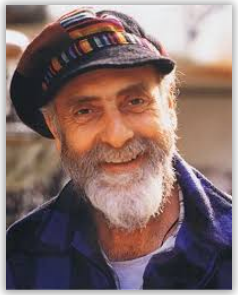
Software Architecture DAT220/DIT544

Truong Ho-Quang
truongh@chalmers.se
Software Engineering Division
Chalmers | GU

Standardization

- **Friedrich Stowasser** (1928 – 2000)



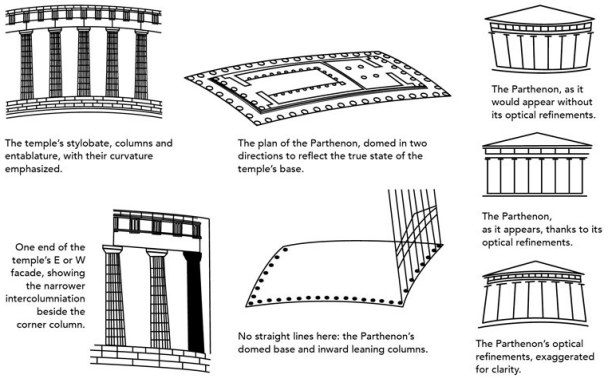
- He was an opponent of "a straight line"

  "Die gerade Linie ist gottlos und unmoralisch."

  (from Hundertwassers' manifest against Rationalism in Architecture, 1958)

- What does this do to
  - Reuse? Complexity?
  - Patterns?
  - Mass-production? Efficiency?

No straight lines - more examples ...



Past

Future

Schedule

Week		Date	Time	Lecture	Note
3	L1	Wed, 20 Jan	10:15 – 12:00	Introduction & Organization	Truong Ho
3	L2	Thu, 21 Jan	13:15 – 15:00	Architecting Process & Views	Truong Ho
4		Tue, 26 Jan	10:15 – 12:00	**Skip**	
4	S1	Wed, 27 Jan	10:15 – 12:00	<< Supervision: Launch Assignment 1>>	TAs
4	L3	Thu, 28 Jan	13:15 - 15:00	Roles/Responsibilities & Functional Decomposition	Truong Ho
5	L4	Mon, 1 Feb	10:15 – 12:00	Architectural Styles P1	Truong Ho
5	S2	Wed, 3 Jan	10:15 – 12:00	<< Supervision/Assignment>>	TAs
5	L5	Thu, 4 Jan	13:15 – 15:00	Architectural Styles P2	Truong Ho
6	L6	Mon, 8 Feb	10:15 – 12:00	Architectural Styles P3	Sam Jobara
6	S3	Wed, 10 Feb	13:15 – 15:00	<< Supervision/Assignment>>	TAs
6	L7	Thu, 11 Feb	13:15 – 15:00	Design Principles (Maintainability, Modifiability)	Truong Ho
7	L8	Mon, 15 Feb	10:15 – 12:00	Performance – Analysis & Tactics	Truong Ho
7	S4	Wed, 17 Feb	13:15 – 15:00	<< Supervision/Assignment>>	TAs
7	L9	Thu, 18 Feb	10:15 – 12:00	Tactics: Reliability, Availability, Fault Tolerance	TBD
8	L10	Mon, 22 Feb	13:15 – 15:00	Guest Lecture 1	TBD
8	S5	Wed, 24 Feb	13:15 – 15:00	<< Supervision/Assignment>>	TAs
8	L11	Thu, 25 Feb	10:15 – 12:00	Guest Lecture 2	TBD
9	L12	Mon, 1 Mar	13:15 – 15:00	Reverse Engineering & Correspondence	Truong Ho
9	S6	Wed, 3 Mar	10:15 – 12:00	<< Supervision/Assignment>>	TAs
9	L13	Thu, 4 Mar	13:15 – 15:00	To be determined (exam practice?)	Truong Ho
9		Fri, 5 Mar	Whole day	Group presentation of Assignment (TBD)	Teachers
11	Exam				4

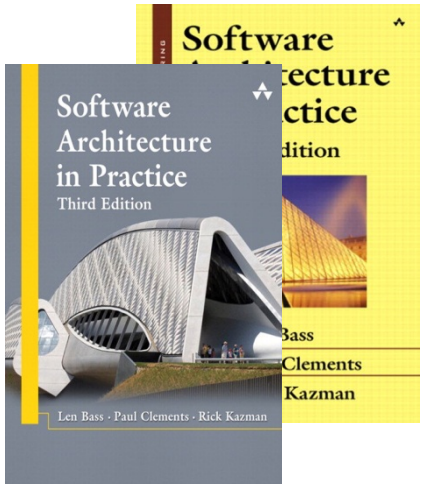
Group Formation!

- Expectation: 8 groups (40 students)
- Current: 1 group formed, 35 students left unassigned to a group
  - Updates in this page: https://chalmers.instructure.com/courses/12514/pages/group-formation-updates
- Deadline: January 25.
- A supervisor will be assigned to every group

FAQs

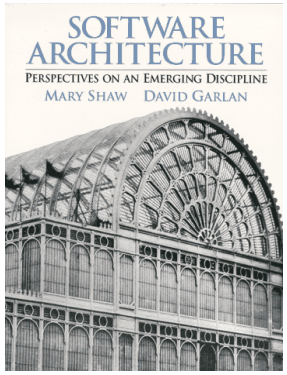
- FAQs are available at this page: https://chalmers.instructure.com/courses/12514/pages/faqs

Software Architecture Books



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

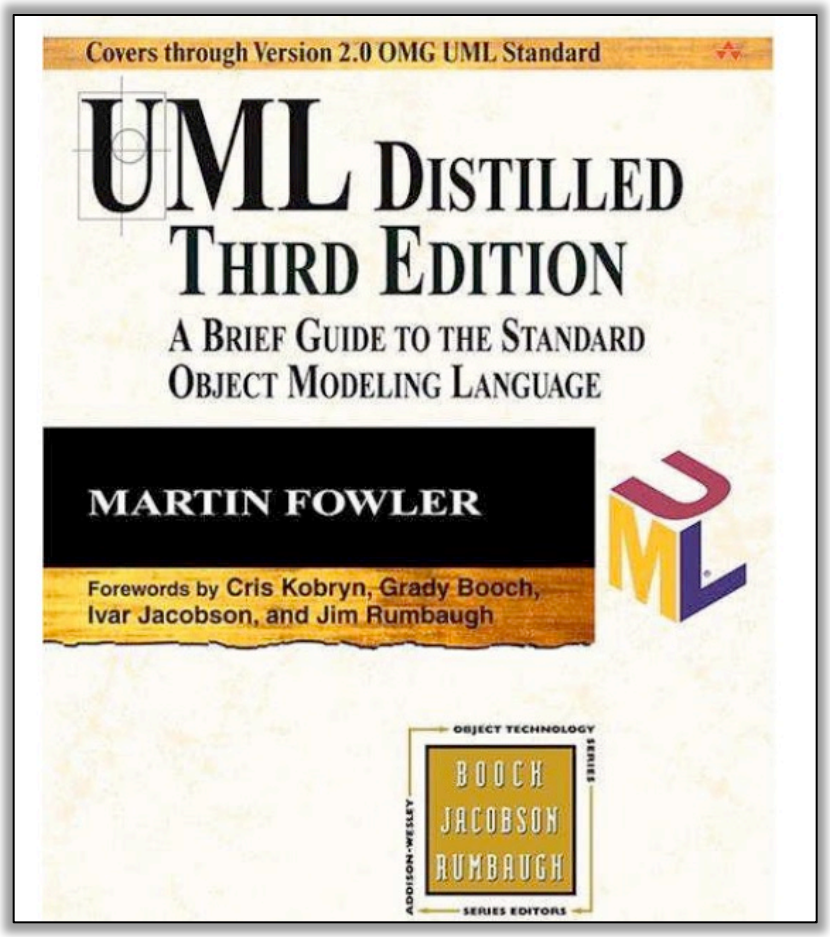
- Software Architecture: Perspectives on an Emerging Discipline, Mary Shaw, David Garlan, 242 pages, 1996, Prentice Hall



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

UML book

- UML Distilled
  4th or 3rd edition



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

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

All of the above are valid!

- Add your own definition:
  http://www.sei.cmu.edu/architecture/start/glossary/community.cfm

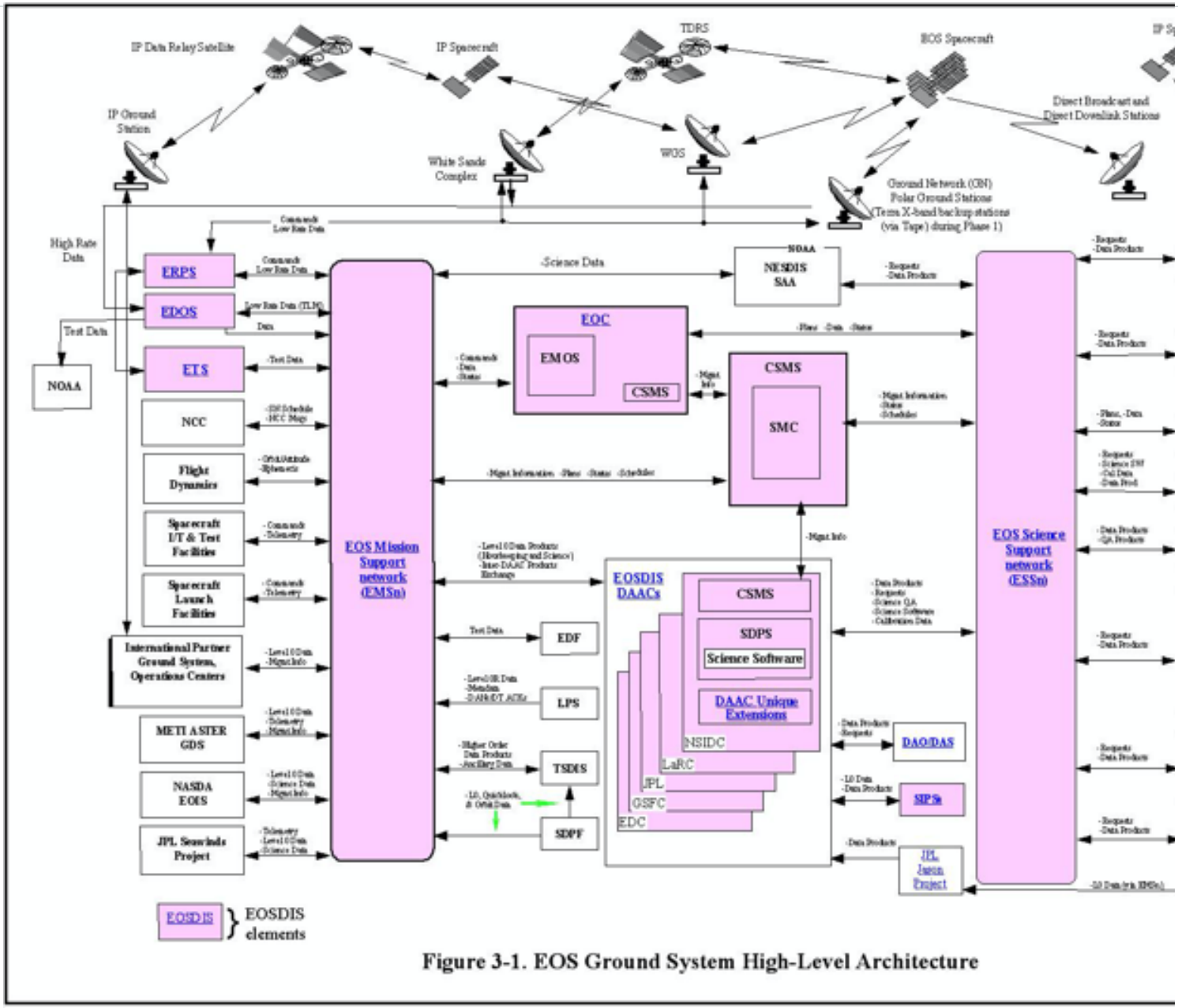


Figure 3-1. EOS Ground System High-Level Architecture

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:

