

Software Engineering DD1365 MVK-09

Gästföreläsning om IT-arkitektur

John Hallén

john.hallen@netlight.se

Defining architecture

- Software engineering is not an art – it is classical engineering.
- Plan, measure and act in a methodical process that is refined all the time.

Why architecture?

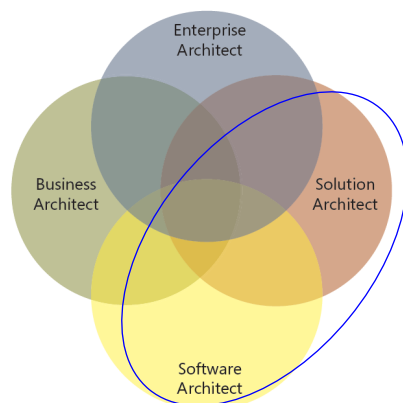
What is the alternative?

1. Define the problem
2. Bridge business to technology (agree the problem)
3. Develop a solution

- Yesterday: IT automated manual processes
- Today: Trim IT in alignment with company business and use IT as differentiator in competition with other companies

3

Roll of the architect



IASA - 4 architect roles

Source: www.iasa.se

- **3 levels**

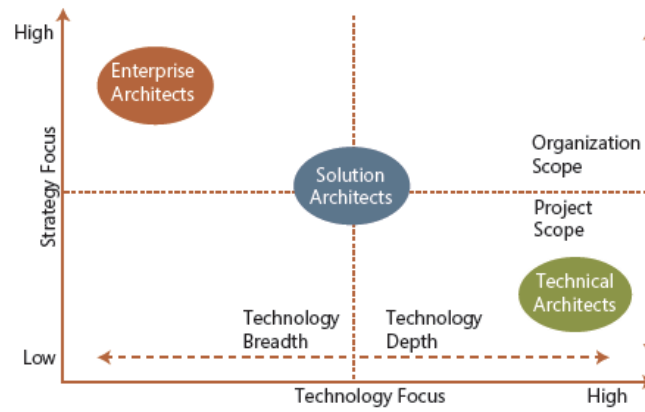
1. IT business strategies
2. Link 1 & 2
3. Technical architecture

→ **4 roles**

**We will talk about the
Solution + Software
architect.**

4

Architecture Journal #15



5

Other stakeholders

- Programmers
- Managers, colleagues
- Customers, 3rd party, tax payers
- Environment

6

Goal of the software architect

Build a common communication model

- Define the problem (requirements)
- Define the concepts (the domain model)
- Define the solution scope (artifacts)
- Paint the big picture AND continue to communicate

7

Competency needed

- **Strategic Mindset**
- **Communication and person skills**
- **Technical skills**
- **Domain knowledge**
- **Business sence**
- **Project management**
- **Experience**



8

Artifacts

Input

- Vision + Strategies
- Processes
- Requirements (high level as well as low level)
- Development model
- Resources / budget
- ...



Output

- Domain model
- Application diagrams
- Data models
- Refined requirements
- Specifications
- Communication
- Assumptions

9

Software architect toolbox

- Architectural principles *
- Standards
- Models *
- Patterns *
- Workshop/Taxonomy/Risk management ...
- (Word, PPT, ...)

10

Architectural principles

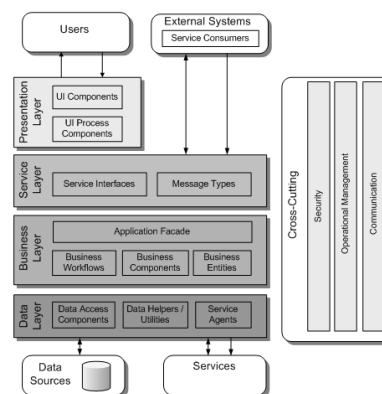
- Build to change over build to last
- Model to analyze and reduce risk
- Models and views are communication and collaboration tools
- Identify key engineering decisions

