Software Architecting

Prof. Dr. M.R.V. Chaudron
chaudron@chalmers.se

Chalmers | Gothenburg University
Sweden

Schedule

Week		Date	Time	Lecture	Reading	Note
36	L1	4 sept	13:00 – 15:00	Introduction & Organization		
37	L2	11 sept	13:00 – 14:30	Architecting Process & Views	Ch 1 & 2	
37	S1	12 sept	10:15 – 12:00	<< Supervision/Assignment>>		
38	L3	18 sept	13:00 - 15:00	Requirements & Quality Attributes	Ch 3 & 4	
38	S2	19 sept	13:00 – 15:00	<< Supervision/Assignment>>		
38	L4	20 sept	13:15 – 15:00	Architectural Styles 1	Ch 13	
39	L5	25 sept	13:15 – 15:00	Architectural Styles 2	Ch 15 & 16	
39	S3	26 sept	10:15 – 12:00	<< Supervision/Assignment>>		
39	L6	27 sept	13:15 – 15:00	Roles and Responsibilities	Check Canvas	
40	L7	2 Oct	13:15 – 15:00	To be determined		UG
40	S4	3 Oct	10:15 – 12:00	<< Supervision/Assignment>>		UG
41		4 Oct	13:00 – 15:00	To be determined		UG
42	L8	9 Oct	13:15 – 15:00	Technical Debt (t.b.confirmed)		PhD defence
41	S5	10 Oct	10:15 – 12:00	<< Supervision/Assignment>>		
42	L9	16 Oct	13:15 – 15:00	Design Principles	Ch 21	
42	S6	17 Oct	10:15 – 12:00	<< Supervision/Assignment>>		check!
43	L10	18 Oct	13:15 – 15:00	Architecture Evaluation	tbd	
43	L11	23 Oct	13:15 – 15:00	Reverse Engineering & Correspondence		
43	L12	24 Oct	13:15 – 15:00	To be determined (slack)	Ch 20	
43	L13	25 Oct	13:00 – 15:00	To be determined (exam practice?)		

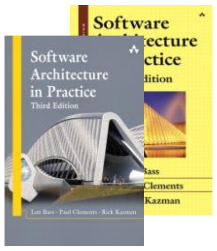
Supervision session

- Group formation
  - Hand in 'pairs of names'
  - We will randomly allocate 2 pairs into a team of 4 students
- Peer-evaluation for assessing individual contribution to assignments

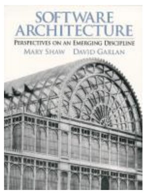
- Assignment follows typical steps:
  - Understand requirements (apply knowledge from RE course)
  - Identify stakeholders
  - Identify architectural drivers
  - Iterate:
    - Identify components & behaviours
    - Use patterns and tactics to achieve quality properties

  - Hand in: RE doc & SW ARCH DESIGN doc

Software Architecture Books



- Software Architecture in Practice, **3rd Edition**, L. Bass, P. Clements, R. Kazman, SEI Series in Software Engineering, Addison-Wesley, 2003

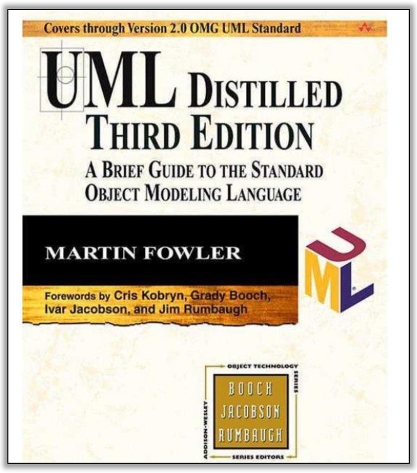


- Software Architecture: Perspectives on an Emerging Discipline, Mary Shaw, David Garlan, 242 pages, 1996, Prentice Hall

- Recommended Practice for Architectural Description, IEEE STD 1471-2000, 23 pages

UML book

- UML Distilled
  4th or 3rd edition


Covers through Version 2.0 OMG UML Standard
UML DISTILLED
THIRD EDITION
A BRIEF GUIDE TO THE STANDARD OBJECT MODELING LANGUAGE
MARTIN FOWLER
Forewords by Cris Kobryn, Grady Booch, Ivar Jacobson, and Jim Rumbaugh
OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

Outline

- Recap : What is Software Architecture?
- Stakeholders
- How to do Software Architecting?
- 4+1 Views
- Concluding Remarks

What is Software Architecture?

- recap

What is Software Architecture?

Classic Definitions 1

An architecture is the set of significant decisions about

- the organization of a software system,
- the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements,
- the composition of these structural and behavioural elements into progressively larger subsystems,
- the architectural style that guides this organization

The UML Modeling Language User Guide, Addison-Wesley, 1999
Booch, Rumbaugh, and Jacobson

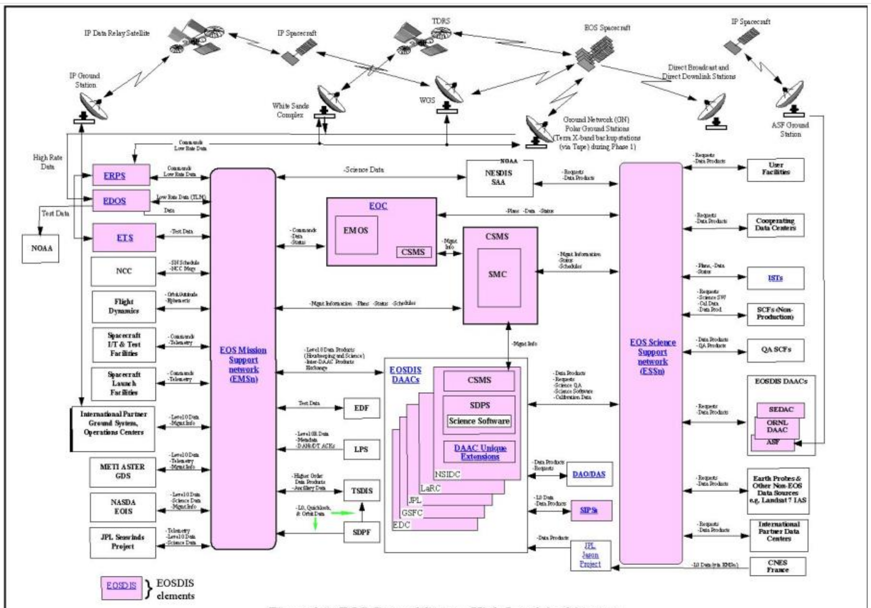


Figure 3-1. EOS Ground System High-Level Architecture

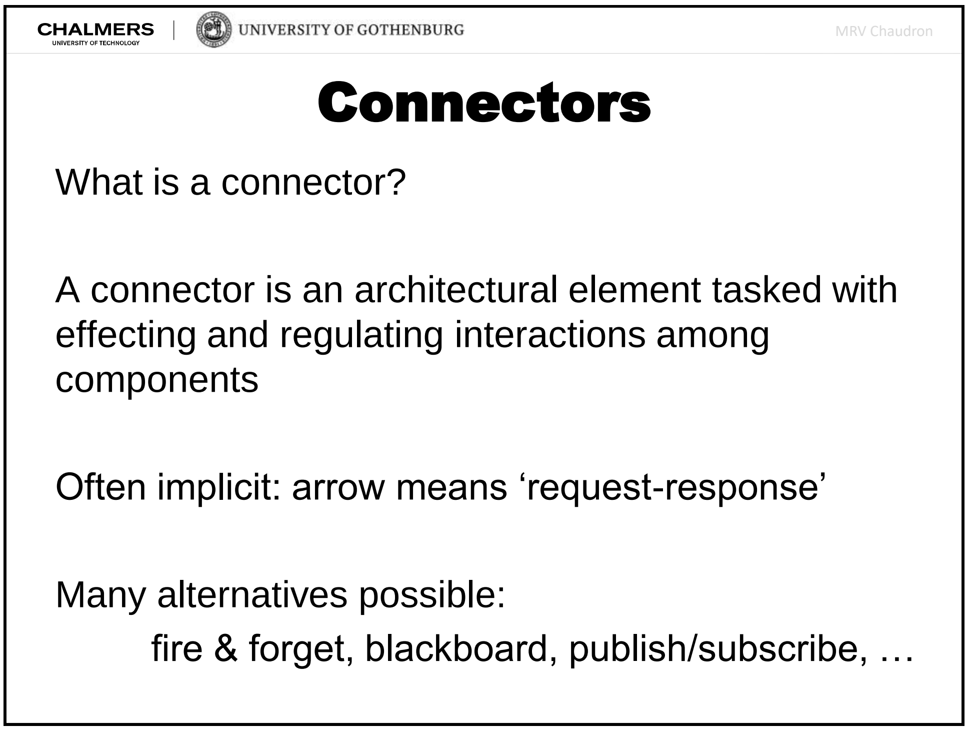
What is a **subsystem?**

A sub-system is a logical grouping of functionality

- Operations on the same data
- Functionality that belongs to the same responsibility

Nice to have's:

- Encapsulates functionality/data (information hiding)
- Explicit interfaces
- Explicit dependencies



Connectors

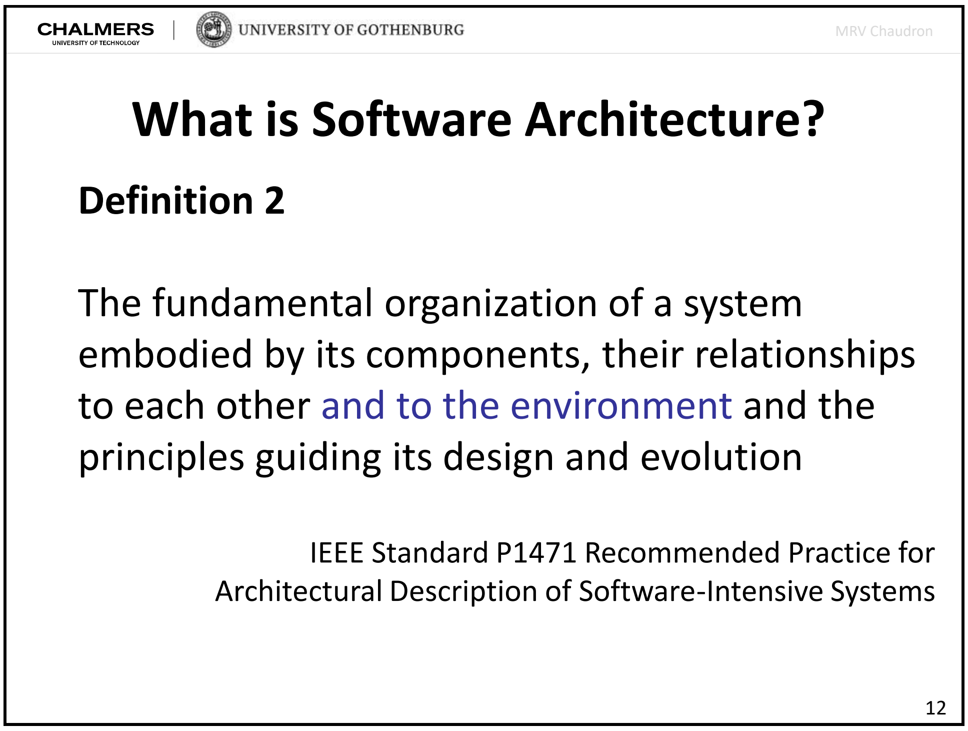
What is a connector?