- 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

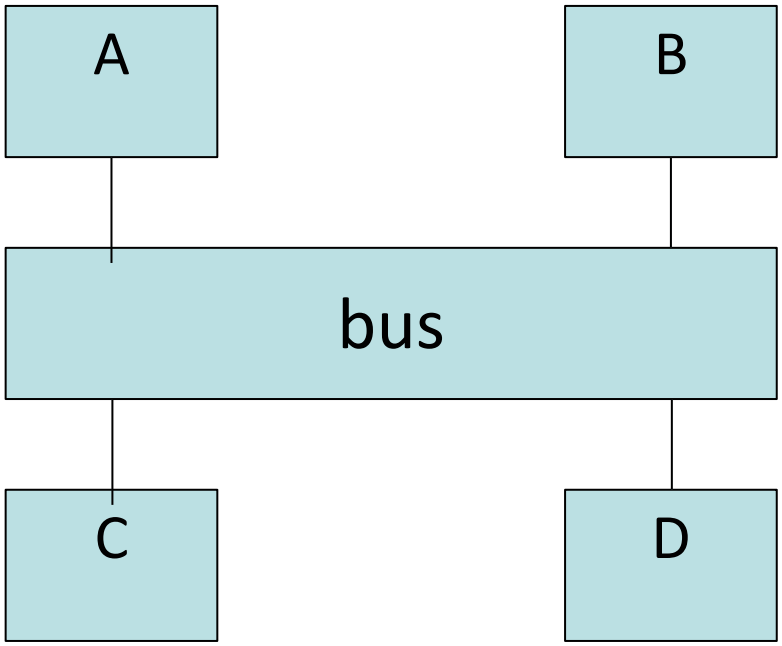
Often implicit: arrow means ‘request-response’

Many alternatives possible:

fire & forget, blackboard, publish/subscribe, …

More interaction patterns:
https://www.enterpriseintegrationpatterns.com/patterns/conversation/BasicIntro.html

Connector example



Architecture Model with explicit connectors

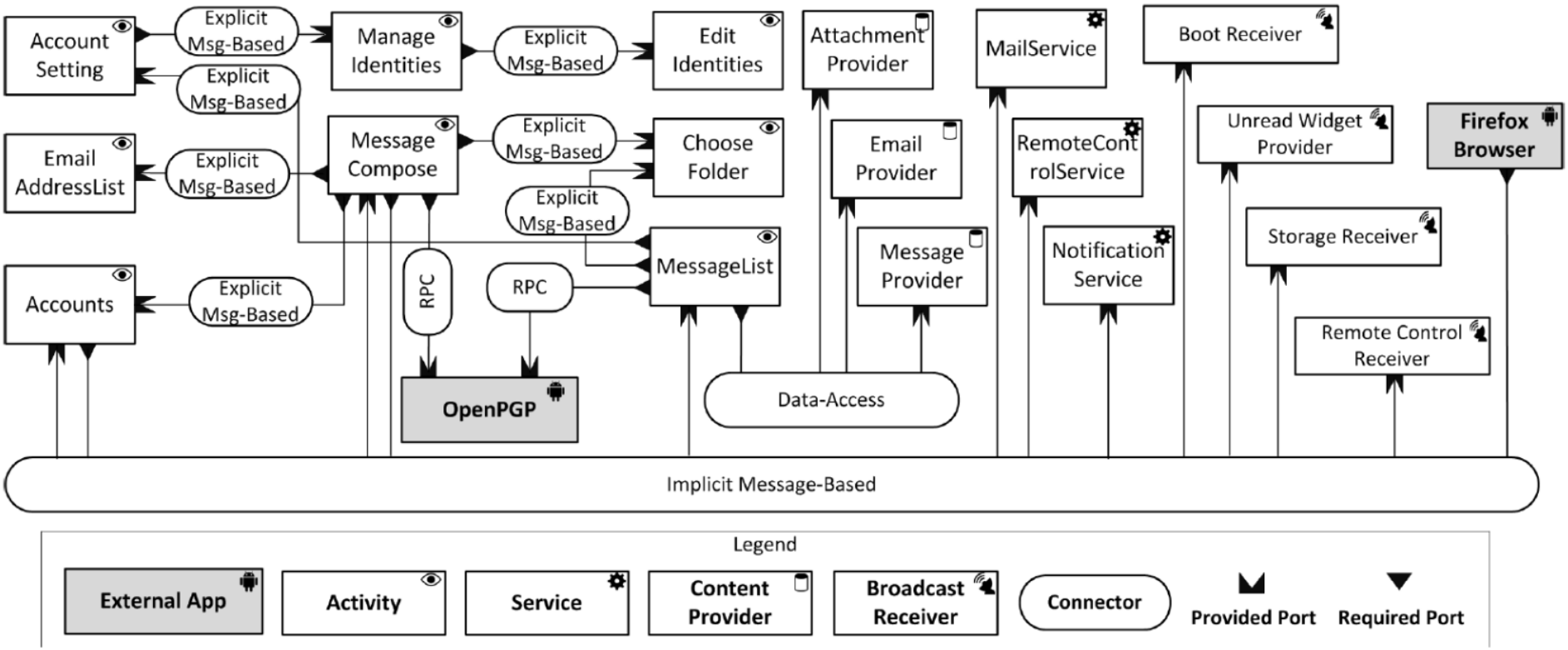


Figure 2: K-9 mail Android app architecture

Why, When and for Whom?

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

Multiple Purposes of Architecture

Understanding + Analyzing
+ Communicating + Constructing

Why is the system needed? What constraints apply?

Understanding the requirements

What are the important design decisions

What functions does the system provide?

What properties does the design have?

How can the system be built?

Figure from Gerrit Muller, How to Create a Manageable Platform Architecture

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

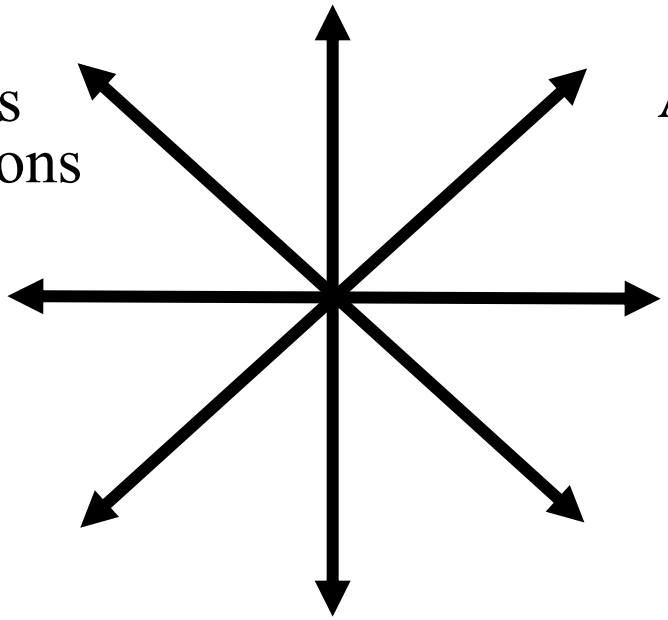
- And some are ***not*** observable via execution, but in the development process: modifiability, portability, reusability, integrability, testability

Architecting = Balancing Objectives

Functional dimensions

Structure
Functionality
Component interfaces
Component configurations

Local/Global Behavior
Control modes
Activities/Transactions
C&S protocols



Dependability
Performance
Timeliness
Reliability
Availability
Safety
Security
Robustness

General
Scalability
Flexibility
Maintainability
Reusability
Openness
User-Friendliness
Costprize
•••

