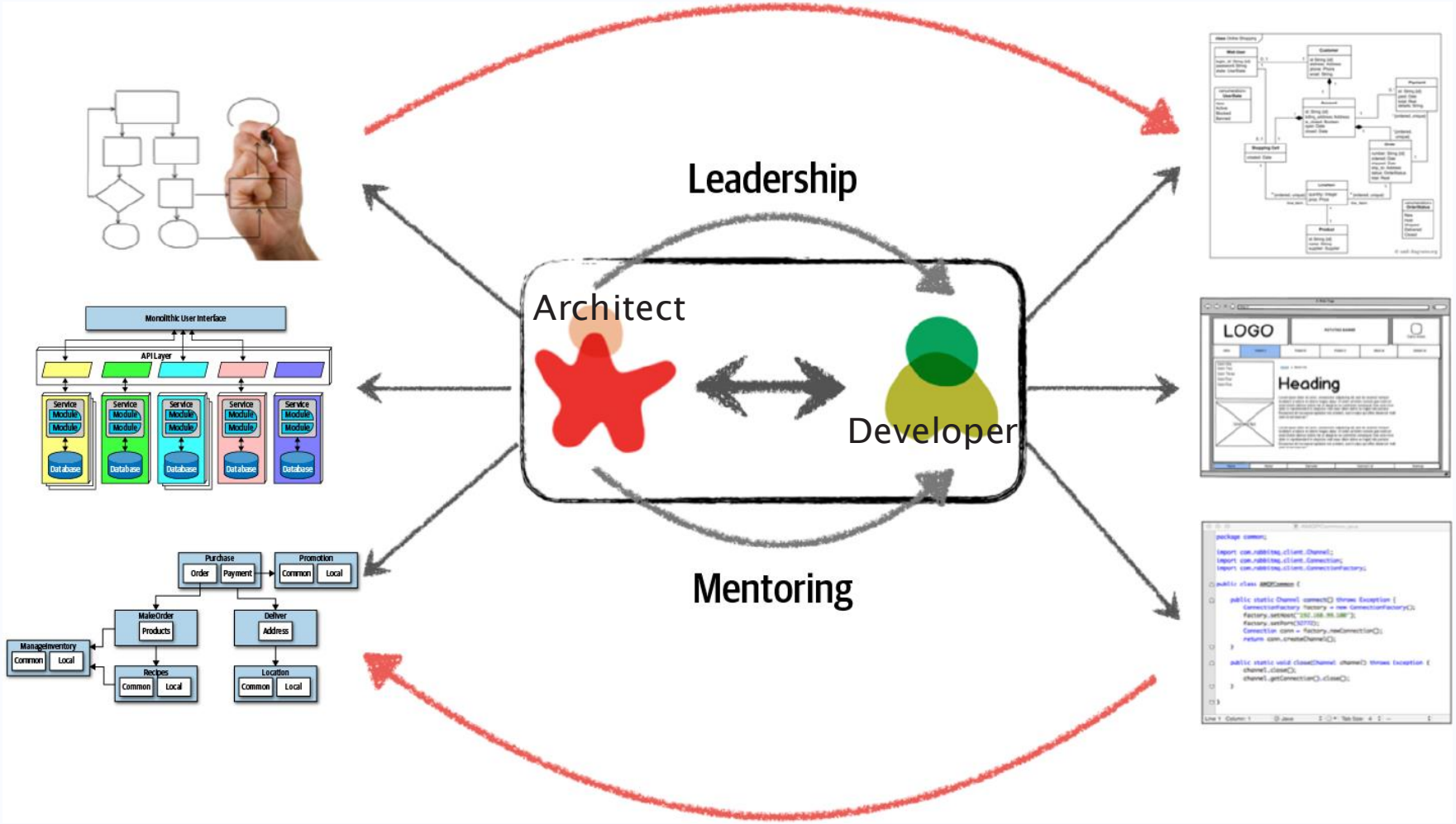
- The definition of software architecture has long been argued.
- For some people, it's the way essential compounds are wired together or the fundamental organisation of a system.
  - In this regard, the abstract concept proposed by Ralph Johnson, Associate Professor, University of Illinois, is something noteworthy:
  - "*Architecture is about the important stuff. Whatever that is.*"
- A high-level view of a software system, which describes aspects such as:
  - The software components and component dependencies and interconnections within the system.
  - Links and interfaces to any external systems.
  - The distribution structure

Why is software architecture important?

