Principles of Software Design
Architecture

Robert Lukoťka
lukotka@dcs.fmph.uniba.sk
www.dcs.fmph.uniba.sk/~lukotka

M-255

What is architecture?

"Software architecture encompasses the set of significant decisions about the organization of a software system including the selection of the structural elements and their interfaces by which the system is composed; behavior as specified in collaboration among those elements; composition of these structural and behavioral elements into larger subsystems; and an architectural style that guides this organization. Software architecture also involves functionality, usability, resilience, performance, reuse, comprehensibility, economic and technology constraints, tradeoffs and aesthetic concerns."

Philippe Kruchten, Grady Booch, Kurt Bittner, and Rich Reitman

What is architecture?

"The highest-level breakdown of a system into its parts; the decisions that are hard to change; there are multiple architectures in a system; what is architecturally significant can change over a system's lifetime; and, in the end, architecture boils down to whatever the important stuff is."

Martin Fowler

What is architecture?

"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. Architecture is concerned with the public side of interfaces; private details of elements—details having to do solely with internal implementation—are not architectural."

Len Bass, Paul Clements, Rick Kazman

SW architecture

Architecture should [2]:

- Expose the structure of the system but hide the implementation details.
- Realize all of the use cases and scenarios.
- Try to address the requirements of various stakeholders.
- Handle both functional and quality requirements.

Expose the structure of the system: How to describe a SW system?

$4 + 1$ view model (Diagram):

- Logical view
- Process view
- Development view
- Physical view
- +1 Use case view

What is part of the system architecture?

Use case view

- What are the actors of the system.
- For what reasons the actor use the system.
- How to attain that goal.
- Which scenarios / use cases are of biggest value / carry biggest risk.
- ...

Example: Logical view

Architectural pattern: 3-layer architecture

- Presentation layer - user interface
- Application layer - classes
- Data access layer - database

Example: Logical view

Architectural pattern: 3-layer architecture

- Presentation layer - user interface
- Business logic layer - classes
- Data access layer - database

+ interfaces.

Physical view

- Deployment conditions (OS, DB server, WEB server, how the nodes communicate), technologies used.
- In large applications, how nodes are structured.

Example: Physical view

Architectural pattern: 3-tier architecture

- Presentation tier - web browser
- Application tier - web server
- Data tier - database server

You need to add lot of details: what browser, server, db, just one server/db is running?, load balancer ...

Process view

- Logical view is static. Sometimes not sufficient to explain how system works.
- Deals with concurrency and parallelism.
- Who owns threads used for program execution, which code given thread may execute?

Development view

- How development is organizes (VCS, packages, names of source files, progamming languages and other tool).
- How the source files map to the elements in the logical view.
-
- Which scenarios / use cases should be implemented as first.

What is architecture depends on the system size

- Very large system - mostly about the structure of the nodes and their relation
- Medium system - subsystem, interfaces between them
- Small system - components or even classes

Principles of Software Architecture

- Separation of concerns
- Single Responsibility principle
- Don't repeat yourself
- Minimize upfront design
- Build to change instead of building to last.
- Analyze and reduce risk.
- Use consistent principles within the components/layers/subsystems.
- ...

Interface

Interface is a shared boundary across which two or more separate components of a computer system exchange information [3].

- Hardware interfaces
- Software interfaces
  - ABI - Application binary interface - typically not relevant (created by compiler / other tools).
  - API - Application programming interface -
- User interfaces

Here, we will speak about API's.

Interface

Interface

is a contract between communicating parties, it typically has two sides:

- Side that **requires interface** (caller) - has a means to access the callee (memory/web address) using an agreed protocol (HTTP, C calling convention).
- Side that **implements interface** (callee) - does an action and may returns some value.

To make larger applications more tractable it is necessary that different parts of an applications communicate only via strictly defined interfaces.

Service

Service is refers to a software functionality or a set of software functionalities (such as the retrieval of specified information or the execution of a set of operations) with a purpose that different clients can reuse for different purposes, together with the policies that should control its usage [4].

- To access the service, the service defines an interface.
- Some definitions requires the service to be useful in various settings.
- In service oriented architecture services form separate application (communication e.g. through HTTP protocol)

SW architecture

Architecture should [2] (repeated list):

- Expose the structure of the system but hide the implementation details.

This is achieved by separating the functionality into different parts (subsystems / components / services) and defining the interfaces they use to communicate.

- Realize all of the use cases and scenarios.

The functionality has to be complete.

- Try to address the requirements of various stakeholders.

The functionality has to be complete.