11

Architectural principles

Layered design

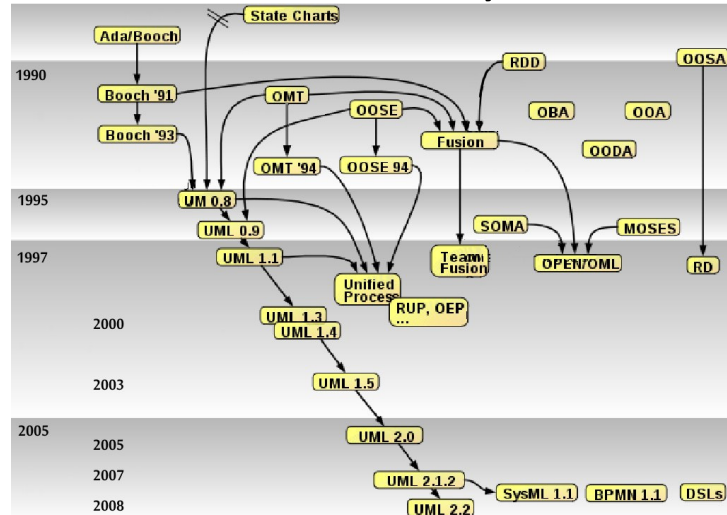
- Defined context
- Abstraction
- Encapsulation
- Separation of concern
- Functional layers
- High cohesion
- Loose coupling



Source: www.codeplex.com/AppArch

12

OO-history



13

Models / UML

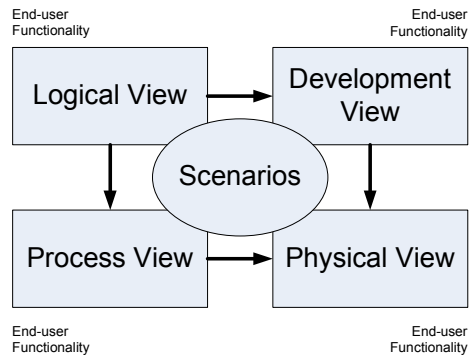
• UML ≠ RUP

- RUP is the process
- UML is the language

- Learn UML - Read the book, use it ...
- Keep it simple - do not overwork (KISS)

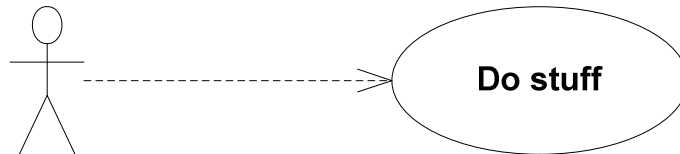
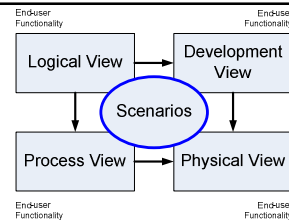
14

Kruchten 4+1 view

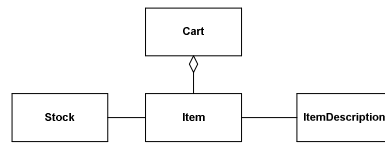
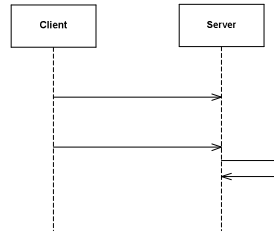
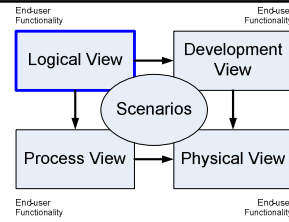


Different stakeholders → Different views

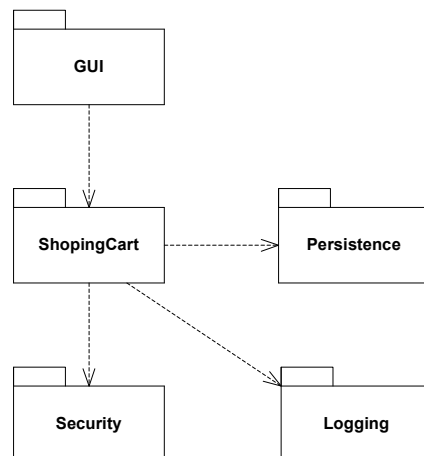
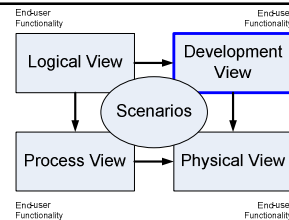
Scenarios