- Architectural designs can help to plan, manage and document a software development, to:
  - Aid project management.
  - Assign different component developments to different teams.
  - Centralise coordination of how the system components should fit together.
  - Document decisions about frameworks, platforms, structure, etc.
  - Document how non-functional properties of systems are achieved, such as performance, reliability, security, etc.
  - Document and modularise the system to aid in future maintenance and evolution.

University of Southampton

The software architect role – Making architecture work through collaboration



University of Southampton

What Makes a "Good" Architecture?

- There is no such thing as an inherently good or bad architecture.
- Architectures are either more or less fit for some purpose.
- For example, a three-tier layered service-oriented architecture may be just the ticket for a large enterprise's web-based B2B system but completely wrong for an avionics application.

Why Architectural Design?

- Making architecture explicit has main benefits or objectives:
  - Clarifying ideas and decisions: Architectural concepts provide a vocabulary that can aid our understanding of an application's high-level design (model).
  - Analysing properties of the architectures: The architecture can help with checking for consistency across different characteristics of a design.
  - Stakeholder communication and understanding: Sharing architecture information among members of a development team, or between different teams.
  - Large-scale reuse: the high-level components com- prising a system architecture may assist with identifying where there is scope for reuse.
  - Management: Anticipating how an application may evolve is (or should be) a factor in determining its architectural form.

Architecture and Abstraction

- At one level, architecture is about the structure of individual services, applications and programs.
  - At this level, we are concerned with the internal structure and decomposition of a single program.
- At another level, architecture is about sophisticated enterprise systems that comprise systems of systems.
  - Enterprise systems often involve interacting collections of programs harnessed to create some overarching and coordinated set of functions.
- Hence, we use architecture to simplify and clarify different levels of abstraction.
  - Abstraction is the process of hiding implementation details from users.

Architecture and Modularity

- If an architect designs a system without paying attention to how the pieces wire together, they end up creating a system that presents myriad difficulties.
- A module is defined as “each of a set of standardised parts or independent units that can be used to construct a more complex structure.”
- For discussions about architecture, we use modularity as a general term to denote a related grouping of code: classes, functions, or any other grouping.
  - This doesn’t imply a physical separation, merely a logical one;

Software Architecture – Modularity

- Modularity is important principle of software architecture.
  - This principle states that software systems should be designed as a collection of independent modules.
- Each module should be responsible for a specific task or function.
  - Modules should be loosely coupled, meaning that they should not depend on each other too much.
- Modularity helps to improve the scalability and maintainability of software systems.
  - It also makes it easier to test and debug the system.

University of Southampton

The Role of Architecture – Managing Complexity

- Complexity in designing software makes the software development process difficult.
- Reducing the complexity is the biggest challenge in software design.
- How is this done?
- The best way to reduce complexity is by dividing the entire design into manageable parts.

The Role of Architecture – Defining structure and behaviour

- Software architectures consist of three main parts:
  - Components – representing computational/data storage nodes.
  - Connectors – for information interchange between components.
  - Configurations – which instantiate components and connectors in particular arrangements.
- Having an overview of the architectural dynamics helps us answer questions such as the following:
  - Which are the main components of my system?
  - How do these components interact and evolve?
  - What resources will I need and what costs will I have in the development process?
  - What are the areas where I can predict change?

The Role of Architecture – Cohesion

- *Cohesion* refers to what extent the *parts of a module* should be *contained* within the *same module*.
- In other words, it is a measure of how *related the parts* are to one another.
- Ideally, a *cohesive* module is one where all the parts should be packaged together.
  - Because breaking them into smaller pieces would require *coupling* the parts together via calls between modules to achieve useful results.
- Attempting to divide a cohesive module would only result in increased *coupling* and decreased readability.

Types of Cohesion

- *Functional cohesion*
  - Every part of the module is related to the other, and the module contains everything essential to function.
- *Sequential cohesion*
  - Two modules interact, where one outputs data that becomes the input for the other.
- *Communicational cohesion*
  - Two modules form a communication chain, where each operates on information and/or contributes to some output.
  - For example, add a record to the database and generate an email based on that information.

Types of Cohesion – Cont.

- *Logical cohesion*
  - The data within modules is related logically but not functionally.
  - For example, consider a module that converts information from text, serialised objects, or streams.
  - Operations are related, but the functions are quite different.
  - A common example of this type of cohesion exists in virtually every Java project in the form of the StringUtils package: a group of static methods that operate on String but are otherwise unrelated.