Composability
Interoperability
Reusability

Extra-Functional dimensions

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, …

ISO standard on Software Product Quality

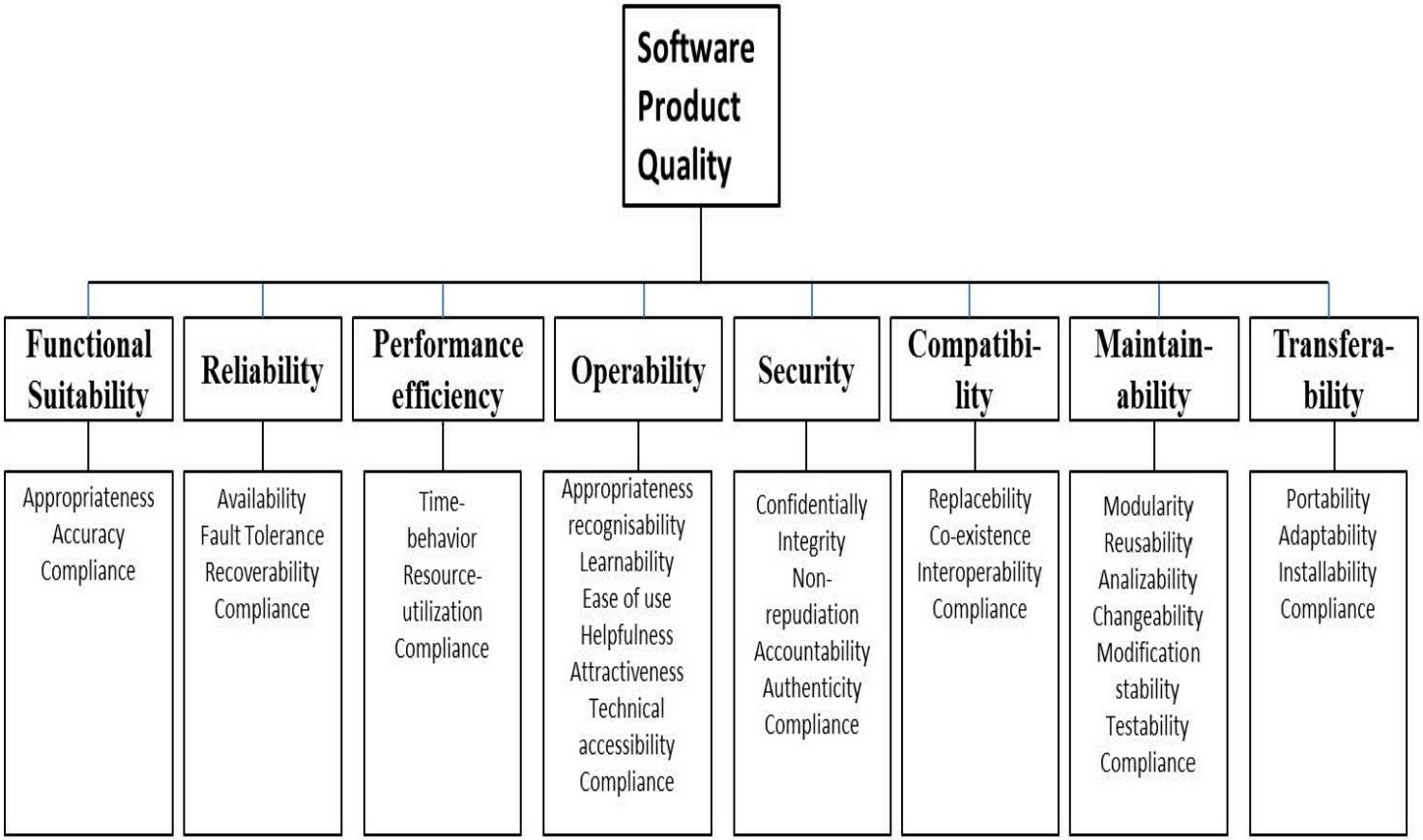
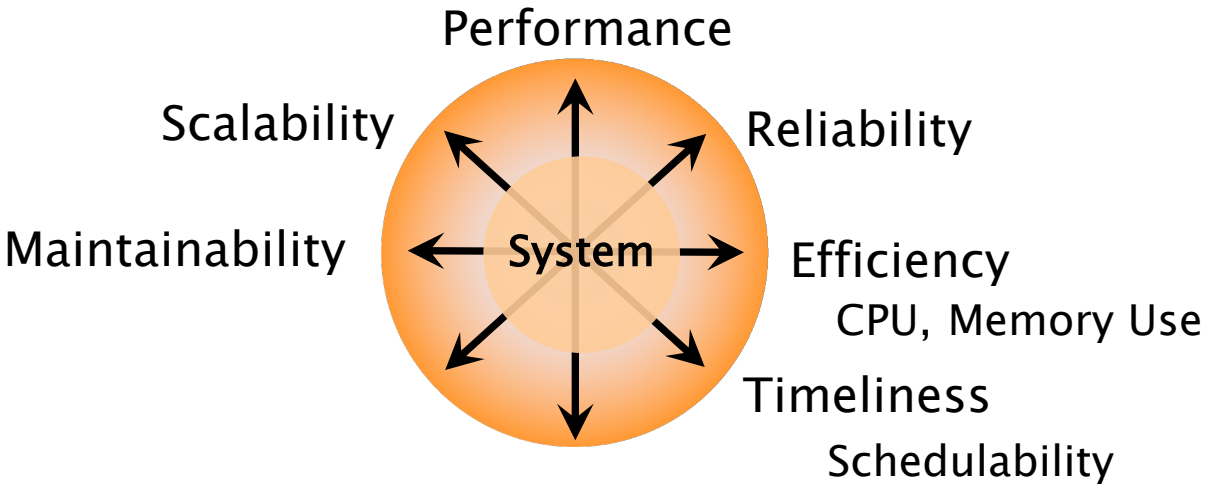


Figure 9 ISO 25010 Model (ISO/ IEC CD 25010 2007)

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.

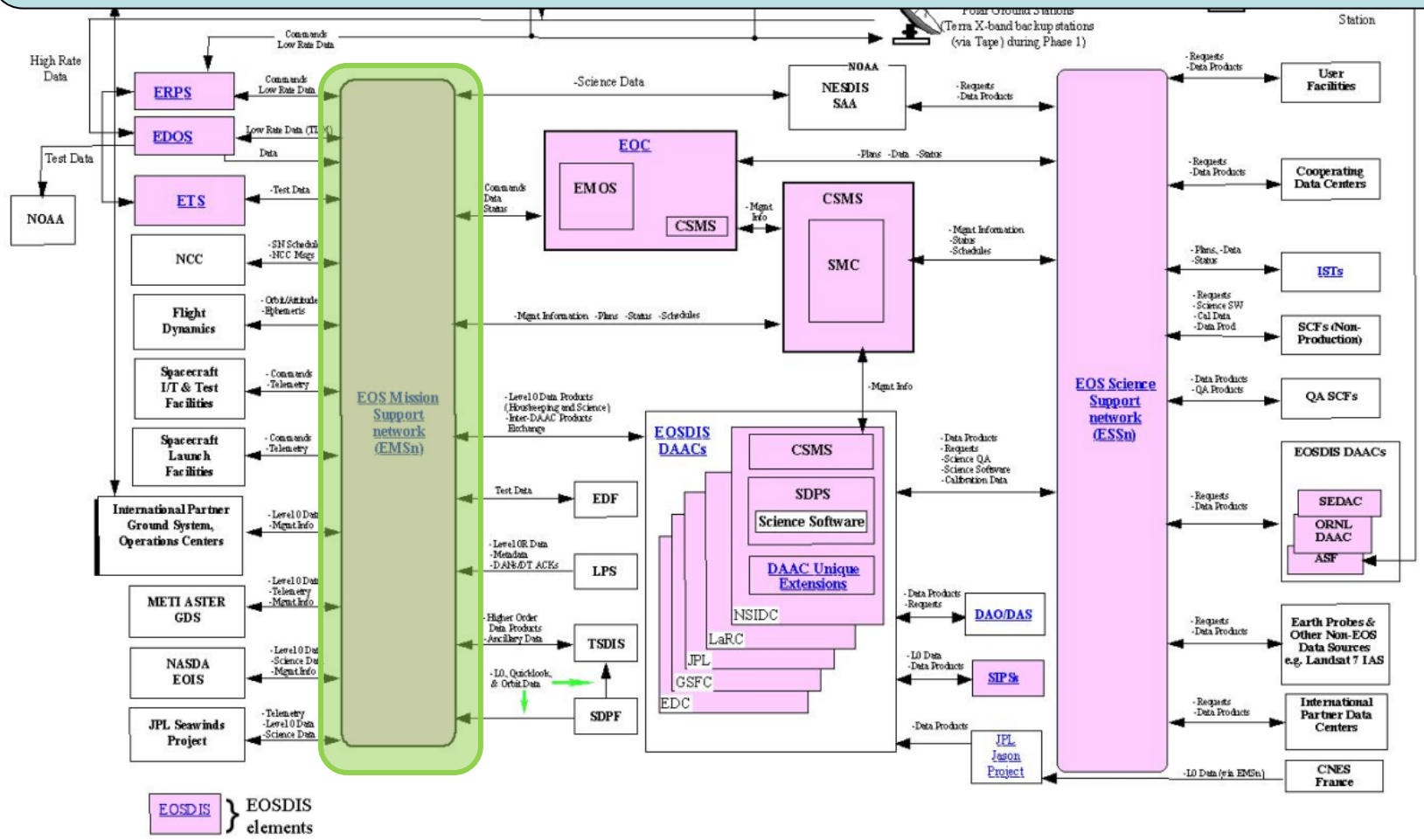


Figure 3-1. EOS Ground System High-Level Architecture

What if we change CSMS?

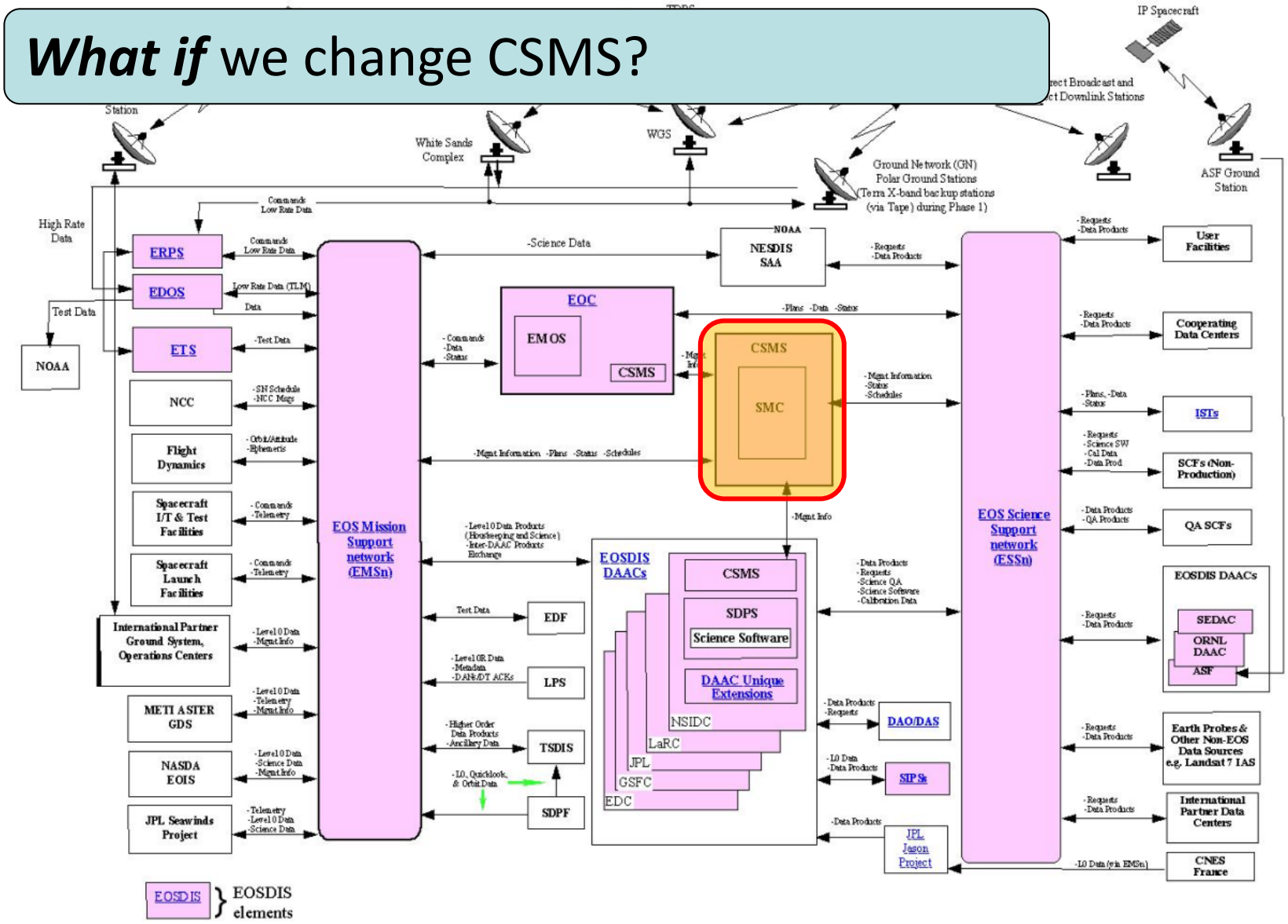


Figure 3-1. EOS Ground System High-Level Architecture

What if we change CSMS?