A connector is an architectural element tasked with effecting and regulating interactions among components

Often implicit: arrow means ‘request-response’

Many alternatives possible:

fire & forget, blackboard, publish/subscribe, …



What is Software Architecture?

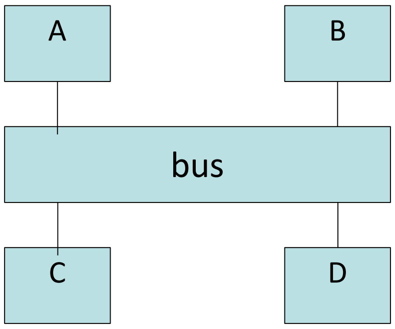
Definition 2

The fundamental organization of a system embodied by its components, their relationships to each other and to the environment and the principles guiding its design and evolution

IEEE Standard P1471 Recommended Practice for Architectural Description of Software-Intensive Systems

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

Connector example



CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

Architecture Model with explicit connectors

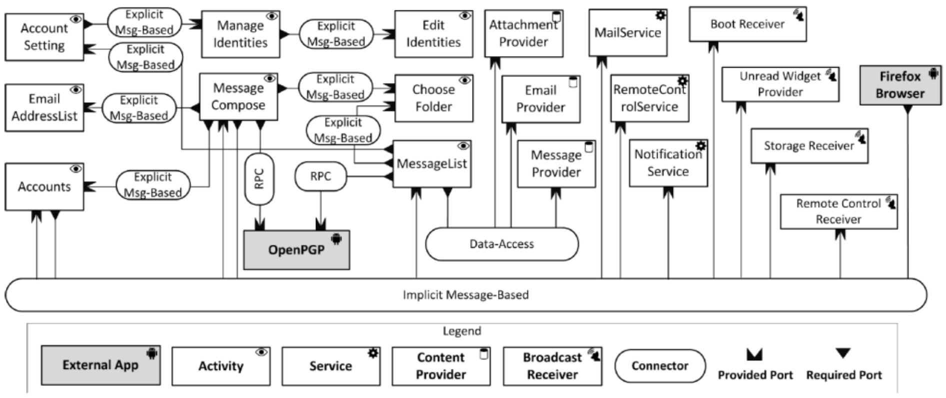


Figure 2: K-9 mail Android app architecture

Why, When and for Whom?

- Why architecting?
- For whom?
- When architecting?

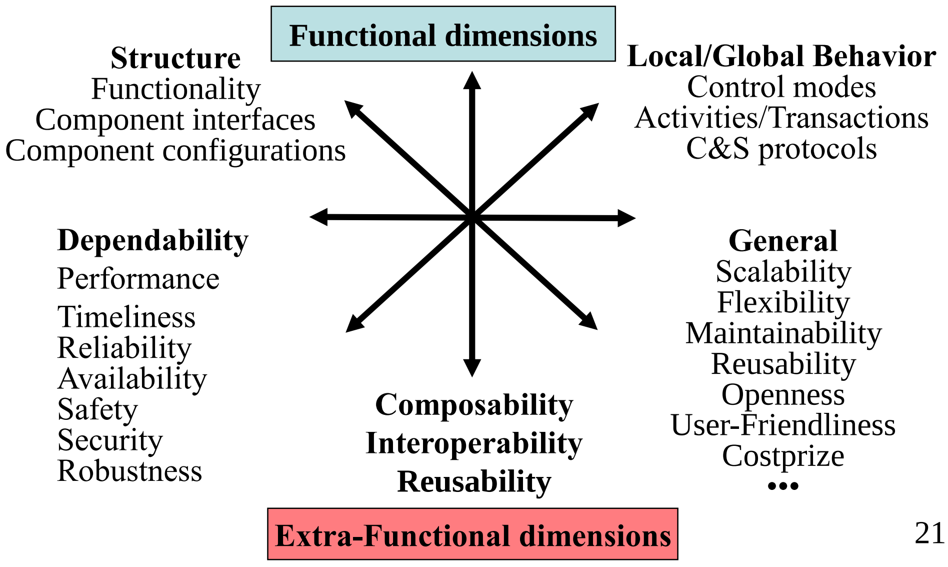
Developing a shared vision

Requirements emerge from a process of co-operative learning in which they are explored, prioritized, negotiated, evaluated, and documented.

Software Architecture & Quality

- The notion of *quality* is central in software architecting: a software architecture is devised to gain insight in the qualities of a system at the earliest possible stage.
- Some qualities are observable via *execution*: performance, security, availability, functionality, usability
- And some are ***not*** observable via execution, but in the development process: modifiability, portability, reusability, integrability, testability

Architecting = Balancing Objectives



CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

MRV Chaudron

Some more examples of *ilities

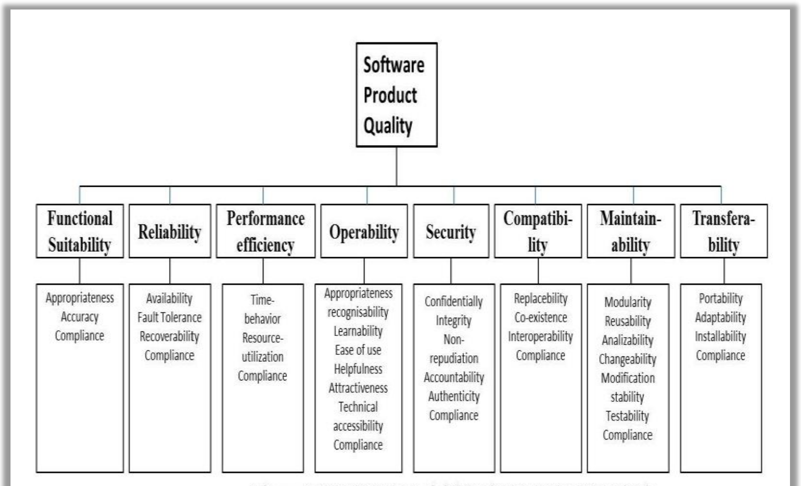
Accessibility, Understandability, Usability, Generality, Operability, Simplicity, Mobility, Nomadicity, Portability, Accuracy, Efficiency, Footprint, Responsiveness, Scalability, Schedulability, Timeliness, CPU utilization, Latency, Throughput, Concurrency, Flexibility, Changeability, Evolvability, Extensibility, Modifiability, Tailorability, Upgradeability, Expandability, Consistency, Adaptability, Composability, Interoperability, Openness, Integrability, Accountability, Completeness, Conciseness, Correctness, Testability, Traceability, Coherence, Analyzability, Modularity, Reusability, Configurability, Distributeability, Availability, Confidentiality, Integrity, Maintainability, Reliability, Safety, Security, Affordability, Serviceablility, …

22

CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

MRV Chaudron

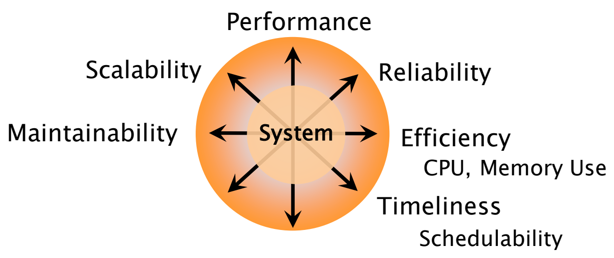
ISO standard on Software Product Quality



Software Product Quality
Functional Suitability
Reliability
Performance efficiency
Operability
Security
Compatibi-lity
Maintain-ability
Transfera-bility
Appropriateness Accuracy Compliance
Availability Fault Tolerance Recoverability Compliance
Time-behavior Resource-utilization Compliance
Appropriateness recognisability Learnability Ease of use Helpfulness Attractiveness Technical accessibility Compliance
Confidentially Integrity Non-repudiation Accountability Authenticity Compliance
Replacebility Co-existence Interoperability Compliance
Modularity Reusability Analizability Changeability Modification stability Testability Compliance
Portability Adaptability Installability Compliance

Figure 9 ISO 25010 Model (ISO/ IEC CD 25010 2007)

Extra Functional Properties



Essential system engineering problem:

- **a plurality of contradictory goals**
- **a plurality of means (technology, process)**
  each of which provides a varying degree of help or hindrance in achieving a given goal

Development Objectives of Software Architecture

- Management of **complexity**
  - Define a model of a system that is intellectually manageable
- Answering of **what-if** questions
  - Allows stakeholders to evaluate different architectural solutions and their consequences (e.g. on satisfying requirements)
- **Feasibility** study & **risk** analysis
  - Analysis of various (non-)functional features of the future product; identification of possible problems during development, production & operation
- Project **estimation**, **planning** & **organization**
  - Allocation of components to concurrent teams

Complexity Analysis: EMsn has quite many connections. Maybe we should split it up.