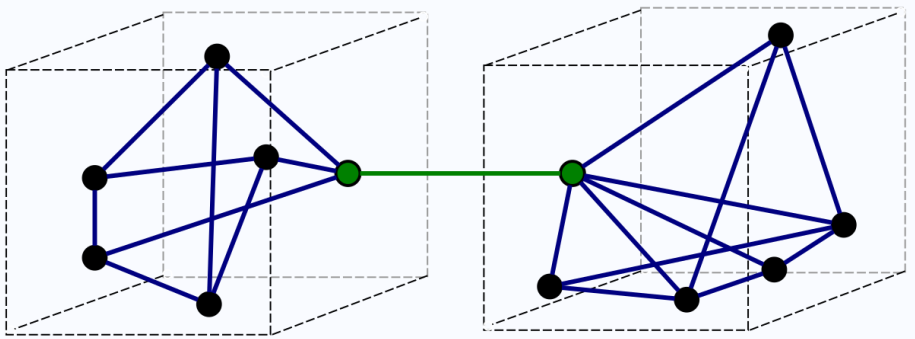
Types of Cohesion – Cont.

- *Procedural cohesion*
  - Two modules must execute code in a particular order.
- *Temporal cohesion*
  - Modules are related based on timing dependencies.
  - For example, many systems have a list of seemingly unrelated things that must be initialized at system startup; these different tasks are temporally cohesive.
- *Coincidental cohesion*
  - Elements in a module are not related other than being in the same source file; this represents the most negative form of cohesion.

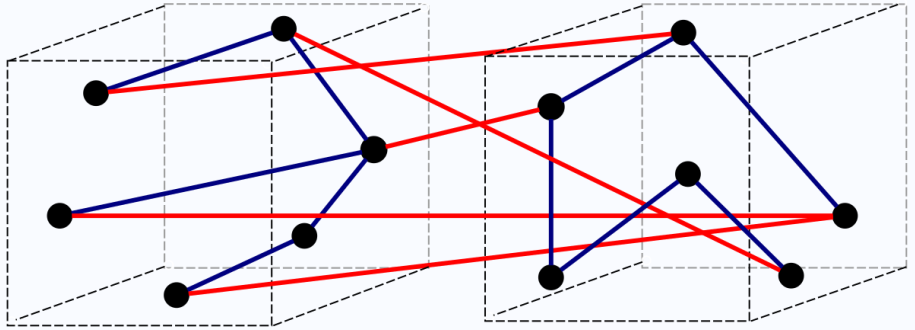
University of Southampton

The Role of Architecture –Coupling

- Coupling refers to the degree of interdependence between different modules, classes, or components of a software system.



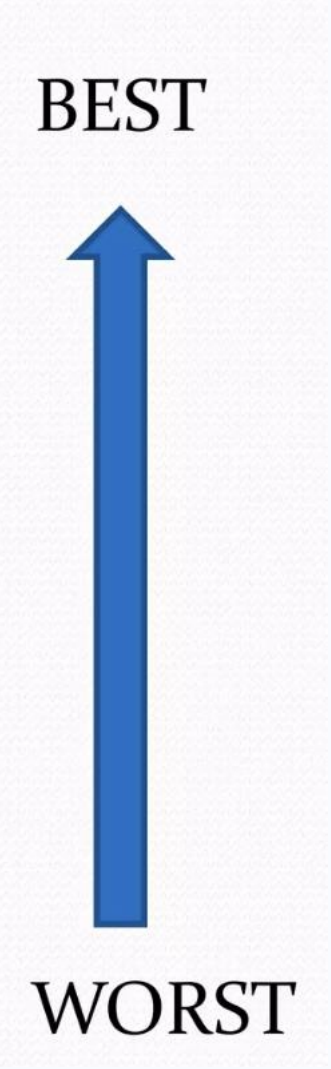
a) Good (loose coupling, high cohesion)



b) Bad (high coupling, low cohesion)

Types of Coupling

1. Data Coupling: When modules shared primitive data between them.
2. Stamp Coupling: When modules shared composite or structural data between them, and It must be a non-global data structure. for example, Passing object or structure variable in react components.
3. Control Coupling: When data from one module is used to direct the structure of instruction execution in another.
4. External Coupling: When two modules shared externally imposed data type that is external to the software like communication protocols, device interfaces.
5. Common Coupling: When two modules shared the same global data & dependent on them, like state management in JavaScript frameworks.
6. Content Coupling: When two modules shared code and can modify the data of another module, which is the worst coupling and should be avoided



Software Architecture Patterns

- Software architecture patterns can be defined as solutions to mainstream and recurring software engineering problems.
- An architectural pattern is a set of architectural design decisions that address recurring design problems in various software development contexts.
  - It offers rules and principles for organizing the interactions between predefined subsystems and their roles.