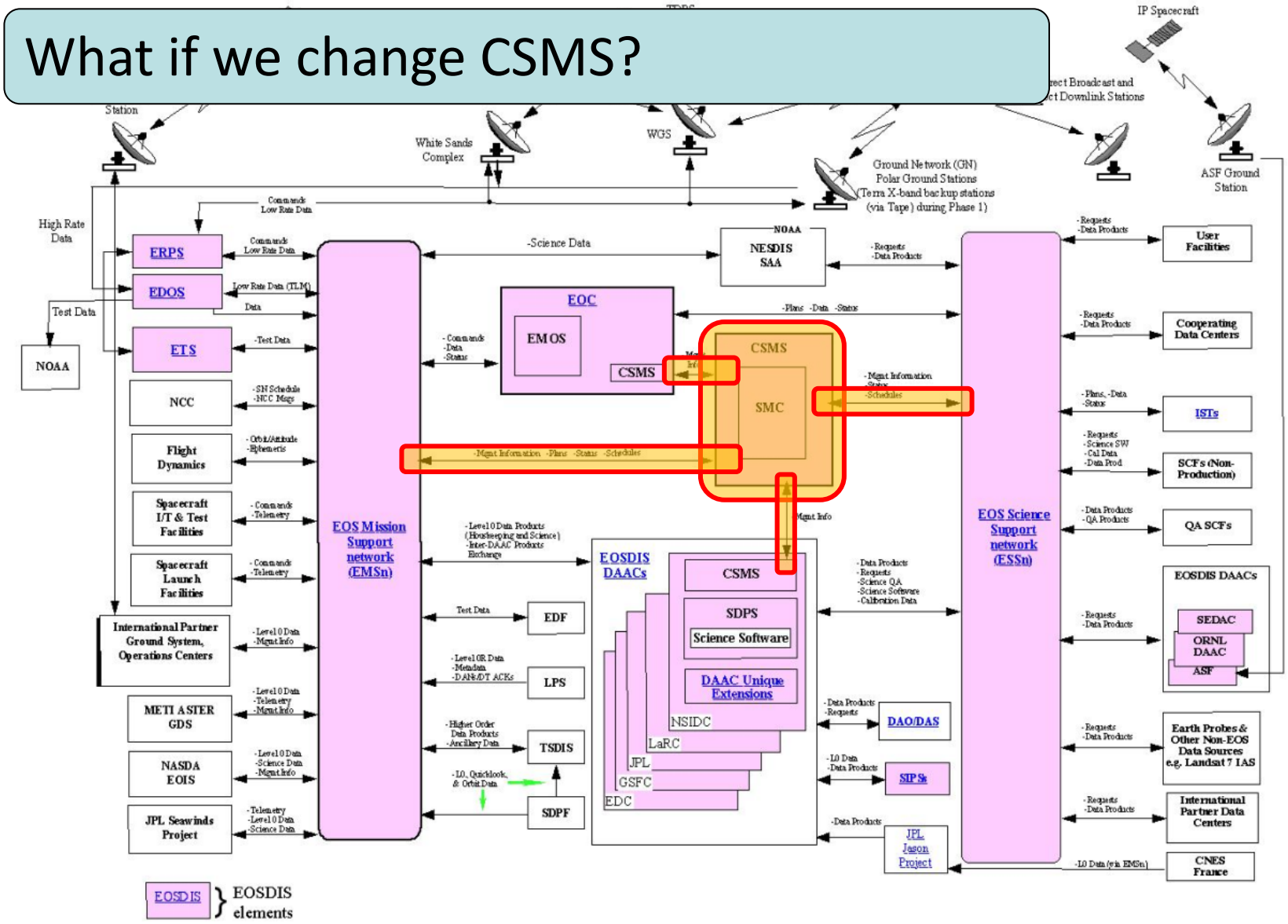


Figure 3-1. EOS Ground System High-Level Architecture

What if we change CSMS?