- Handle both functional and quality requirements.

The architecture should allow to control those quality requirements.

Service design / Interface design

Service design

- Separation of concerns
- Single Responsibility principle
- Don't repeat yourself
- Build to change instead of building to last.

The principles are similar to design principles at lower level.

Service design / Interface design

Interface design

- Stateless service / Stateful interface - usage protocol
- Minimal interface vs Humane interface
- Uniformity

Interface design REST

Representational state transfer

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- Code on demand (optional)
- Uniform interface

Example

Interface design REST

Representational state transfer

- Client-server architecture
- Statelessness
- Cacheability
- Layered system
- Code on demand (optional)
- Uniform interface

Example

Can be extended by GraphQL.

How to find the architecture

Important questions [2]:

- What are the foundational parts of the architecture that represent the greatest risk if you get them wrong?
- What are the parts of the architecture that are most likely to change, or whose design you can delay until later with little impact?
- What are your key assumptions, and how will you test them?
- What conditions may require you to refactor the design?

How to find the architecture

You need to determine

- Application Type (mobile app, rich client, rich internet, service, web app, cloud, . . .)
- Deployment Strategy
- Appropriate Technologies
- Quality Attributes
- Crosscutting Concerns (authentication, logging, caching, . . .)

Example - Crosscutting Concerns

Crosscutting Concerns for security [2]:

- Auditing and Logging
- Authentication
- Authorization
- Configuration Management
- Cryptography
- Exception Management
- Input and Data Validation
- Sensitive data
- Session Management

You cannot focus on everything.

Key scenario [2]:

- It represents an issue—a significant unknown area or an area of significant risk.
- It refers to an architecturally significant use case
  - Business Critical.
  - High impact.
- It represents the intersection of quality attributes with functionality.
- It represents a trade-off between quality attributes.

How to decrease the risk involved

Order of analysis / implementation - high priority stuff:

- Things that are unknown / not well understood / the solution is not known
- Things used throughout the system, later change has high impact (e.g. cross-cutting concerns)
- Most valuable use cases

Testing the architecture [2]

- What assumptions have I made in this architecture?
- What explicit or implied requirements is this architecture meeting?
- What are the key risks with this architectural approach?
- What countermeasures are in place to mitigate key risks?
- In what ways is this architecture an improvement over the baseline or the last candidate architecture?

Architectural styles and patterns

Similar to design patterns:

- Provide abstract framework for a family of systems
- Help communication

Architectural styles and patterns

Architecture addresses a wide variety of issues, thus we have various types of styles/patterns

- Deployment
- Structure
- Communication
- Domain
- ...

Key architectural styles and patterns

- Client/Server - Segregates the system into two applications, where the client makes requests to the server. In many cases, the server is a database with application logic represented as stored procedures.
- Peer-to-peer - Each workstation has the same capabilities and responsibilities.

Key architectural styles and patterns

- Component-Based Architecture
  Decomposes application design into reusable functional or logical components that expose well-defined communication interfaces.
- Service-Oriented Architecture (SOA)
  Decomposes application into a collection of independently deployable services.
- Microservice architecture
  SOA + dump pipes, independent services + many more
- Message Bus
  SOA + intelligent pipes

Key architectural styles and patterns

- Object-Oriented
- Domain Driven Design
  An object-oriented architectural style focused on modeling a business domain and defining business objects based on entities within the business domain.
- Layered Architecture
  Partitions the concerns of the application into stacked groups (layers).
- N-Tier / 3-Tier
  Segregates functionality into separate segments in much the same way as the layered style, but with each segment being a tier located on a physically separate computer.
- Model-View-Controller

Key architectural styles and patterns

Most of the phrases associated with architectural styles patterns have additional meanings assoviated

- Good practices
- Restrictions
- Associated techniques
- Contents of a homonymous book.

User interface

How well should we separate user interface

- Code involving user interface should be separated - presentation code vs. domain code
- User interface changes frequently - it is good to isolate the changes
- Business logic should be independent of UI, UI depends on business logic
- Ideally, UI should communicate with business logic only through defined interface, however, sometimes it is practical to read values directly.
  - Writes should be done strictly through interface.
  - If representation of data changes, it is highly likety that also the interface needs to change.

Resources I

- Microsoft Application Architecture Guide

References I


SWEBOK V3 - Chapter 2.3, 2.4


Microsoft Application Architecture Guide, Software Architecture and Design


Wikipedia - Interface


Wikipedia - Service


M. Fowler - API Design


Wikipedia - REST


REST API example