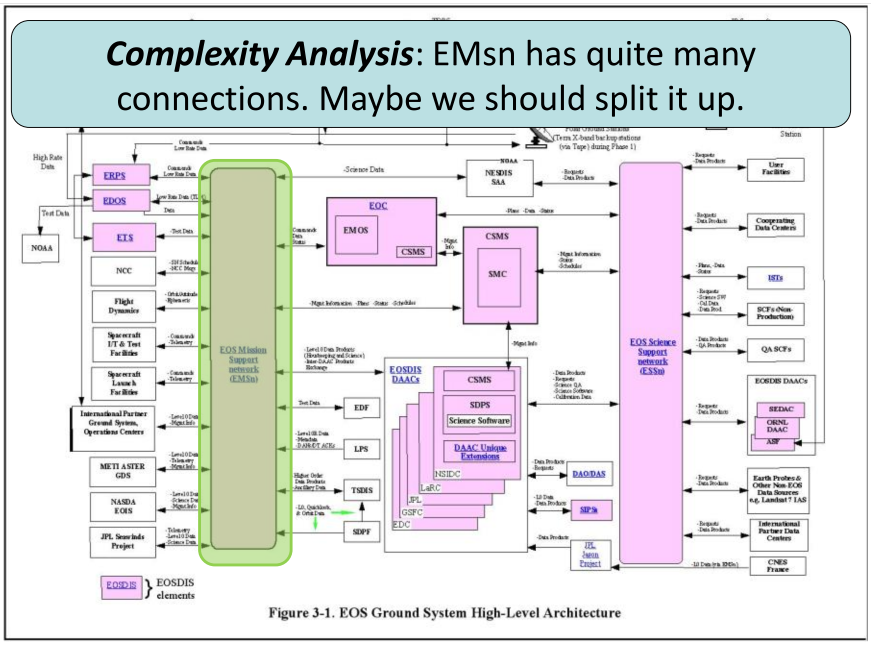


Figure 3-1. EOS Ground System High-Level Architecture

What if we change CSMS?

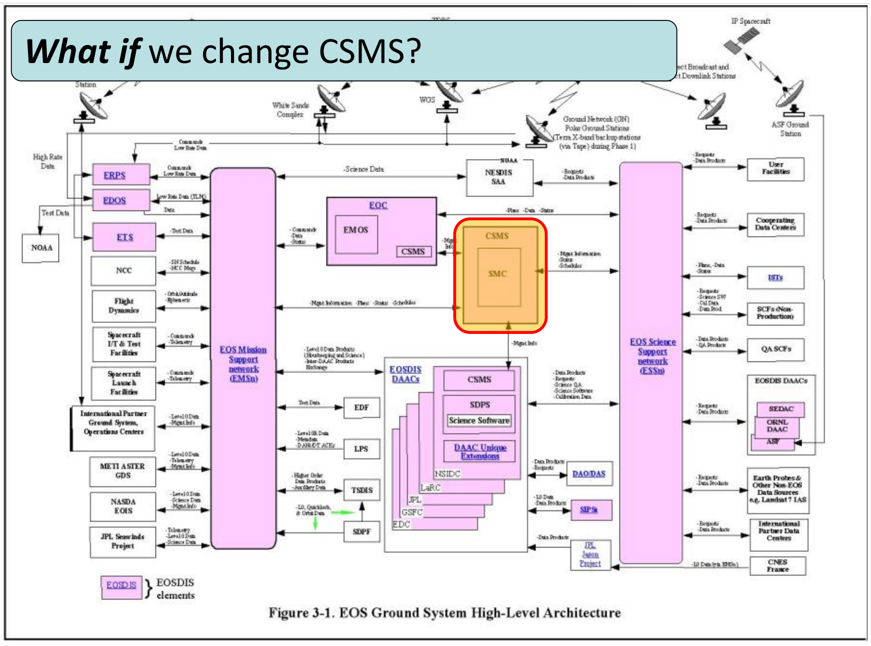


Figure 3-1. EOS Ground System High-Level Architecture

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

What if we change CSMS?

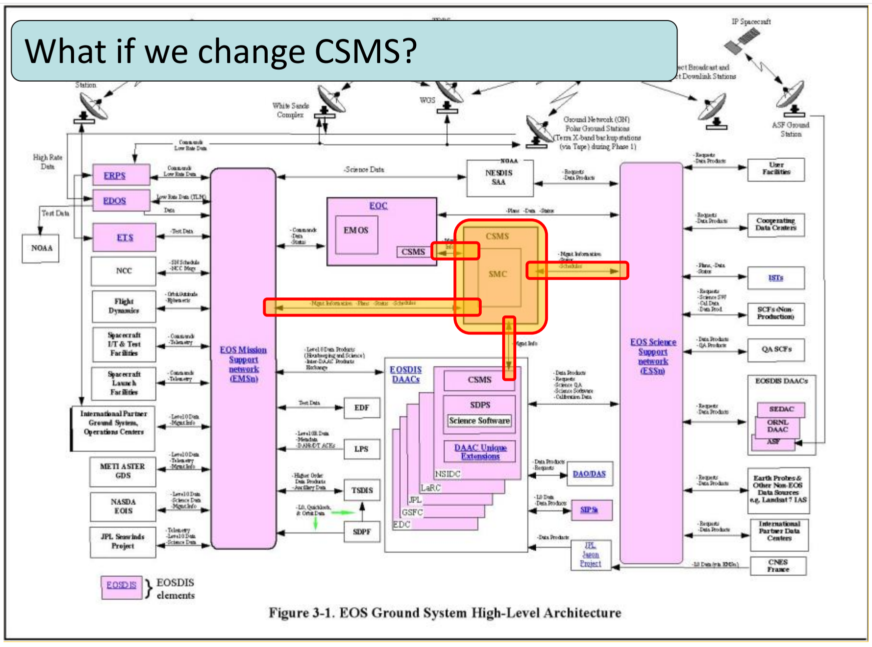


Figure 3-1. EOS Ground System High-Level Architecture

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

What if we change CSMS?

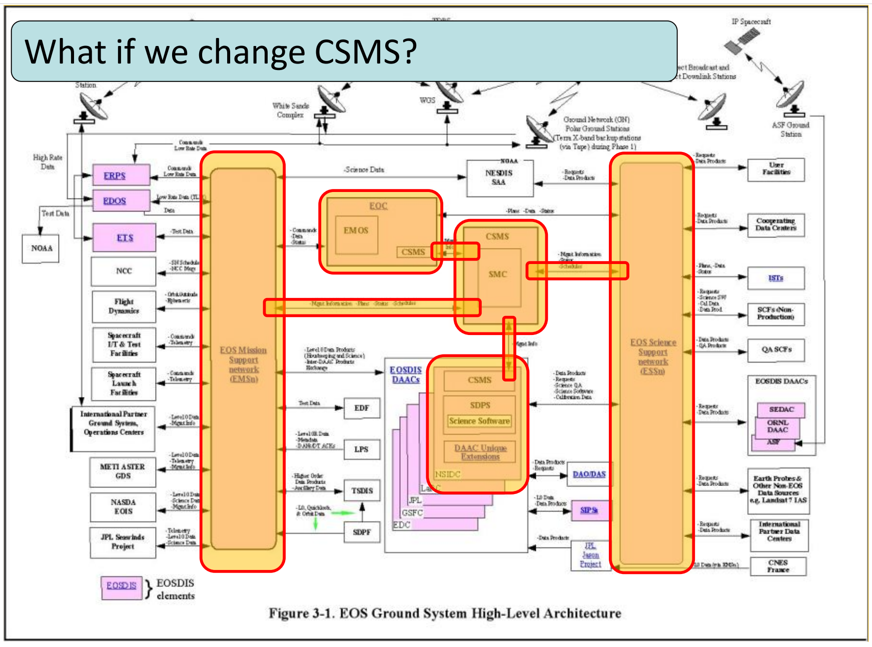


Figure 3-1. EOS Ground System High-Level Architecture

What if

- What happens if the load peaks?
- What happens if this connection fails?
- What happens if this technology changes?
- …

Feasibility and Risk

- Is there a business case for the system?

- Will the system be affordable?
- Will the system be able to handle peak load?
  - Is the security/compression/… fast enough?

Risks

- Which things can go wrong and what would their consequences be?
  - Both development and operation
  - Which things do we not yet know enough about?

Planning & Estimation

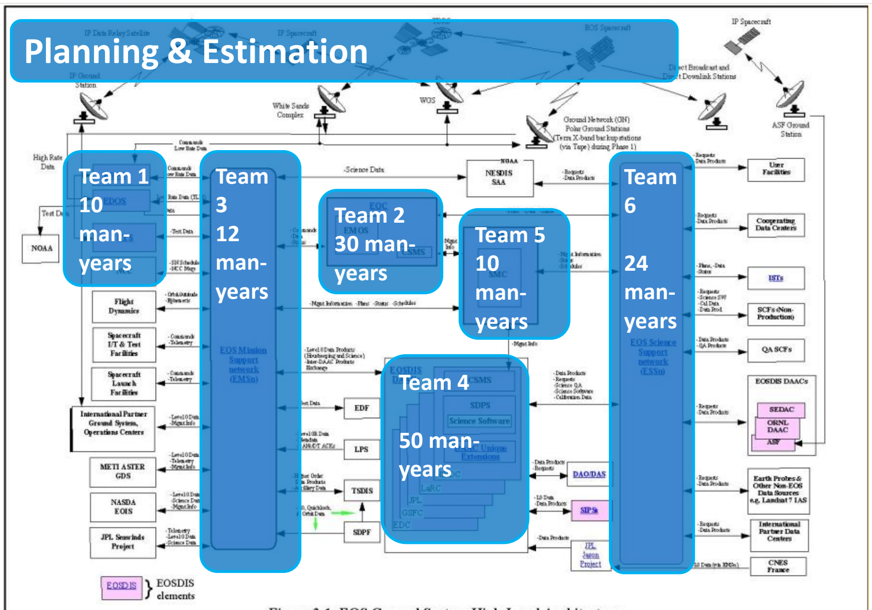
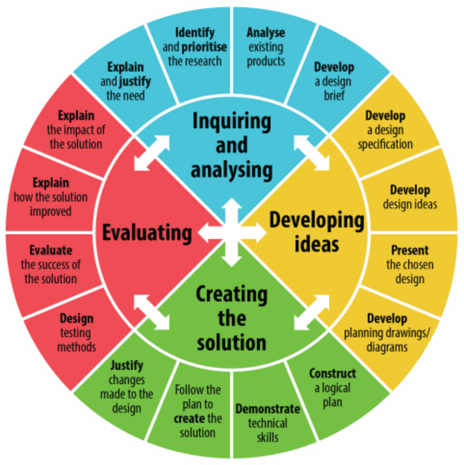