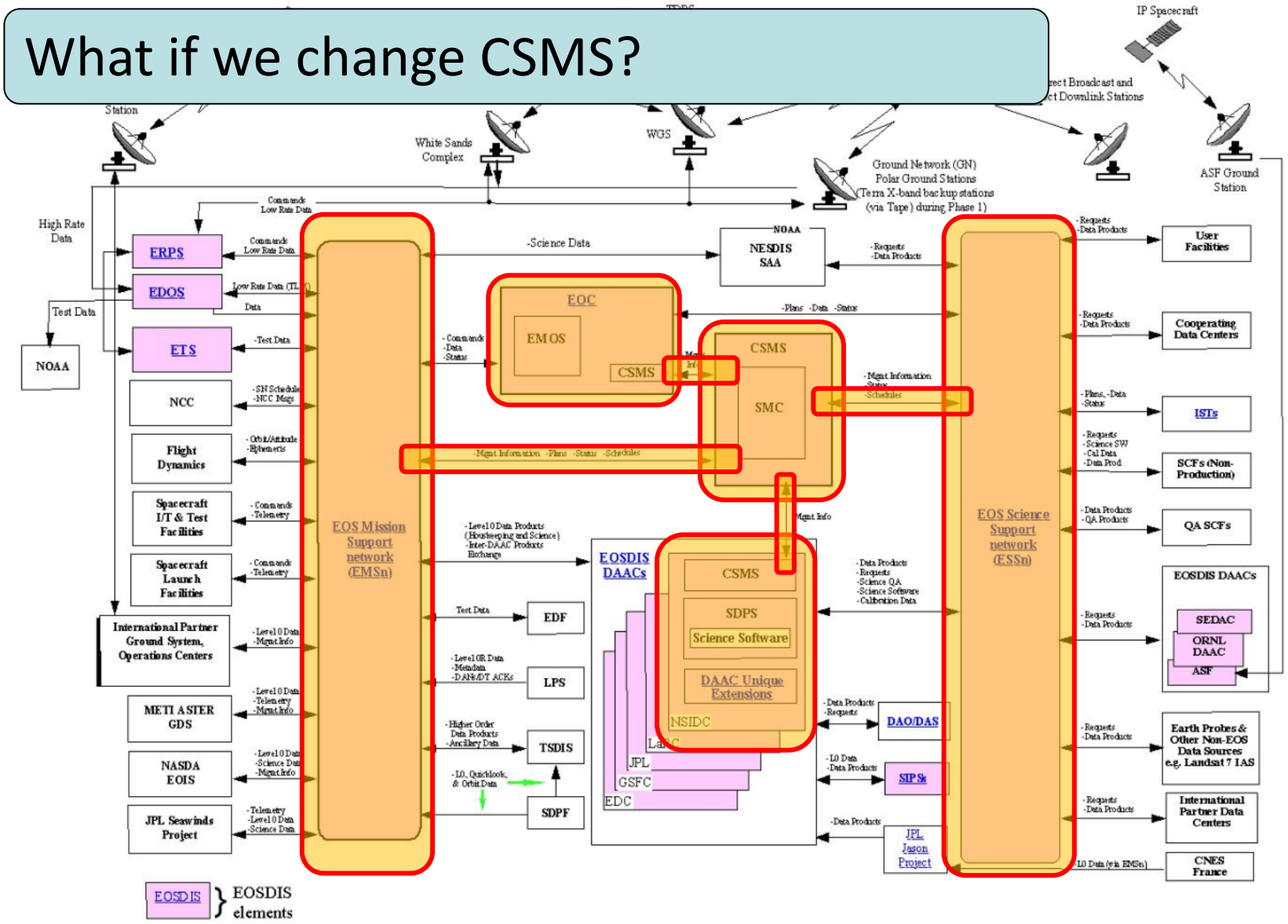


Figure 3-1. EOS Ground System High-Level Architecture

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

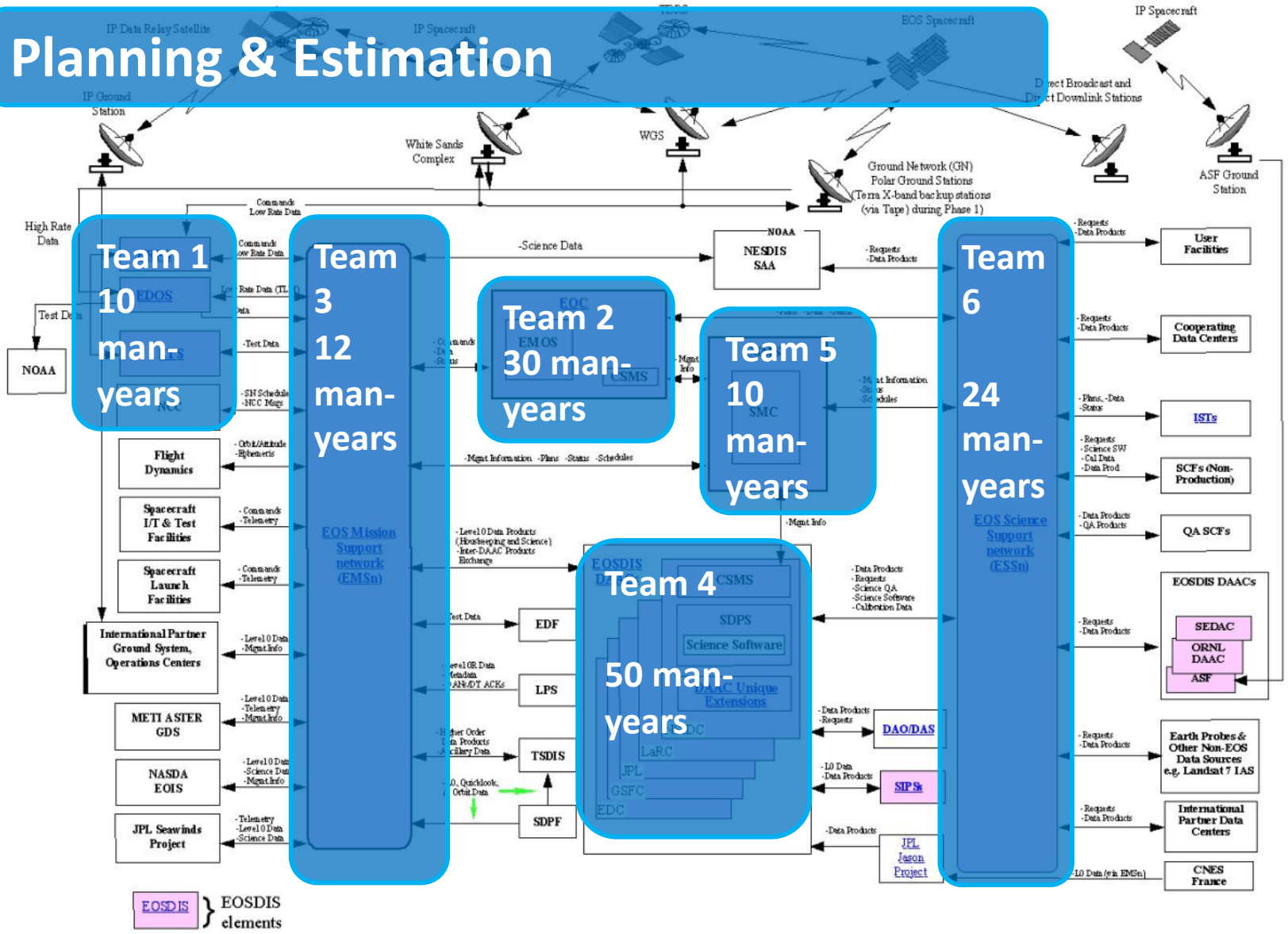


Figure 3-1. EOS Ground System High-Level Architecture

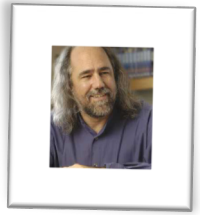
Software Architecting = Designing

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):

Example of forces

- Business **constrains**
  - Time/Schedule
  - Budget
  - Team composition
  - Software licensing restrictions or requirements
- Technical **constrains**
  - Programming language
  - Operating system or platforms supported
  - Use of a specific library or framework

Tip: *"Seperate the constraints you are given from the constraints you give yourself"*

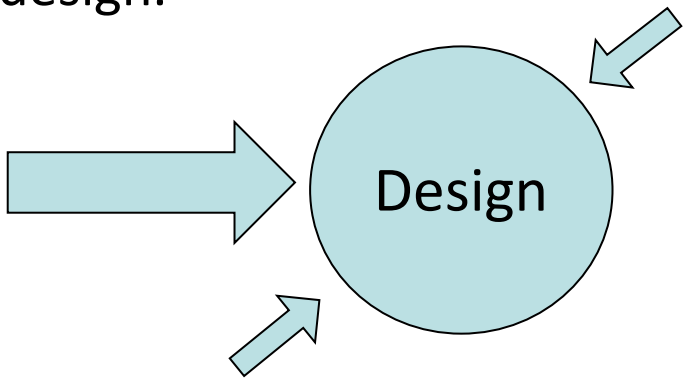
Michael Keeling in this blog post

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.



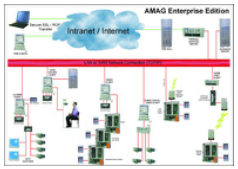
Forces vs Drivers

- There is no clear separation between forces and drivers
- Identification of architectural drivers is very contextual. This often bases on:
  - Architect's experience
    - Pitfalls: Noone knows everything!
  - A thorough architectural reviews/evaluations
    - Business value, architectural impacts

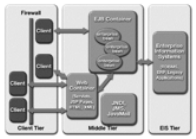
What to keep in mind?

- **<u>Always</u>** mind **what** you are/will be architecting!
  - Input/output
  - What constrains are relevant?

Enterprise architecture

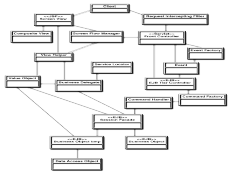


System architecture



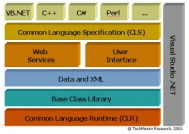
Subsystem

Application architecture



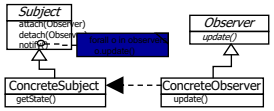
Application

Macro-architecture



Frameworks

Micro-architecture



Design patterns

Is 'cost' an architectural driver?

- This is an 'ultimate' driver to any aspects of software development projects
- 'Cost' affects
  - Functionalities (quality & quantity)
  - Quality of the system
  - Technical choices
- What happens when considering 'cost' in any design decision?
  - As an architect, you cannot decide everything!

- My advice:
  - Cost should be treated in project management level.
  - Ask "stakeholders" to break down the cost-constrains to concrete functional and non-functional constrains (as input for requirements).

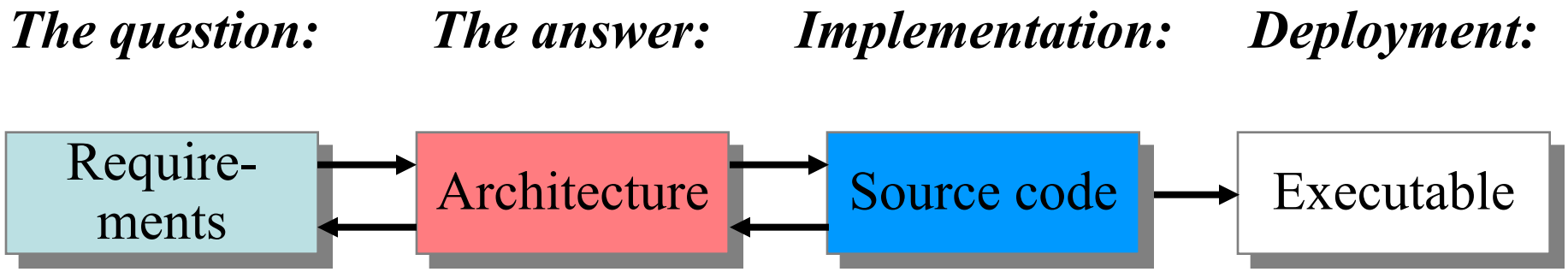
Is 'Quality' an arch. driver?

- YES, but 'Quality' itself is too generic!
  - 'Quality' cannot be measured!
- Quality is often viewed through specific set of quality attributes
- It's important to point out what aspect of quality the software/system should fullfil:
  - Performance
  - Availability
  - Maintainability
  - …

Is 'Functionality' an arch. driver?

- YES, but it is an 'ultimate' driver, too.
- Functionalities affects the design in many ways
  - Functional sub-system/components
  - Domain-specific logics
  - Interaction between these components
  - …
- Many tools are being used to address the functional aspect of sw system
  - Funtional decomposition
  - Functional testing
  - ...

Positioning Architecture



- 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.

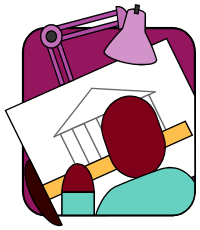
Outline

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

The Role of the Architect

client, users

requirements



architect

solutions

developers

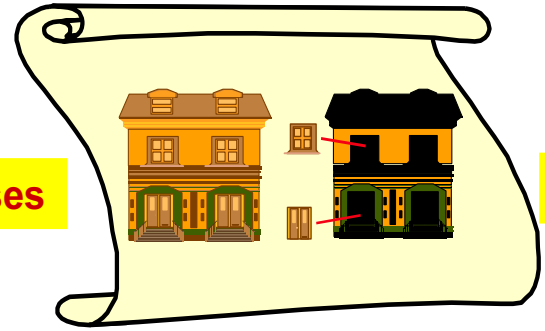
assess

creates

assess

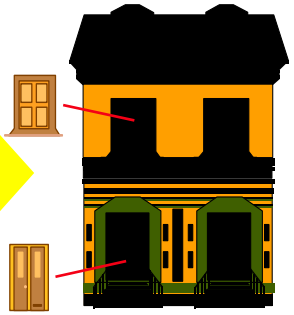
appearance, behaviour

visualises



architectural design

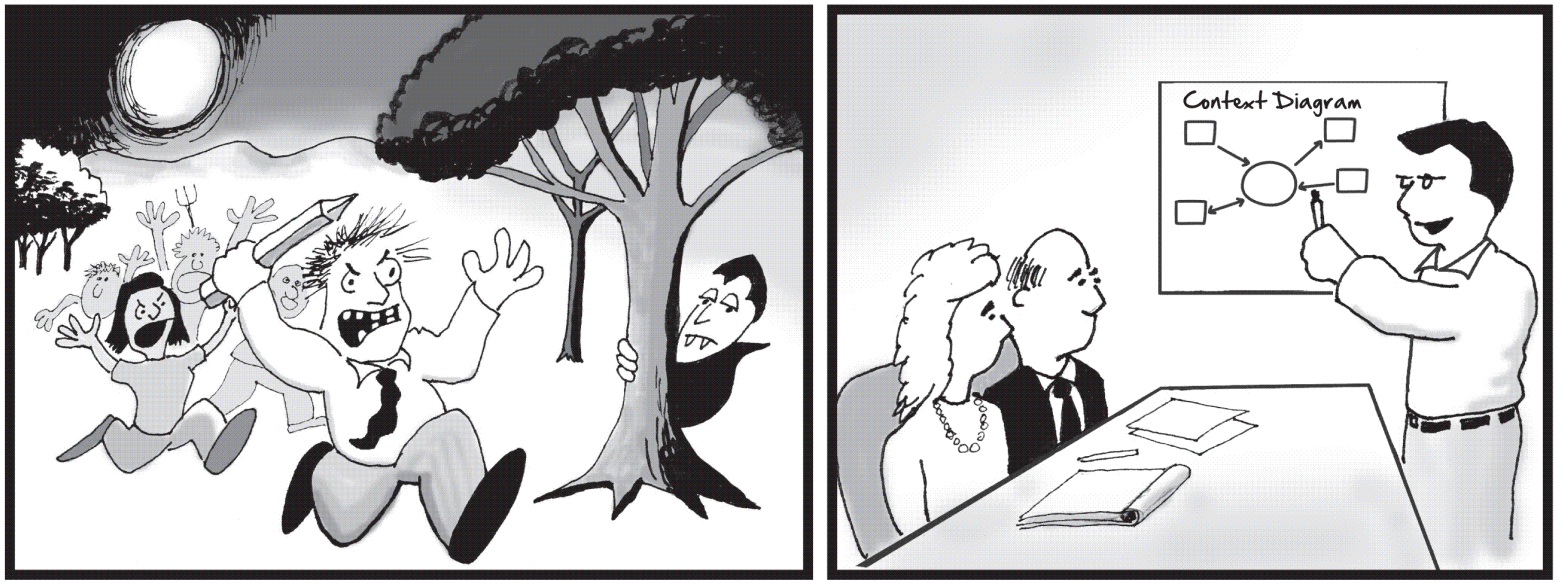
prescribes



construction, co-operation

Stakeholder?