Software architecture pattern vs. design pattern

	Architecture Patterns	Design Patterns
Defination	Fundamental structural organization for software systems	Specification that could help in implementation of a software
Role	Conversion of software characteristics to a high-level structure.	Description of all the units of the software system to support coding
Example	Microservice, serverless and event-driven	Creational, structural and behavioral
Level	Large Level tool - concerns large scale components, global properties, and mechanism of the system	Small level tool- concerns schemes for refining and building smaller subsystems - structure and behavior of entities and their relationships
Problem Addressed	Distributed functionality, system partitioning, protocols, interfaces, scalability, reliability, security	Problems in software construction

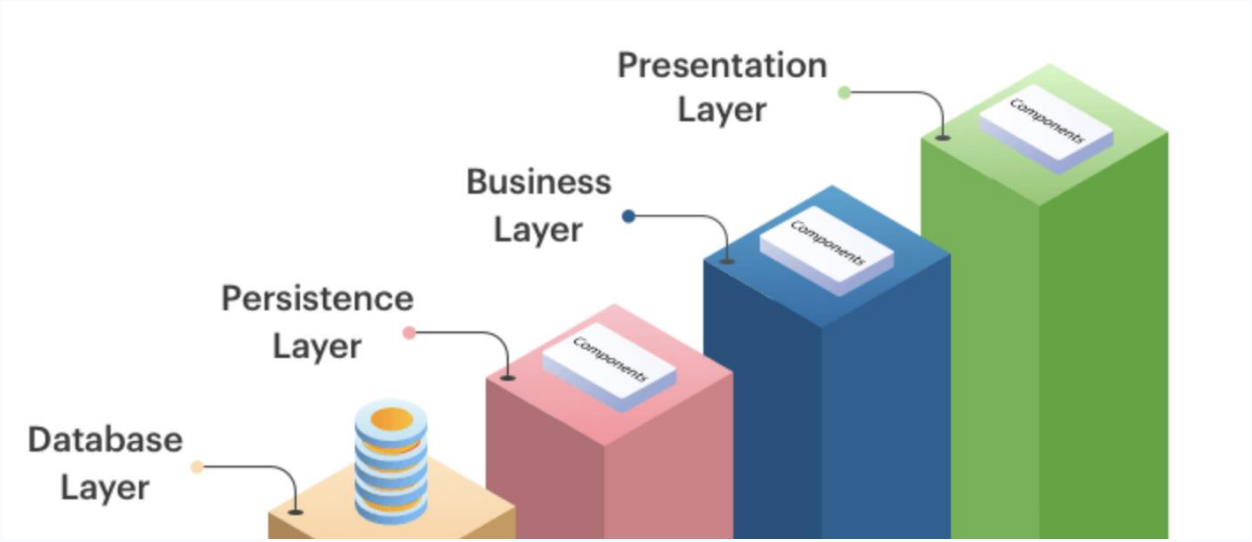
University of Southampton

Different types of software architecture pattern

- Layered Architecture Pattern
- Event-driven Architecture Pattern
- Microkernel Architecture Pattern
- Microservices Architecture Pattern
- Space-Based Architecture Pattern
- Client-Server Architecture Pattern
- Master-Slave Architecture Pattern
- Pipe-Filter Architecture Pattern
- Broker Architecture Pattern
- Peer-to-Peer Architecture Pattern

Layered Architecture Pattern

- multi-layered, aka tiered architecture, or n-tier architecture.
- This pattern stands out because each layer plays a distinct role within the application and is marked as closed.
  - It means a request must pass through the layer below it to go to the next layer.



- Another one of its concepts – layers of isolation – enables you to modify components within one layer without affecting the other layers.

exam

Layered Architecture Pattern

advantages and shortcoming Architecture and situation

- Usage:
  - Applications that are needed to be built quickly.
  - Enterprise applications that require traditional IT departments and processes.
  - Appropriate for teams with inexperienced developers and limited knowledge of architecture patterns.
  - Applications that require strict standards of maintainability and testability.
- Shortcomings:
  - Unorganized source codes and modules with no definite roles can become a problem for the application.
  - Skipping previous layers to create tight coupling can lead to a logical mess full of complex interdependencies.
  - Basic modifications can require a complete redeployment of the application.

University of Southampton

YOUR QUESTIONS