Figure 3-1. EOS Ground System High-Level Architecture

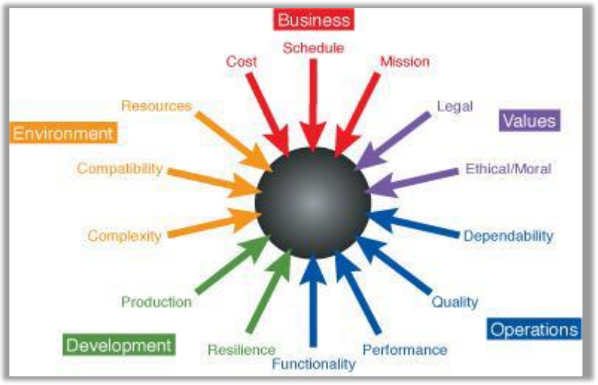
Software Architecting = Designing


Inquiring and analysing
Developing ideas
Creating the solution
Evaluating
Identify and prioritise the research
Analyse existing products
Develop a design brief
Develop a design specification
Develop design ideas
Present the chosen design
Develop planning drawings/ diagrams
Construct a logical plan
Demonstrate technical skills
Follow the plan to create the solution
Justify changes made to the design
Design testing methods
Evaluate the success of the solution
Explain how the solution improved
Explain the impact of the solution
Explain and justify the need

Respond to change:
- Iterative
- Feedback
- Evolve

Forces that affect the Design

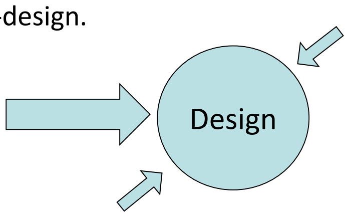
"In physics, a force is any influence that causes an object to undergo a certain change, either concerning its movement, direction, or geometrical construction.“
(wikipedia, Force)



The "software forces" image of below is from Grady Booch's Models09 keynote, The Other Side of Model Driven Development (2009):

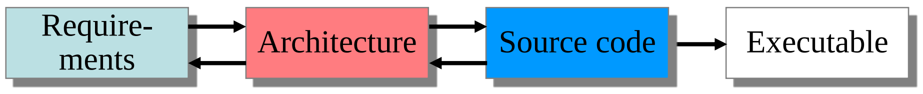
Architectural Drivers

- **Architectural drivers** are the design **forces** that will influence the early design decisions the architects make
- Architectural drivers are not all of the requirements for a system, but they are those requirements that are **most influential** to the architectures design.
- The ’art’ of the architect is to identify which forces have the strongest effect on the architecture-design.



Positioning Architecture

The question: *The answer:* *Implementation:* *Deployment:*



- Features
- Use cases
- Dependability
  Timing
  Reliability
  Security
- Quality
- Standards
- Etc.

- HL-Design
  Components
  Interfaces
  Interactions
- Styles
- Constraints
- Guidelines
- Reuse
- Etc.

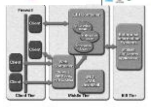
- Decomposition
- Algorithms
- Data structures
- Distribution
- Scheduling
- Recovery
- Language
- Encryption
- Etc.

- Memory allocation
- Dynamic Instantiation
- Call stacks
- Garbage collection
- Machine code
- Etc.

Levels of architecture*

Enterprise architecture

System architecture



Subsystem

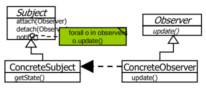
Application architecture

Application

Macro-architecture

Frameworks

Micro-architecture



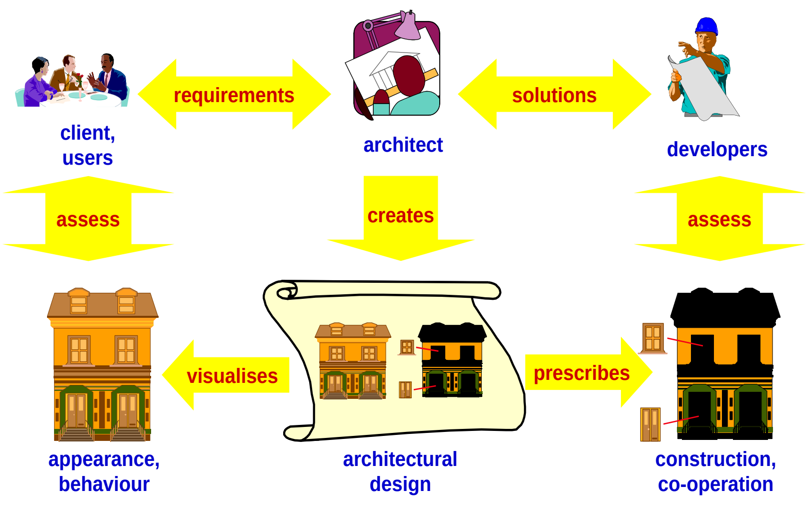
Design patterns

* Mowbray and Malveau

Outline

- Recap : What is Software Architecture?
- Stakeholders
- How to do Software Architecting?
- 4+1 Views
- Concluding Remarks

The Role of the Architect



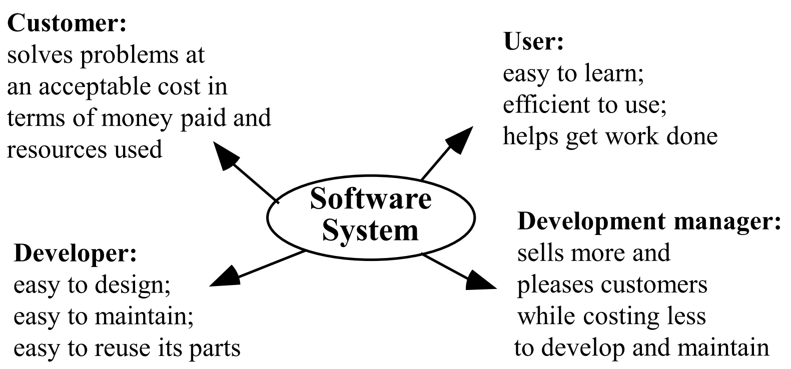
For Whom?

- An architecture is a (common) **means of understanding** of a system
  - Customers, Users, Domain Experts
  - Engineers:
    - Analysts
    - Architects
    - Programmers: maintenance, development, testing
    - New members of the development team
  - Marketing
  - Sales
  - Management

Stakeholders

"4.16 Stakeholder: An interested party having a right, share or claim in the system or in its possession of qualities that meet their needs."

Draft Standard ISO/IEC 15288 (ISO/IEC 1999)



Customer:
solves problems at
an acceptable cost in
terms of money paid and
resources used

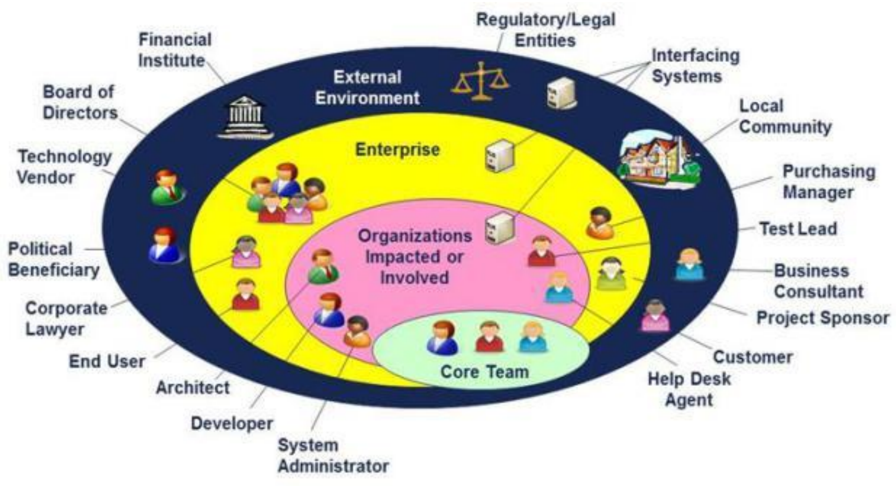
User:
easy to learn;
efficient to use;
helps get work done

Developer:
easy to design;
easy to maintain;
easy to reuse its parts

Development manager:
sells more and
pleases customers
while costing less
to develop and maintain

Figure from: Lethbridge and Laganiere

Stakeholders



Stakeholders & their Concerns 1/2

(Table 3.1 in BCK)

Stakeholder	*Concern (Examples)*
Customer	Business goals
	Schedule & budget estimation
	Feasibility and risk assessment
	Requirements traceability & progress tracking
	Product-line compatibility
User	Consistency with requirements & use cases
	Future requirements growth accommodation
	Support of dependability & other X-abilities
Service manager	Reliability, availability and maintainability

Stakeholders & their Concerns 2/2

Stakeholders	*Concern (Examples)*
System engineer	Requirements traceability Support of tradeoff analyses Completeness of architecture Consistency of architecture with requirements
Developer	Sufficient detail for design and development Workable framework for system construction, e.g. selection/assembly of components & technologies Resolution of development risks
Maintainer	Guidance on software modification Guidance on architecture evolution Interoperability with existent systems

When Architecting?

- When developing a **new system**
- When **changing a system**
  - if an architecture description is not available, or insufficient, as a basis for change
  - adapt the architecture documentation to changes
- When **integrating** existing systems
- For special **communication needs** to provide a common ground for understanding

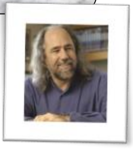
Outline

- What is Software Architecture?
- Stakeholders
- How Software Architecting?
- 4+1 Views
- Concluding Remarks & References

Architecture is making decisions

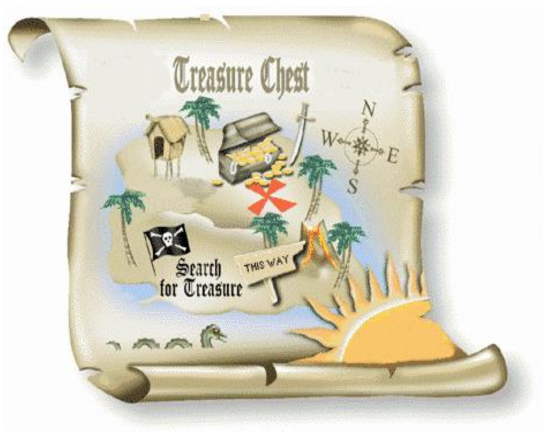
THE LIFE OF A SOFTWARE ARCHITECT IS A LONG (AND SOMETIMES PAINFUL) SUCCESSION OF SUBOPTIMAL DECISIONS MADE PARTLY IN THE DARK.