If you think a stakeholder is someone running through the woods looking for a vampire...



You might not be a Business Analyst!

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."

Standard ISO/IEC 15288 (ISO/IEC 1999)

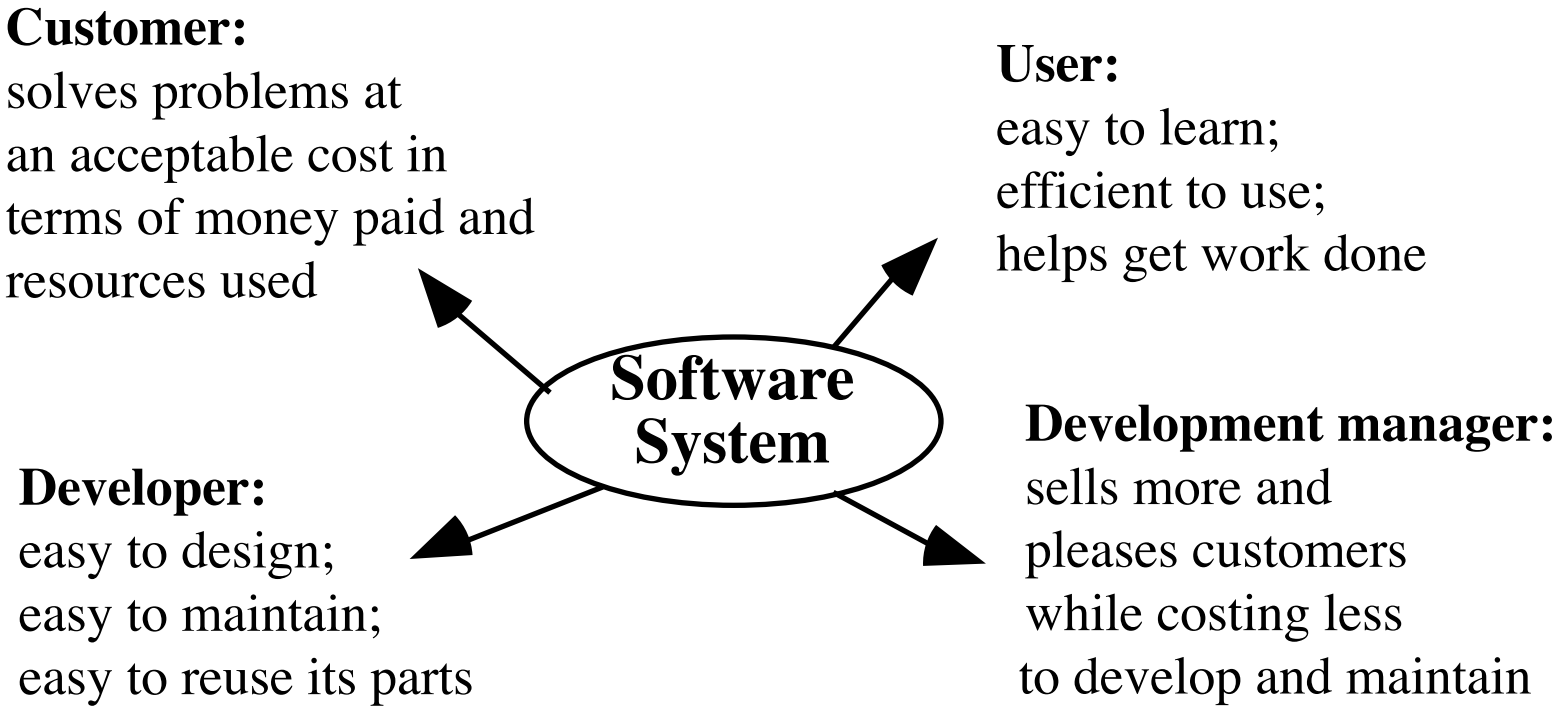
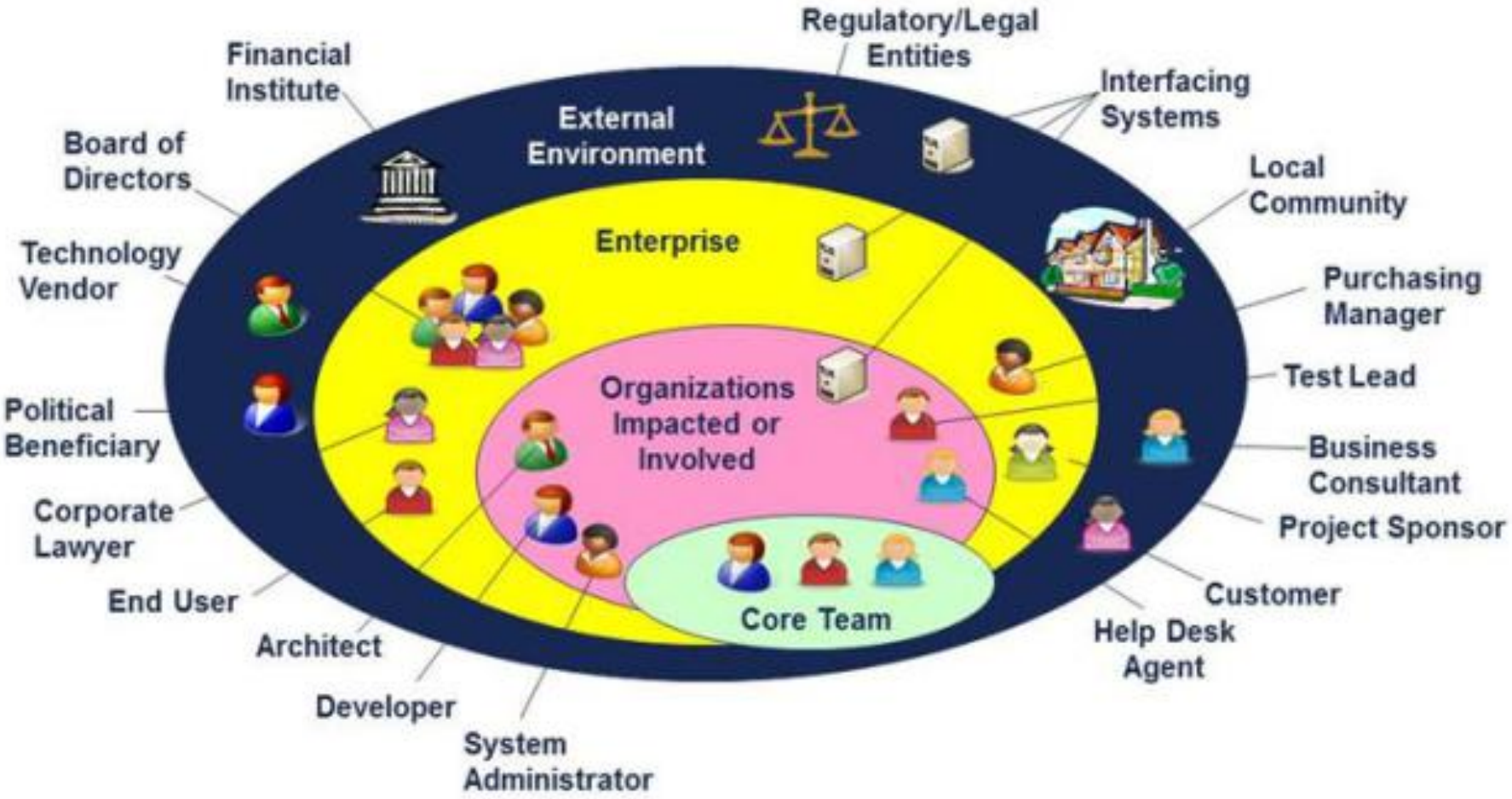


Figure from: Lethbridge and Laganiere

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?

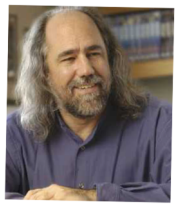
- When developing a **new system**
- When **changing a system**
  - if an architecture description is not available, or insufficient, as a basis for change
  - adapt the architecture documentation to changes
- When **integrating** existing systems
- For special **communication needs** to provide a common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

Outline

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

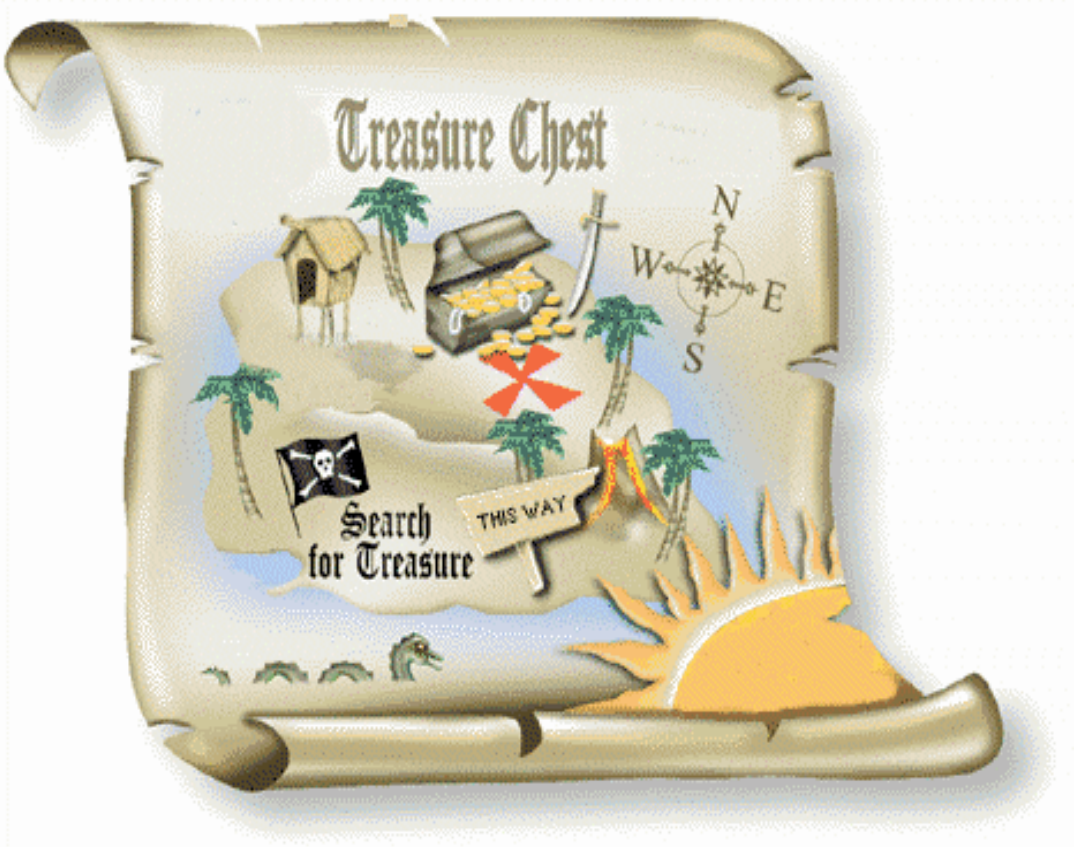
Architecture is making decisions

> 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’

