Software Architecture in Practice

Erwann Wernli (@wrnli)
System architect, SBB
Agenda

Your job as an architect

Architecture and organisation

Your toolbox
Your job as an architect
Levels of architecture

- Enterprise
- Application
- Component
Modularisation

The system is decomposed in components
Each component has a lifecycle
Components provide and require interfaces
Two components connected form a dependency
Your job

- Group things that have the same lifecycle together ("What change together goes together")
- Expose interfaces that provide capabilities and hide internal details
- Manage dependencies
- Manage cross-cutting concerns
- (!) Several decompositions that are possible!
Well-known principles

- Single-responsibility principle
- Separation of concerns
- Information hiding
- Cohesion/coupling
## Examples

| Enterprise                          | ● Interfaces between IT domains  
|                                    | ● Customer Information - Logistics |
| Application                        | ● Interfaces between applications  
|                                    | ● Shopping Cart App - Product Catalog app |
| Component                          | ● Interfaces between classes  
|                                    | ● e.g. Data access layer - Business Logic  
|                                    | ● e.g. Layering |
You realize software systems with various technologies

- Platform technologies, e.g. AWS EC2
- Application technologies, e.g. Spring, HTTP
- Development technologies, e.g. IntelliJ, git, Jenkins

Technologies enable various architectural patterns:

- Eventual consistency vs. Strong consistency
- Synchronous vs Event-driven architecture
- …
Your job

- Define architectural patterns
- Select technologies for your architecture
- Balance “cost of ownership” vs “benefit”
- Balance standardisation vs specialisation of technologies
# Examples

| Enterprise          | ● Cloud providers, e.g. AWS vs Azure  
|                     | ● Central services, e.g. APIM, IAM   
|                     | ● Java vs C#                          |
| Application         | ● Synchronous vs Asynchronous API    
|                     | ● NoSQL vs Transactional DB          
|                     | ● SPA vs. Server-side Rendering      
|                     | ● Monitoring technology              |
| Component           | ● Spring Webflux (reactive) vs Spring Function  
|                     | ● Caching                            |
Every application has non-functional requirements, sometimes implicitly:

- Performance
- Availability,
- Maintainability,
- ...

Having 10 customers is not the same as 1 mio. Storing bank transactions is not the same as storing to-do lists.
Your job

- Figure out how to decompose the system (modularity) and use technologies to meet the NFRs.
- Balance NFR and costs of development / operations.
- Make the NFR and trade-off explicit
# Examples

| Enterprise          | ● Security strategy  
|                    | ● Multi-cloud strategy  
| Application         | ● 99.8 availability of application X  
|                    | ● Processing time of operation X < 50ms  
| Component           | ● Memory need of algorithm X  

Fit for purpose

https://www.enterpriseintegrationpatterns.com/ramblings/86_isthisarchitecture.html
## Fit for purpose

<table>
<thead>
<tr>
<th></th>
<th>Architecture 1</th>
<th>Architecture 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modularity</strong></td>
<td>Webapp with product search and shopping cart combined</td>
<td>Product search and shopping cart as microfrontend</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>LAMP (Linux/Apache/Mysql/PHP)</td>
<td>Java, Spring, Postgres, Docker, Kafka</td>
</tr>
<tr>
<td><strong>NFR</strong></td>
<td>Everybody work in the same codebase (Maintainability)</td>
<td>Two codebases (Maintainability)</td>
</tr>
<tr>
<td></td>
<td>Single database for search and shopping cart (Scalability)</td>
<td>Read model for product search can scale independently of write model for shopping cart (Scalability)</td>
</tr>
</tbody>
</table>
Architecture and Teamwork

If development is collaborative, managing collective knowledge is a challenge.

Architecture tends to reflect the organisation (and not the way around)

Knows Component A

Knows Component B

https://en.wikipedia.org/wiki/Conway%27s_law
Your job

- Communicate the architecture
- Define guidelines
- Share knowledge
- Align people
- Decentralize decision making
- Co-evolve architecture and organisation
Who's the architect?

- Architect decides, team executes
  - "Benevolent Dictator"

- Architect in the team
  - "Primus inter pares"

- Architecture done by team members
  - "Architecture without architects"

- "Implicit" architecture
  - "The inmates running the asylum."

https://architectelevator.com/transformation/agile_architecture/
Approaches to architecture

Retired: ivory tower architect
Tired: hands-on architect
Wired: architect as change agent

Static
Cost-optimization mentality

Dynamic
Business enabling mentality
Your toolbox

Tools

- Architecture styles
- Architecture patterns
- Architecture frameworks *
- Architecture templates **
- UML
- ...

Methods

- Modell visually
- Use architecture viewpoints
- Domain-Driven Design
- Record architecture decisions
- Automate architecture checks***
- ...

* e.g. AWS Well-architected Framework
** e.g. Arc42, 4+1
*** e.g. ArchUnit
Your My bookshelf

Enterprise

Application

Component

Inspired from: https://architectelevator.com/architecture/architect-bookshelf/
Summary

- There are three main facets to architecture: modularisation, technology, and NFR.
- Architecture and organisation go hand-in-hand in a modern organisation.
- There’s a vast body of knowledge around architecture - use it!