GRADY BOOCH



- You will not have all information available
- You will make mistakes, but you should learn from them
- There is no absolute measure for ‘goodness’

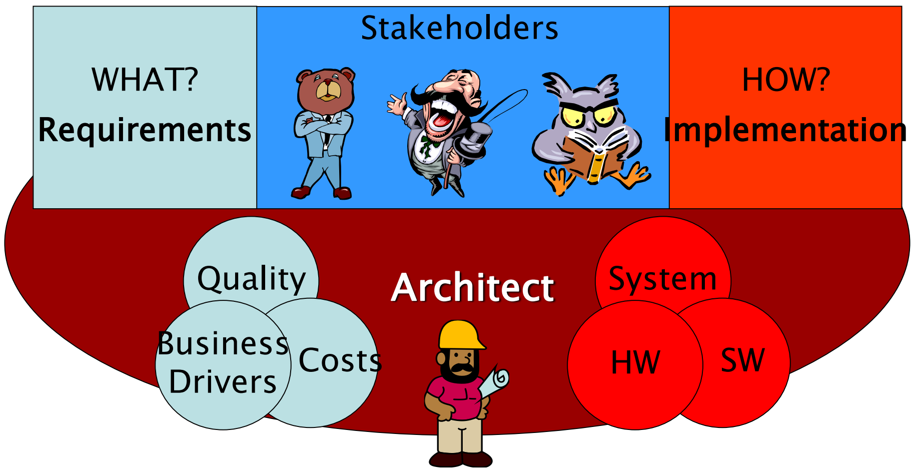
No ideal solution



Discovery may be exploratory

There is no ideal system to be discovered.

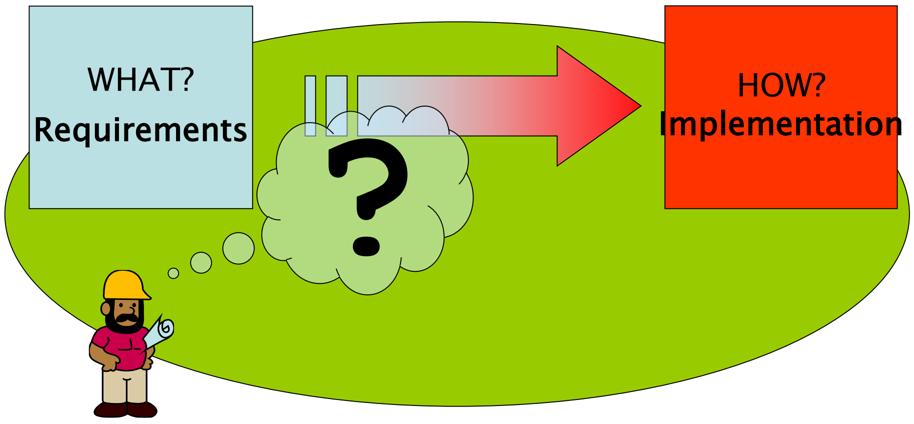
Process: Working Together

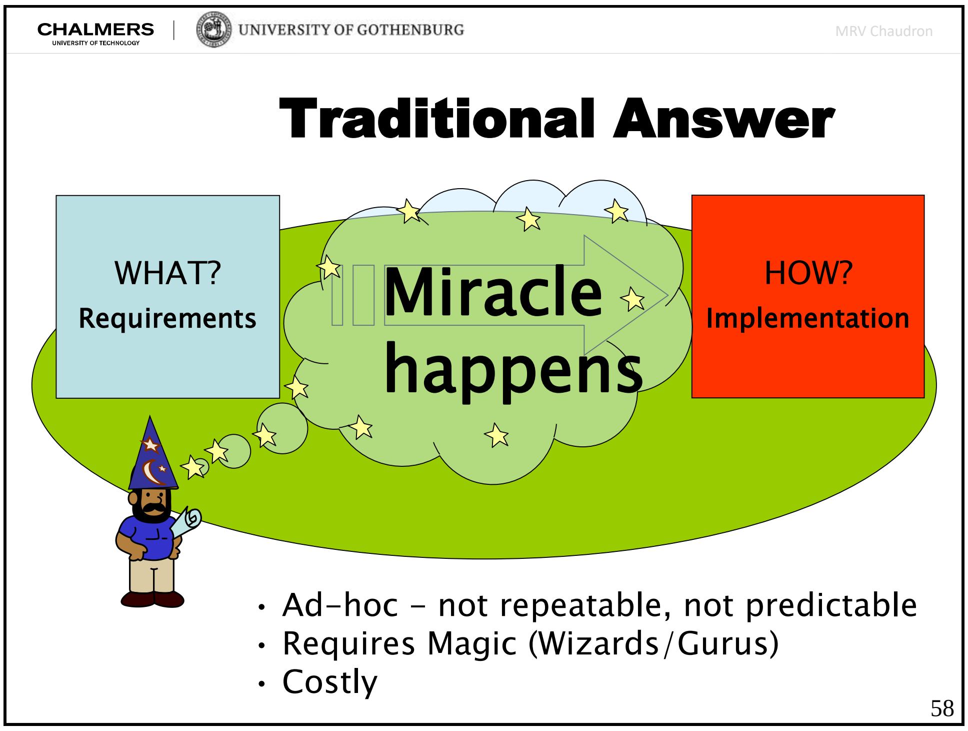


Close and effective interaction between these actors is essential!

Make process transparent: Get/Give *feedback* early and often

How to Bridge the Gap?





CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

Traditional Answer

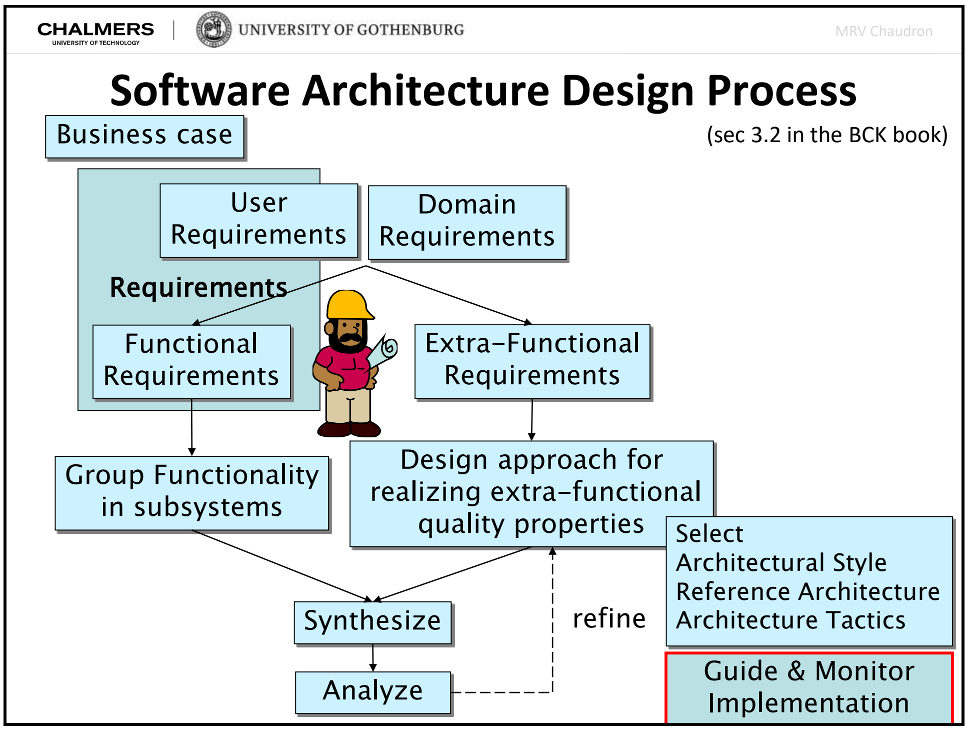
WHAT?
Requirements

Miracle happens

HOW?
Implementation

- Ad-hoc – not repeatable, not predictable
- Requires Magic (Wizards/Gurus)
- Costly

58



CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

Software Architecture Design Process

(sec 3.2 in the BCK book)

Business case

Requirements

User Requirements

Domain Requirements

Functional Requirements

Extra-Functional Requirements

Group Functionality in subsystems

Design approach for realizing extra-functional quality properties

Select
Architectural Style
Reference Architecture
Architecture Tactics

Synthesize

Analyze

refine

Guide & Monitor Implementation

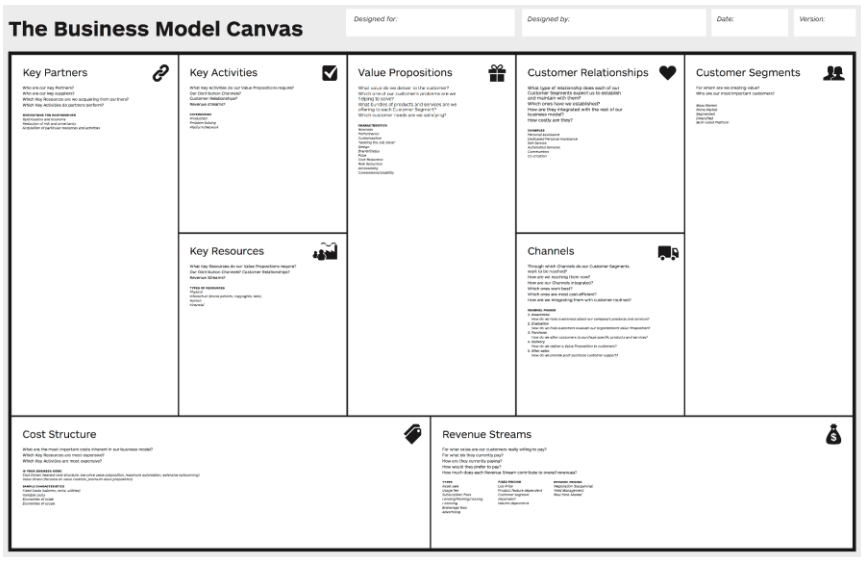
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

Business Case

- Will benefits outway costs?
- How much does the product cost
  - To develop
  - & to maintain!
- What is the time-to-market of the system?
- Market: Who are the customers?
  - How many? What will they pay?