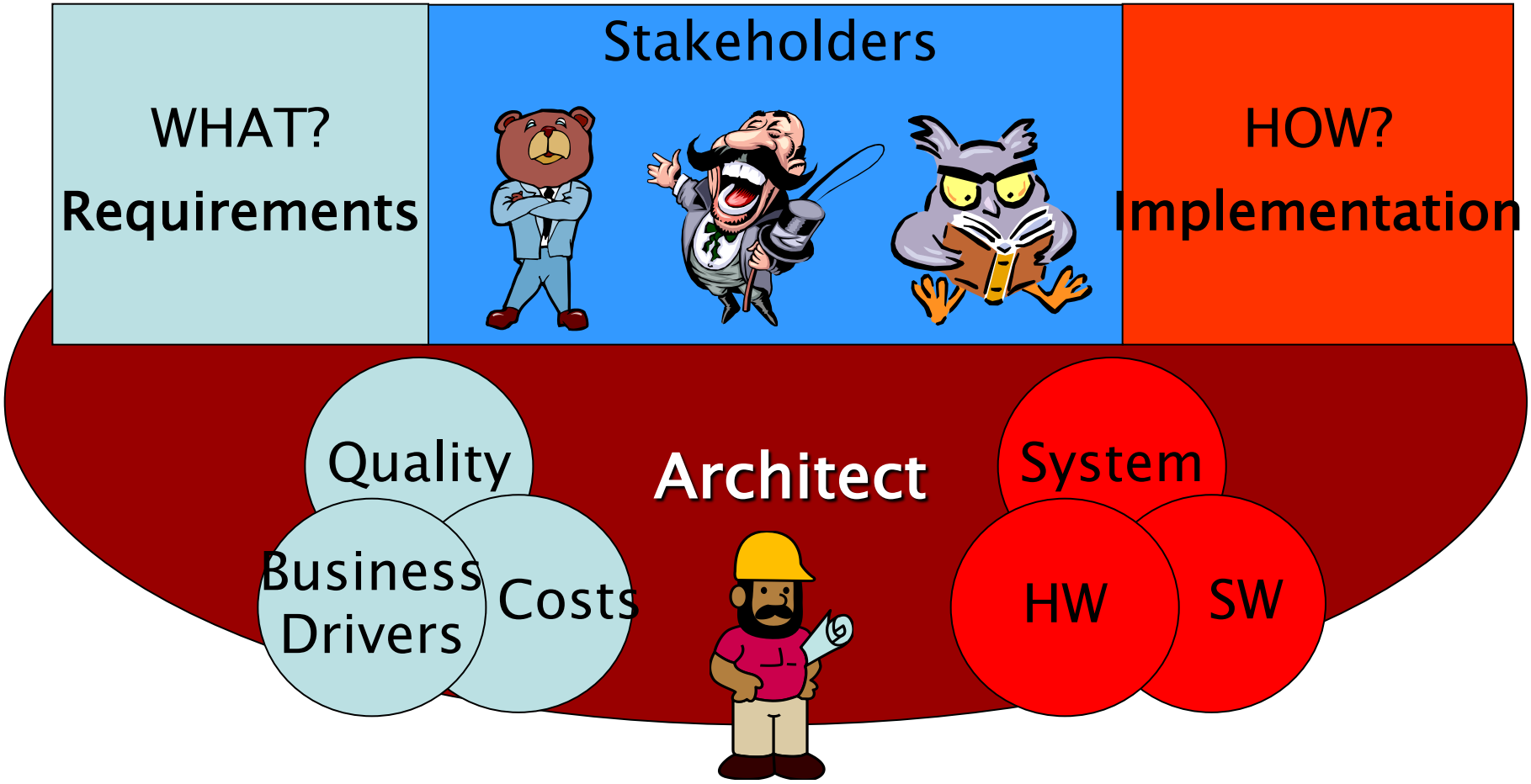
No ideal solution



Discovery may be exploratory

There is no ideal system to be discovered.

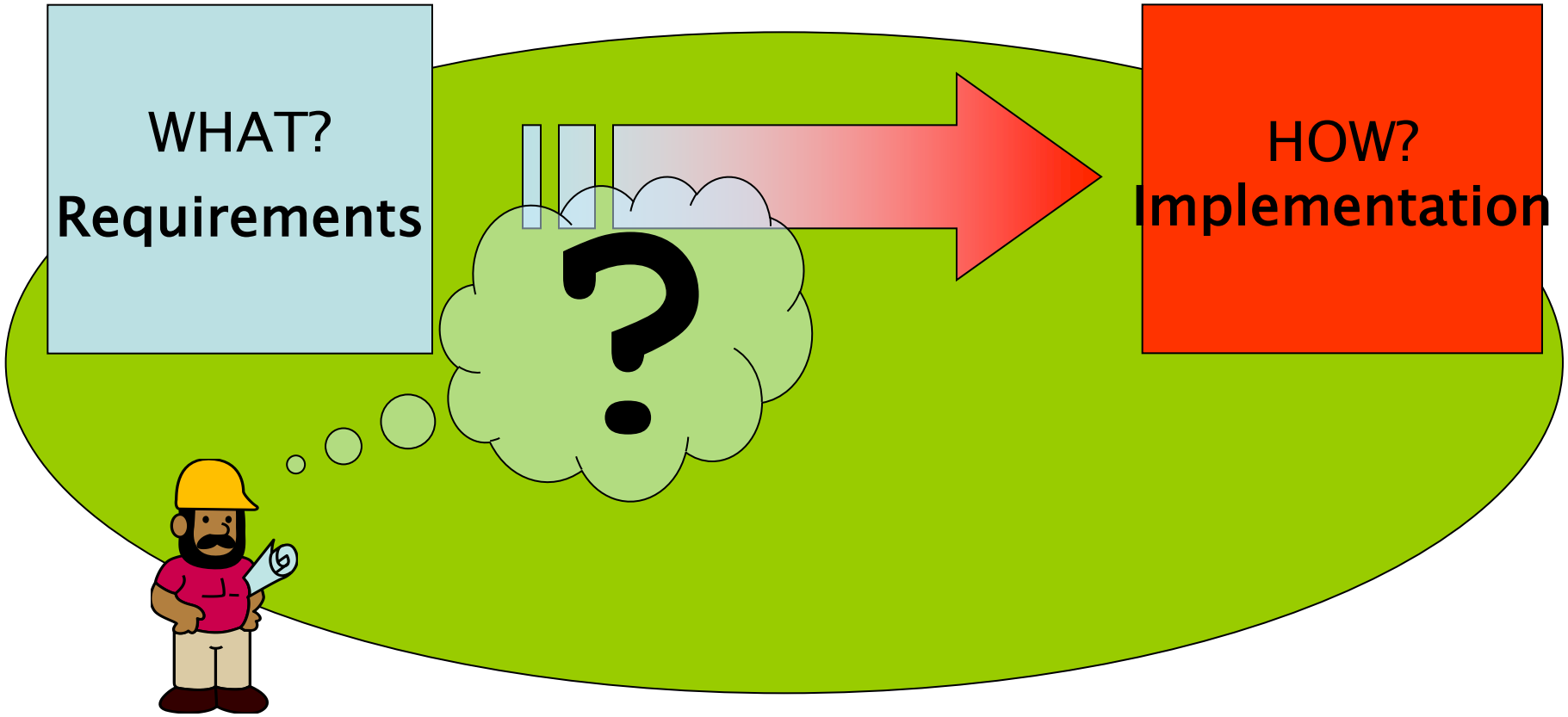
Process: Working Together



Close and effective interaction between these actors is essential!

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

How to Bridge the Gap?