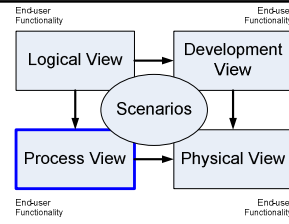
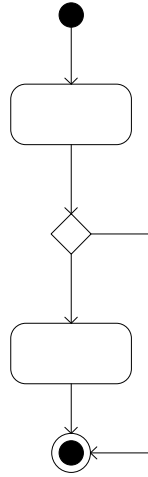
Locical View



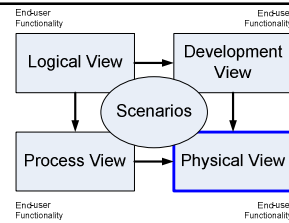
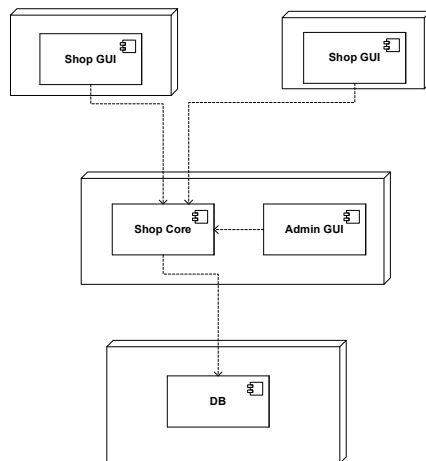
Development View



Process View



Physical View



Design patterns

- GoF : Design Patterns: Elements of Reusable Object-Oriented Software
- Tech staff can build common vocabulary by using patterns
- Non tech staff do not talk patterns fluently – try to use common language

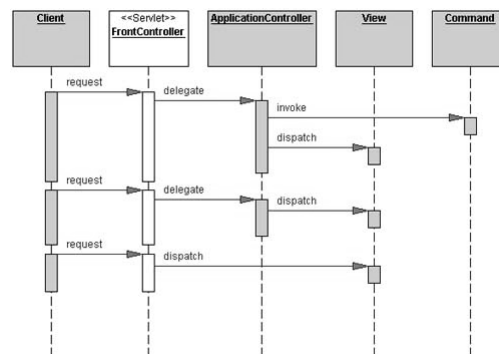
Common use is to combine different patterns in the specific solution.

Viktigt att påpeka att Patterns används som standardiserade begrepp
Dock ska man vara försiktig med arkitekter som bara pratar Patterns.

21

Example : Front Controller

Core J2EE Patterns



<http://java.boot.by/scea5-guide/>

22

Enterprise Integration Patterns

<http://eapatterns.com/toc.html>

Integration Styles	
	Introduction to Integration Styles
	File Transfer
	Shared Database
	Remote Procedure Invocation
	Messaging

Message Routing	
	Introduction to Message Routing
	Content-Based Router
	Message Filter
	Dynamic Router
	Router List
	Splitter
	Aggregator

Messaging Systems	
	Introduction to Messaging Systems
	Message Channel
	Message
	Filter and Filter
	Message Router
	Message Translator
	Message Endpoint

23

Architectural evolution

- Client server technology
- 3-tier / N-tier(Layered)
- Distributed
- Component based architectures
- SOA / EDA/ ...
- SaaS – cloud computing

24

In the real world

- Find the 'the usual suspects'
 - Functionality
 - Security, capacity, configuration
 - Integration and data needed
- Divide and Conquer
 - Divide into components by functionality
 - Verify against UC – the main UC + some error cases
 - Think about security, capacity, configuration and deployment
 - Talk to colleagues
 - Think and change (= measure and act)

25

My recommendations

- Each model shall have clear goal
 - Put focus on what is relevant
- Be consequent using your domain model terms
 - Be precise what the terms mean
- Good way to describe your models:
 - Simple image
 - Tables with explanations
 - Concrete examples
- Only use as many models as you need
 - Different context needs different views

26

Ref cases

1. IdM – identifiering och behörighetskontroll
2. SL – Signalsystem, trafikinformationssystem
3. ICA – SOA och EDA
4. Spel – startup