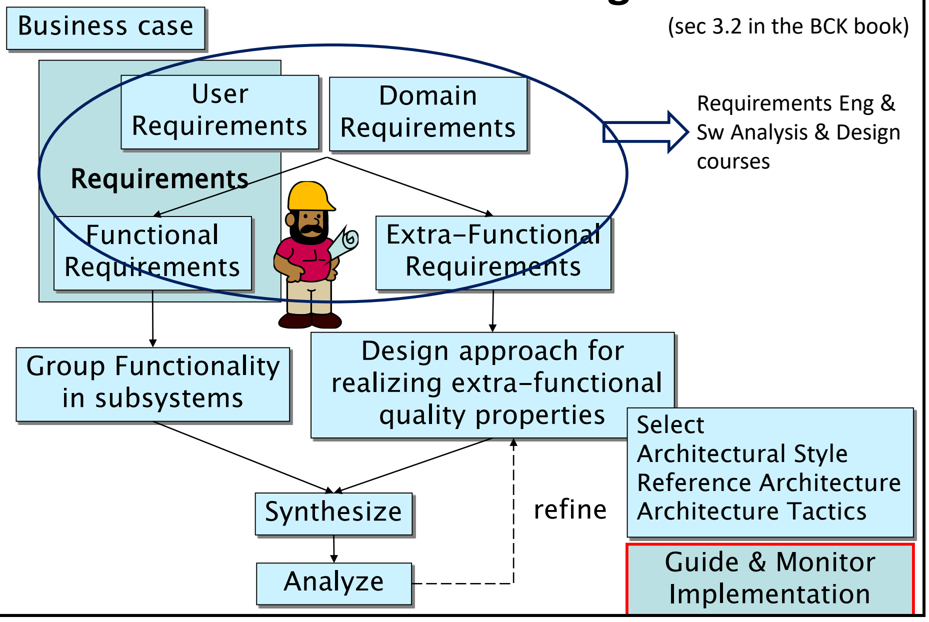
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

Business Model Canvas



Software Architecture Design Process

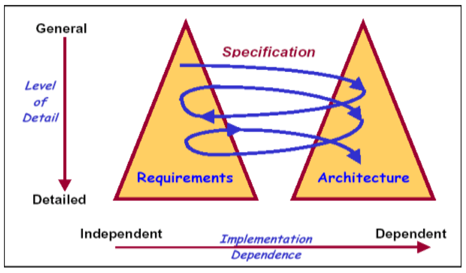
(sec 3.2 in the BCK book)


Business case
User Requirements
Domain Requirements
Requirements
Functional Requirements
Extra-Functional Requirements
Requirements Eng & Sw Analysis & Design courses
Group Functionality in subsystems
Design approach for realizing extra-functional quality properties
Select Architectural Style Reference Architecture Architecture Tactics
Synthesize
refine
Analyze
Guide & Monitor Implementation

Twin Peaks Process

Separate but concurrent development of requirements & architecture

WHAT:
problem
structuring

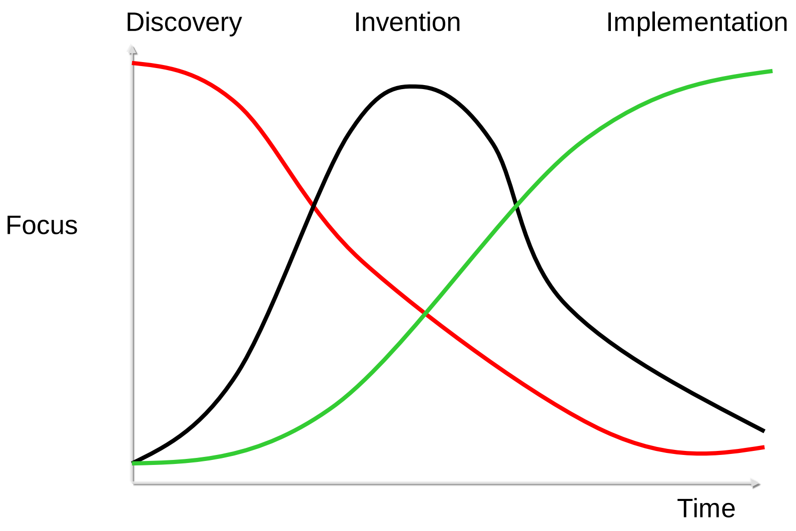


HOW:
solution
structuring

Progressing understanding of *architecture & design* provides a basis for discovering further *problem space & requirements* and vice versa.

There is interaction between available solutions and requirements

Focus over time



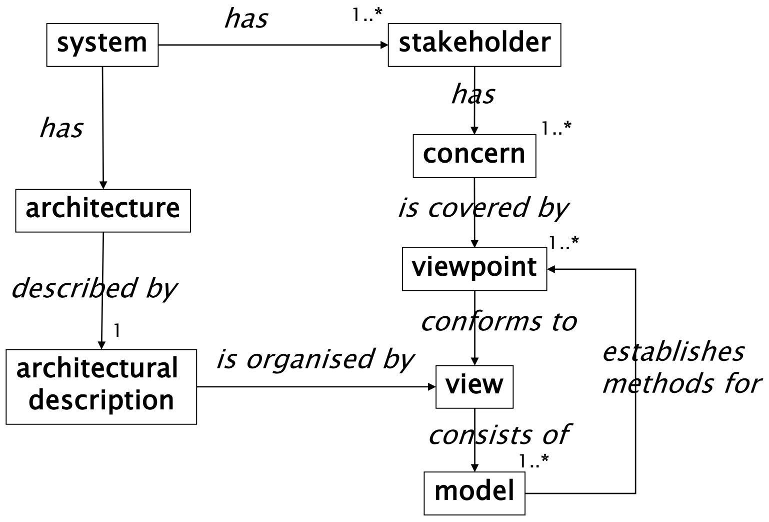
From : Bran Selic

Outline

- What is Software Architecture?
- Stakeholders
- How Software Architecting?
- 4+1 Views
- Concluding Remarks & References

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

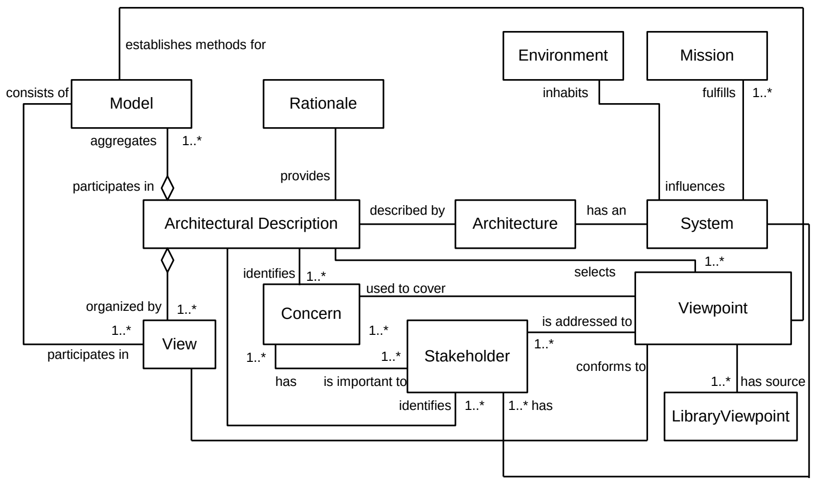
Overview (According to IEEE 1471)



69

CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG MRV Chaudron

ISO/IEC/IEEE 42010:2011 Conceptual Framework

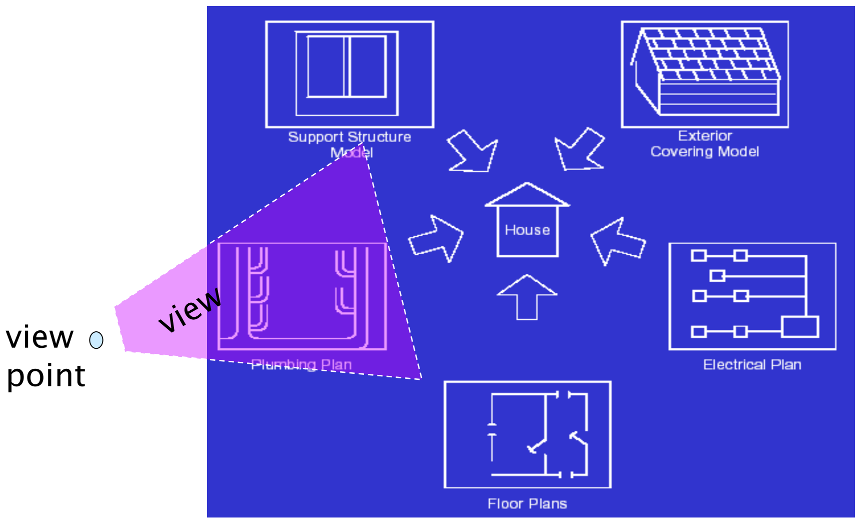


7

Outline

- What is Software Architecture?
- Stakeholders
- How Software Architecting?
- 4+1 Views
- Summary

Viewpoints & views



View: Definition (from IEEE 1471)

3.4 Architectural Description (AD): A collection of products to document an architecture.

3.9 View: A representation of a whole system from the perspective of a related set of concerns.

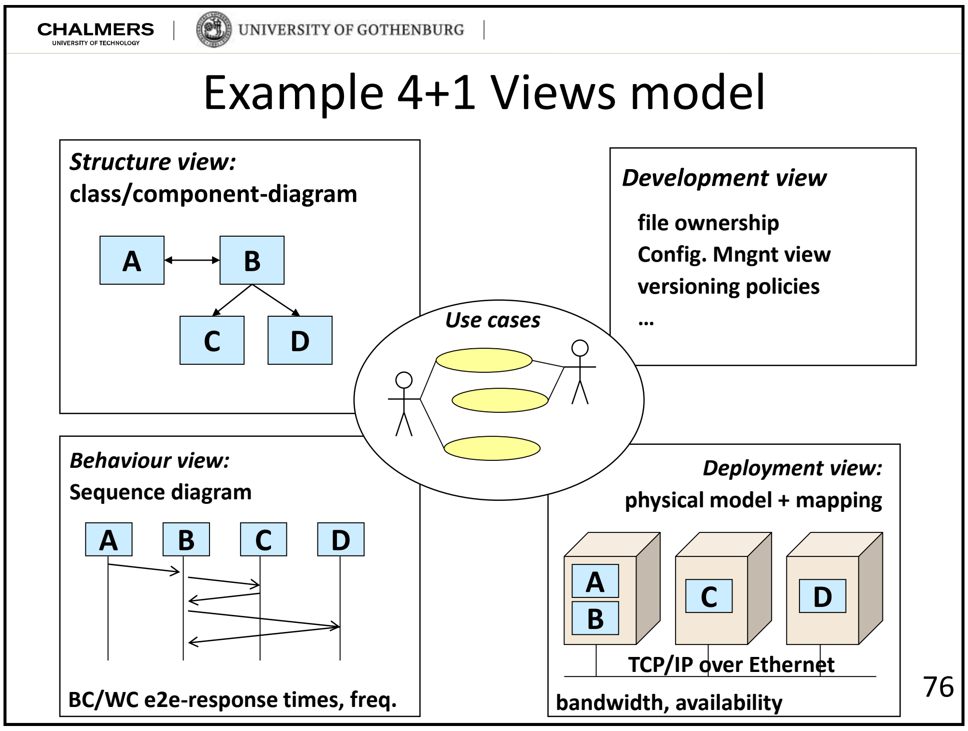
A view may consist of one or more *architectural models*

Each such architectural model is developed using the methods established by its associated architectural viewpoint.

An architectural model may participate in more than one view.

Architectural view

- An architectural view is a simplified description (an abstraction) of a system from a particular perspective/view point, covering particular concerns, and omitting entities that are not relevant to this perspective



CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Example 4+1 Views model

Structure view:
class/component-diagram

Development view
file ownership
Config. Mngnt view
versioning policies
…

Use cases

Behaviour view:
Sequence diagram

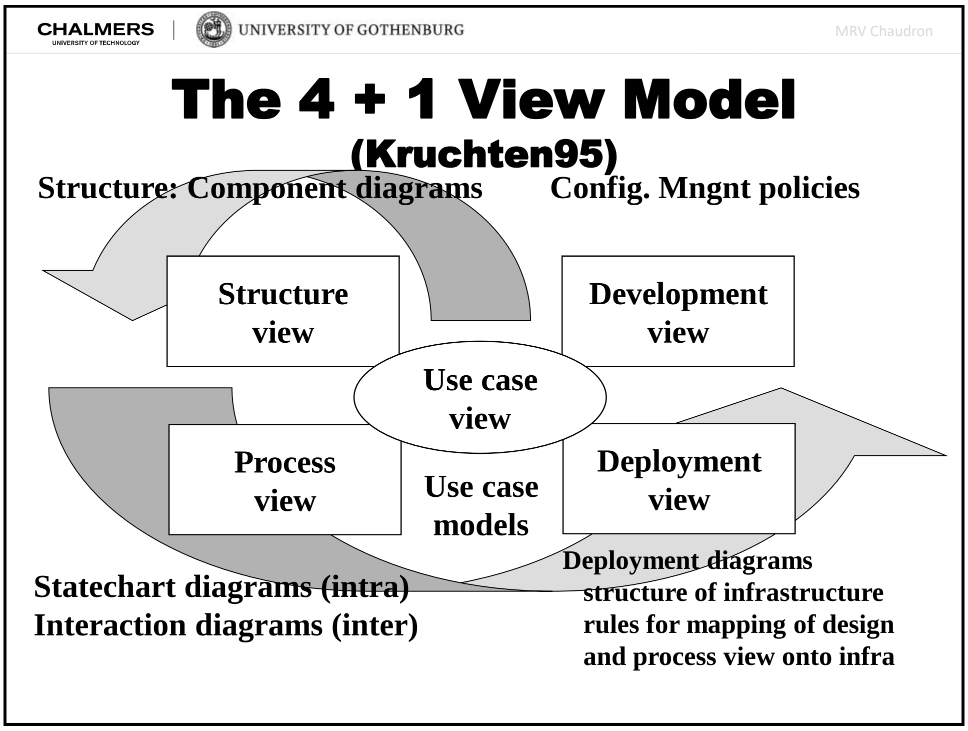
BC/WC e2e-response times, freq.

Deployment view:
physical model + mapping

TCP/IP over Ethernet

bandwidth, availability

76



CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

MRV Chaudron

The 4 + 1 View Model (Kruchten95)

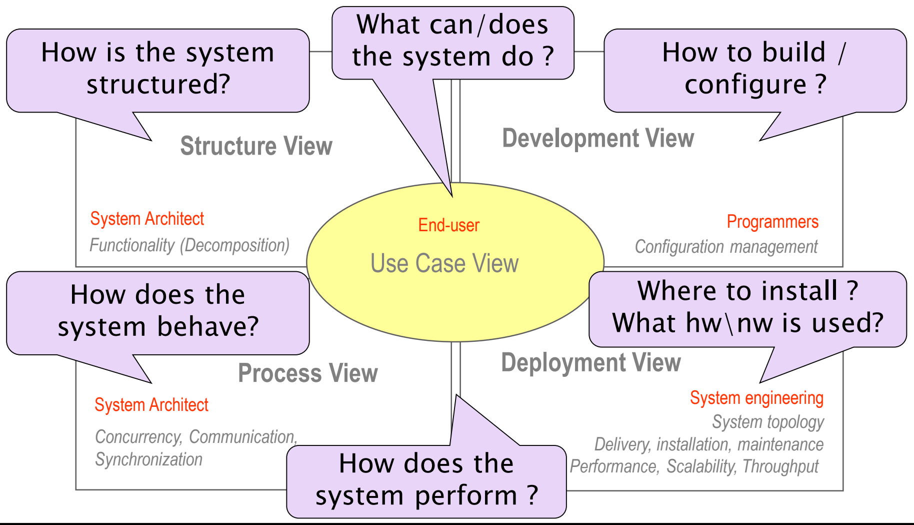
Structure: Component diagrams

Config. Mngnt policies

Statechart diagrams (intra)
Interaction diagrams (inter)

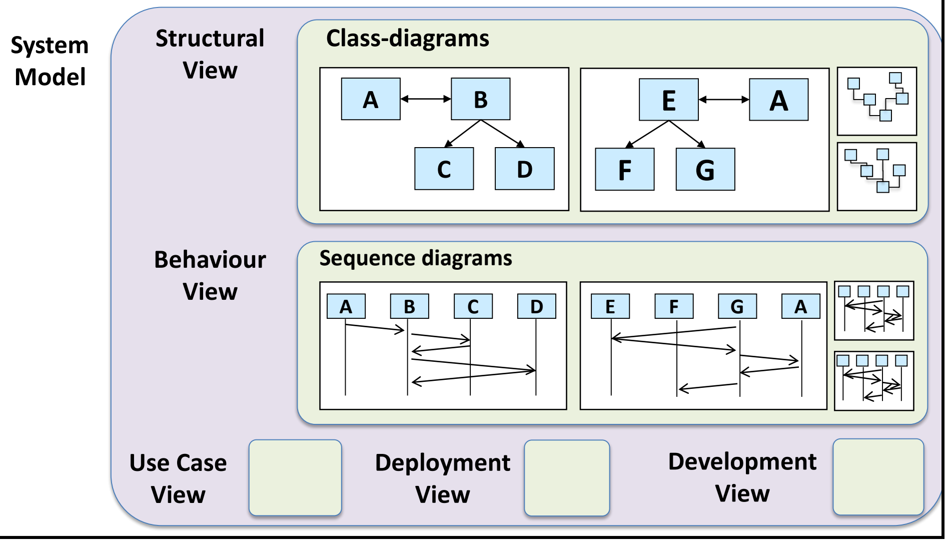
Deployment diagrams
structure of infrastructure
rules for mapping of design
and process view onto infra

4+1 Views Representation of System Architecture


How is the system structured?
What can/does the system do ?
How to build / configure ?
Structure View
Development View
System Architect
Functionality (Decomposition)
End-user
Use Case View
Programmers
Configuration management
How does the system behave?
Where to install ? What hw\nw is used?
Process View
Deployment View
System Architect
Concurrency, Communication, Synchronization
System engineering
System topology
Delivery, installation, maintenance
Performance, Scalability, Throughput
How does the system perform ?

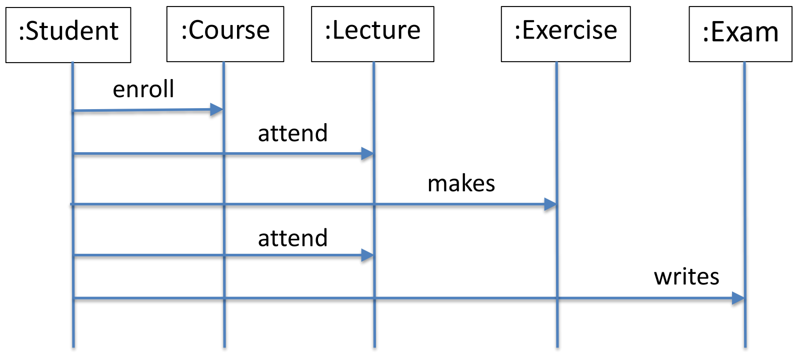
1 Model = union of multiple views

each view has one or more diagrams



Behaviour View!

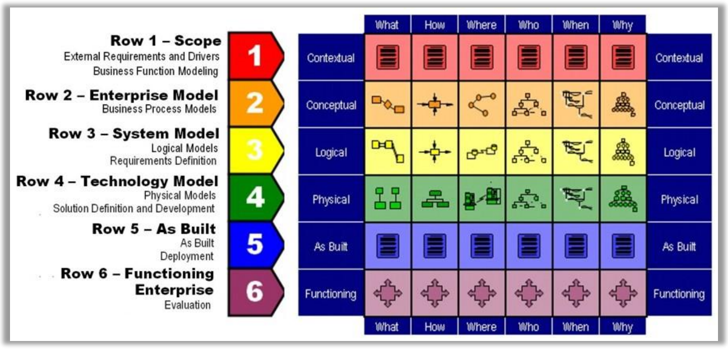
Most illustrations of software architecture us structural views, but the behavioural views as just as important!



Other modeling languages can be used for describing the behaviour(e.g. activity diagrams)

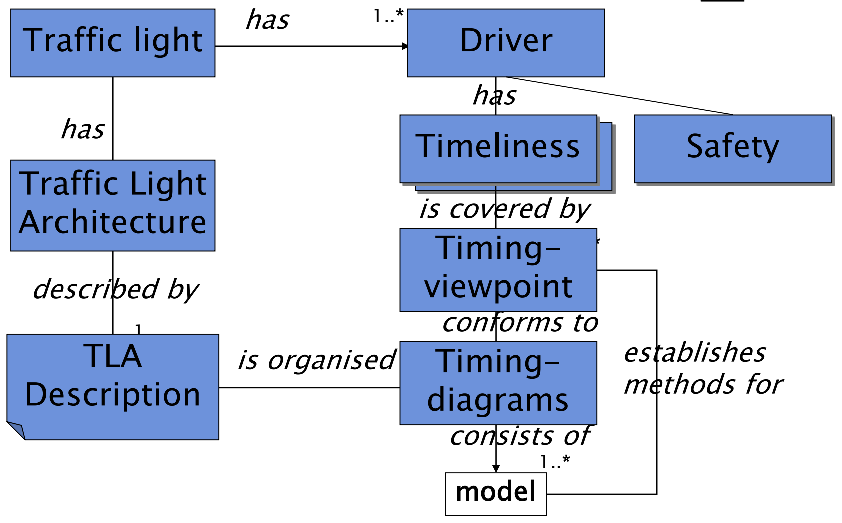
Other 'Views'-paradigms exist

- Soni-and-Nord (4-views, Siemens)
- Zachman (36-views, IBM)
  - Mostly for Enterprise Architecture

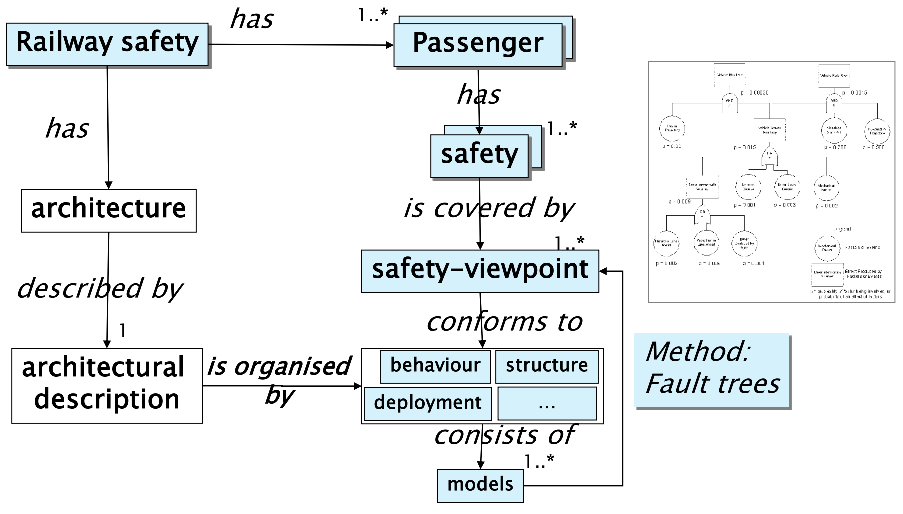


https://en.wikipedia.org/wiki/Zachman_Framework

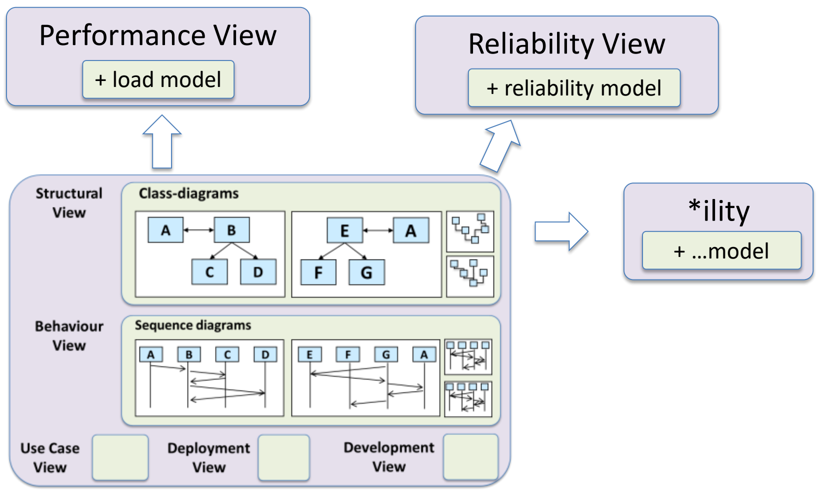
Overview - example



Example (According to IEEE 1471)

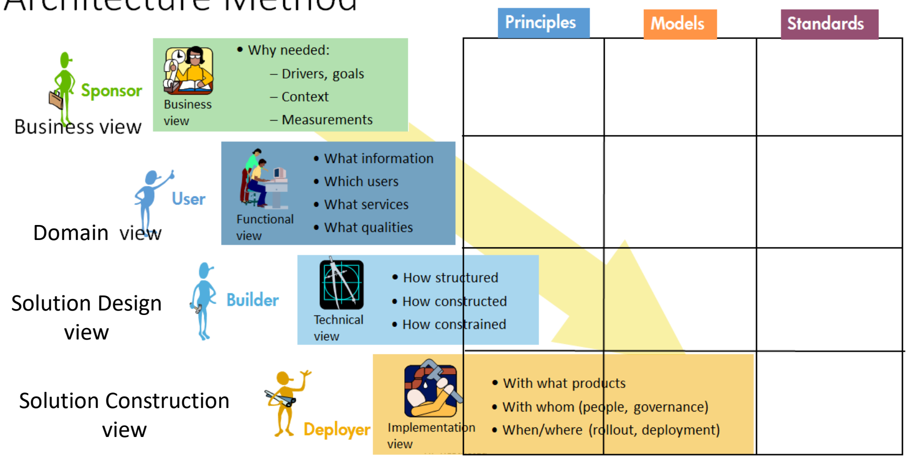


Views for Extra-Functional Properties



Additional views can sometimes be generated from the ‘basic’ views.
Benefits are: reduced effort & up-to-date- & consistent views

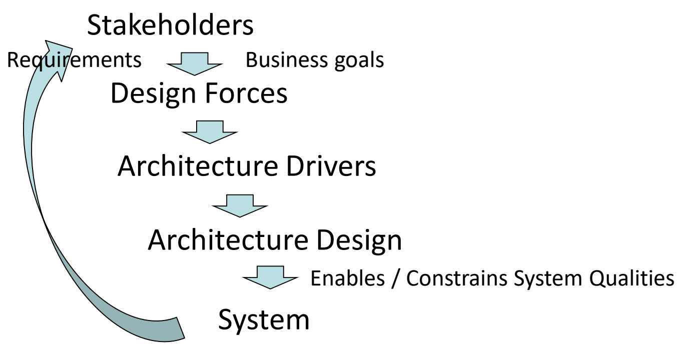
Architecture Method



Discussion

- Why should we use different diagrams?
- Why should we use different views?
- What is the relation between 'forces' and 'qualities'?

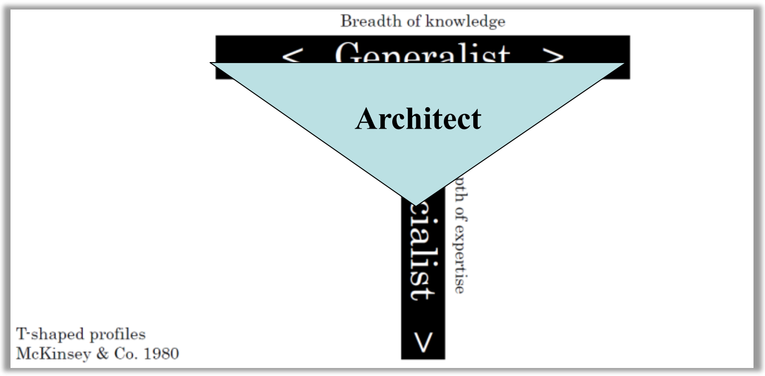
Summary - 1



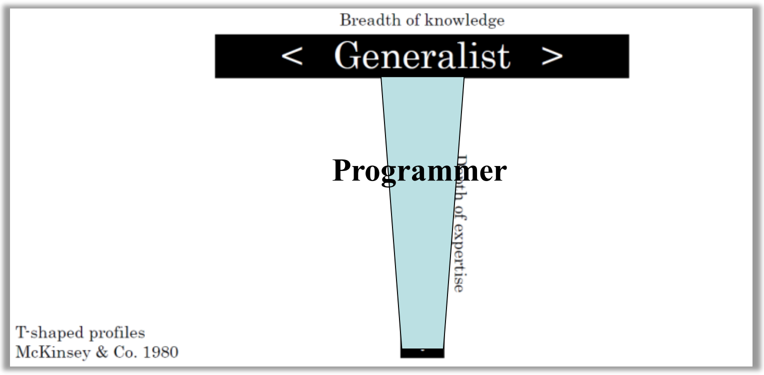
Summary - 2

- Architecture Design process
  - Iterative
  - Feedback early and often
- Architecture Description
  - Multiple concerns => multiple views (e.g. 4 + 1)
  - Include Design Rationale

Knowledge Profile



Knowledge Profile



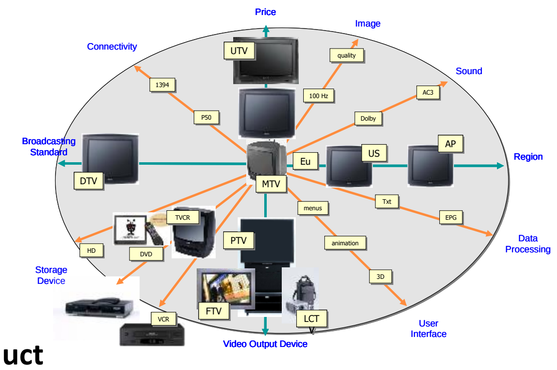
Disclaimer

No matter how good your architecture, it can always be ruined by bad implementation!

Is this a reason for not doing architecture?

Guide & Monitor Implementation !

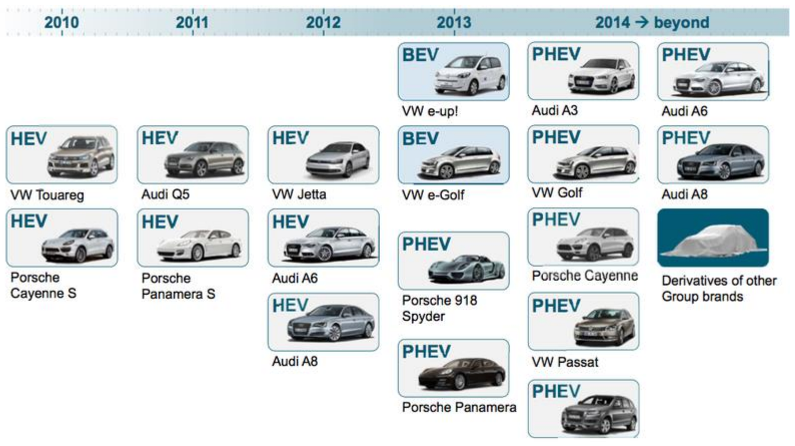
Types of Architectures



- Single **product**
  → Future-proof with respect to a particular product
- Product **family –** e.g. TVs / Telephones
  → Instantiation of many *related products*
  → Stable with respect to a restricted set of Quality attributes
  → Future-proof with respect to a product type

Time/Product Scope

Automotive Product Line


2014 → beyond
BEV VW e-up!
PHEV Audi A3
PHEV Audi A6
HEV VW Touareg
HEV Audi Q5
HEV VW Jetta
BEV VW e-Golf
PHEV VW Golf
PHEV Audi A8
HEV Porsche Cayenne S
HEV Porsche Panamera S
HEV Audi A6
PHEV Porsche 918 Spyder
PHEV Porsche Cayenne
Derivatives of other Group brands
HEV Audi A8
PHEV VW Passat
PHEV Porsche Panamera
PHEV