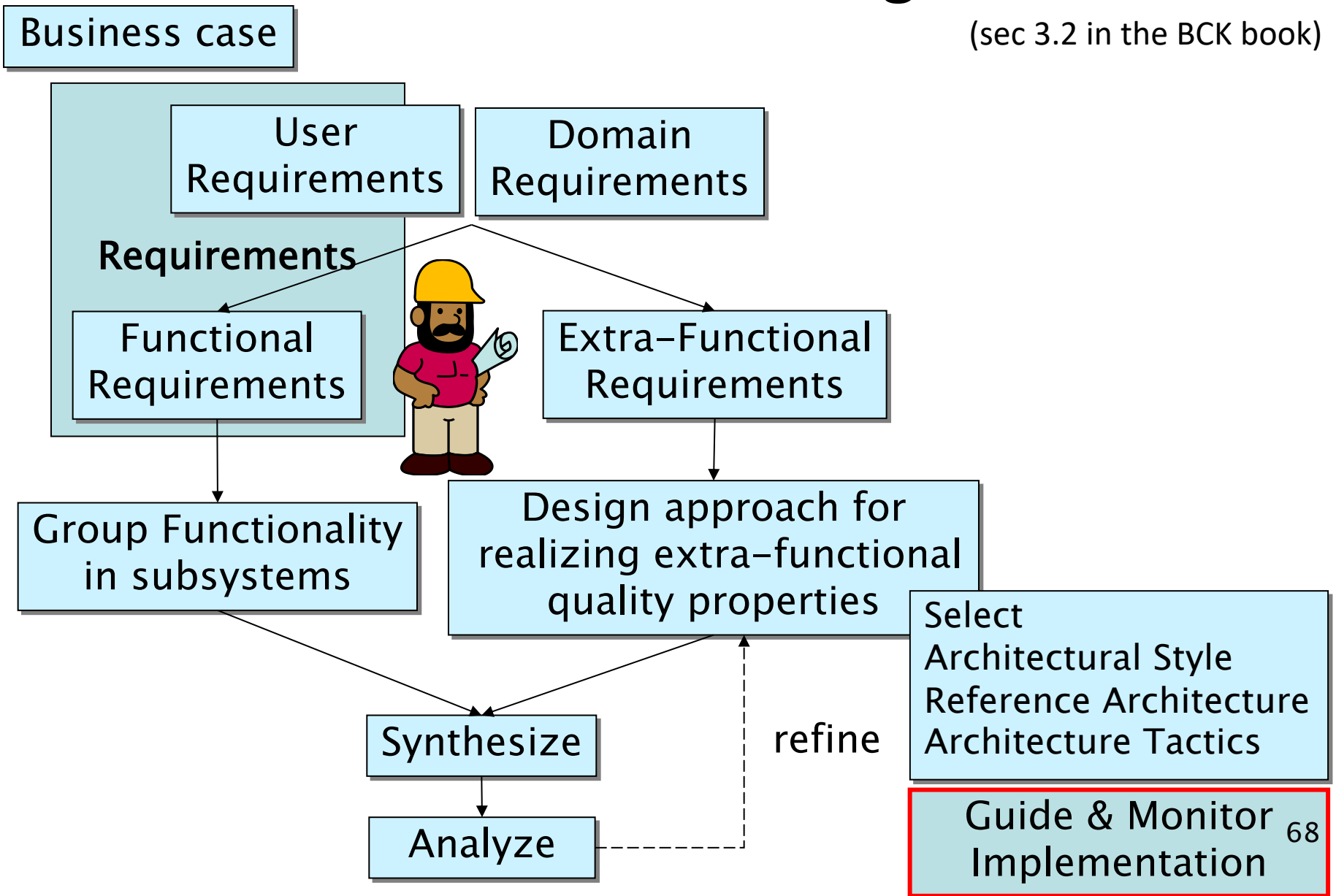


Traditional Answer

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

Software Architecture Design Process

(sec 3.2 in the BCK book)



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?

Business Model Canvas

The Business Model Canvas

Designed for:	Designed by:	Date:	Version:

Key Partners

Who are our Key Partners?
Who are our key suppliers?
Which Key Resources are we acquairing from partners?
Which Key Activities do partners perform?

MOTIVATIONS FOR PARTNERSHIPS
Optimization and economy
Reduction of risk and uncertainty
Acquisition of particular resources and activities

Key Activities

What Key Activities do our Value Propositions require?
Our Distribution Channels?
Customer Relationships?
Revenue streams?

CATERGORIES
Production
Problem Solving
Platform/Network

Key Resources

What Key Resources do our Value Propositions require?
Our Distribution Channels? Customer Relationships?
Revenue Streams?

TYPES OF RESOURCES
Physical
Intellectual (brand patents, copyrights, data)
Human
Financial

Value Propositions

What value do we deliver to the customer?
Which one of our customer's problems are we helping to solve?
What bundles of products and services are we offering to each Customer Segment?
Which customer needs are we satisfying?

CHARACTERISTICS
Newness
Performance
Customization
"Getting the Job Done"
Design
Brand/Status
Price
Cost Reduction
Risk Reduction
Accessibility
Convenience/Usability

Customer Relationships

What type of relationship does each of our Customer Segments expect us to establish and maintain with them?
Which ones have we established?
How are they integrated with the rest of our business model?
How costly are they?

EXAMPLES
Personal assistance
Dedicated Personal Assistance
Self-Service
Automated Services
Communities
Co-creation

Channels

Through which Channels do our Customer Segments want to be reached?
How are we reaching them now?
How are our Channels integrated?
Which ones work best?
Which ones are most cost-efficient?
How are we integrating them with customer routines?

CHANNEL PHASES
1. *Awareness*
   How do we raise awareness about our company's products and services?
2. *Evaluation*
   How do we help customers evaluate our organization's Value Proposition?
3. *Purchase*
   How do we allow customers to purchase specific products and services?
4. *Delivery*
   How do we deliver a Value Proposition to customers?
5. *After sales*
   How do we provide post-purchase customer support?

Customer Segments

For whom are we creating value?
Who are our most important customers?

Mass Market
Niche Market
Segmented
Diversified
Multi-sided Platform

Cost Structure

What are the most important costs inherent in our business model?
Which Key Resources are most expensive?
Which Key Activities are most expensive?

IS YOUR BUSINESS MORE
Cost Driven (leanest cost structure, low price value proposition, maximum automation, extensive outsourcing)
Value Driven (focused on value creation, premium value proposition)

SAMPLE CHARACTERISTICS
Fixed Costs (salaries, rents, utilities)
Variable costs
Economies of scale
Economies of scope

Revenue Streams

For what value are our customers really willing to pay?
For what do they currently pay?
How are they currently paying?
How would they prefer to pay?
How much does each Revenue Stream contribute to overall revenues?

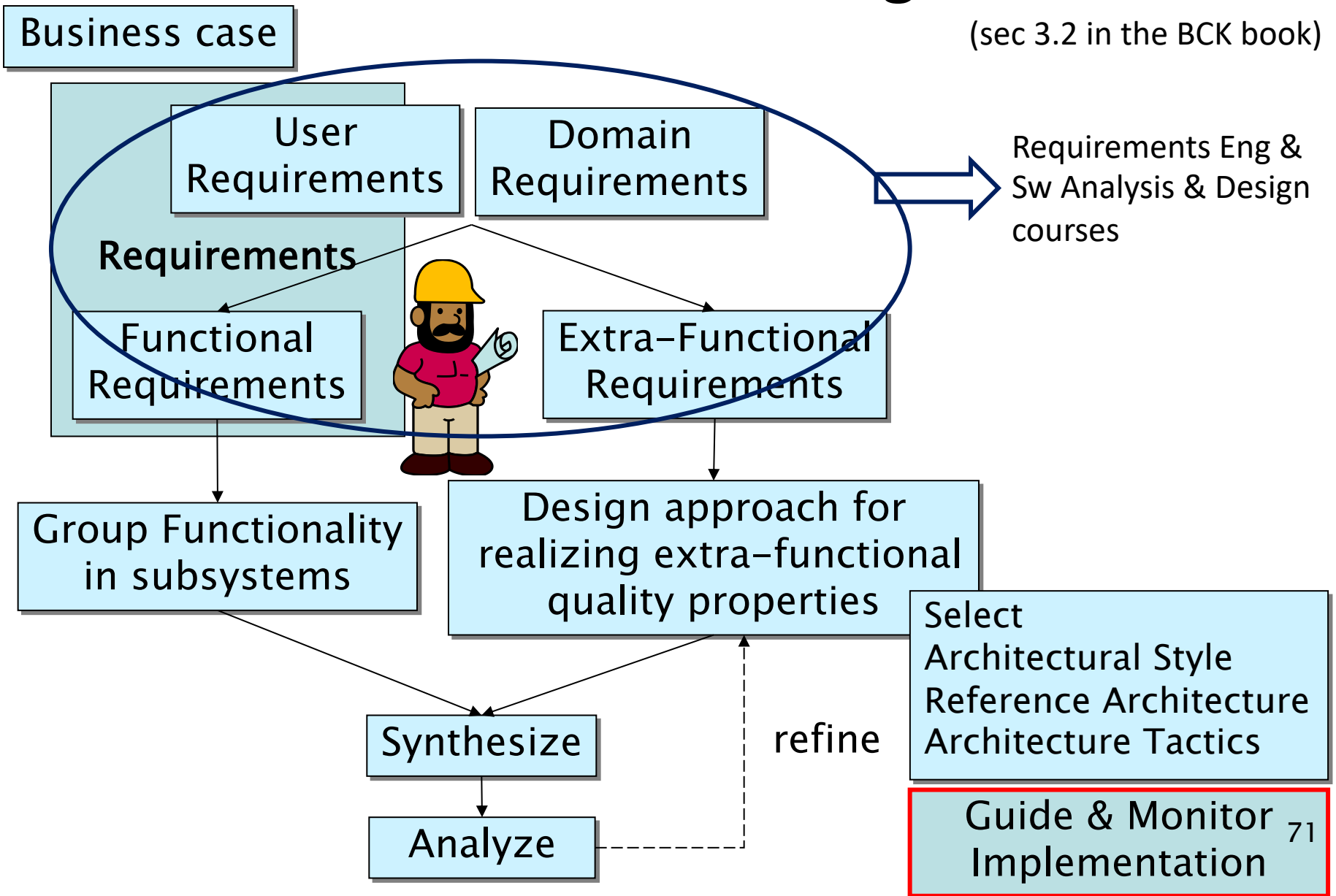
TYPES
Asset sale
Usage fee
Subscription Fees
Lending/Renting/Leasing
Licensing
Brokerage fees
Advertising

FIXED PRICING
List Price
Product feature dependent
Customer segment dependent
Volume dependent

DYNAMIC PRICING
Negotiation (bargaining)
Yield Management
Real-time-Market

Software Architecture Design Process

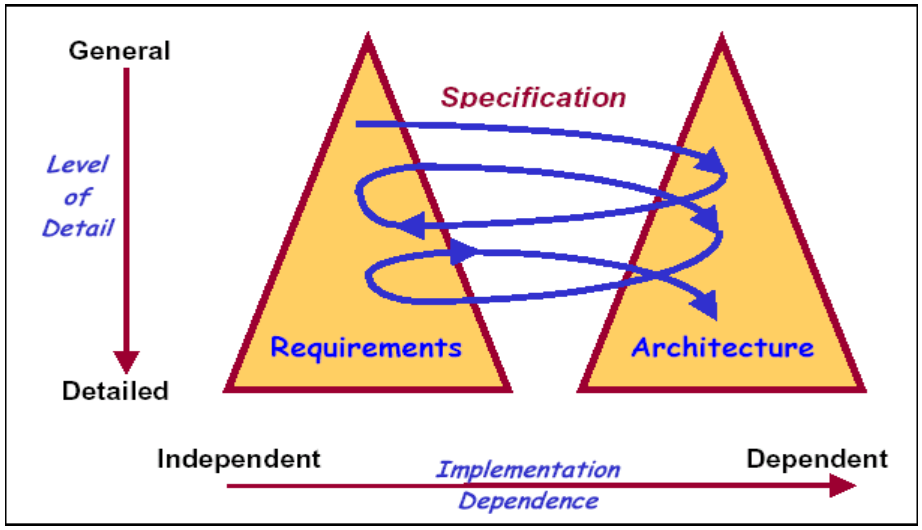
(sec 3.2 in the BCK book)



Twin Peaks Process

Separate but concurrent development of requirements & architecture

WHAT:
problem structuring

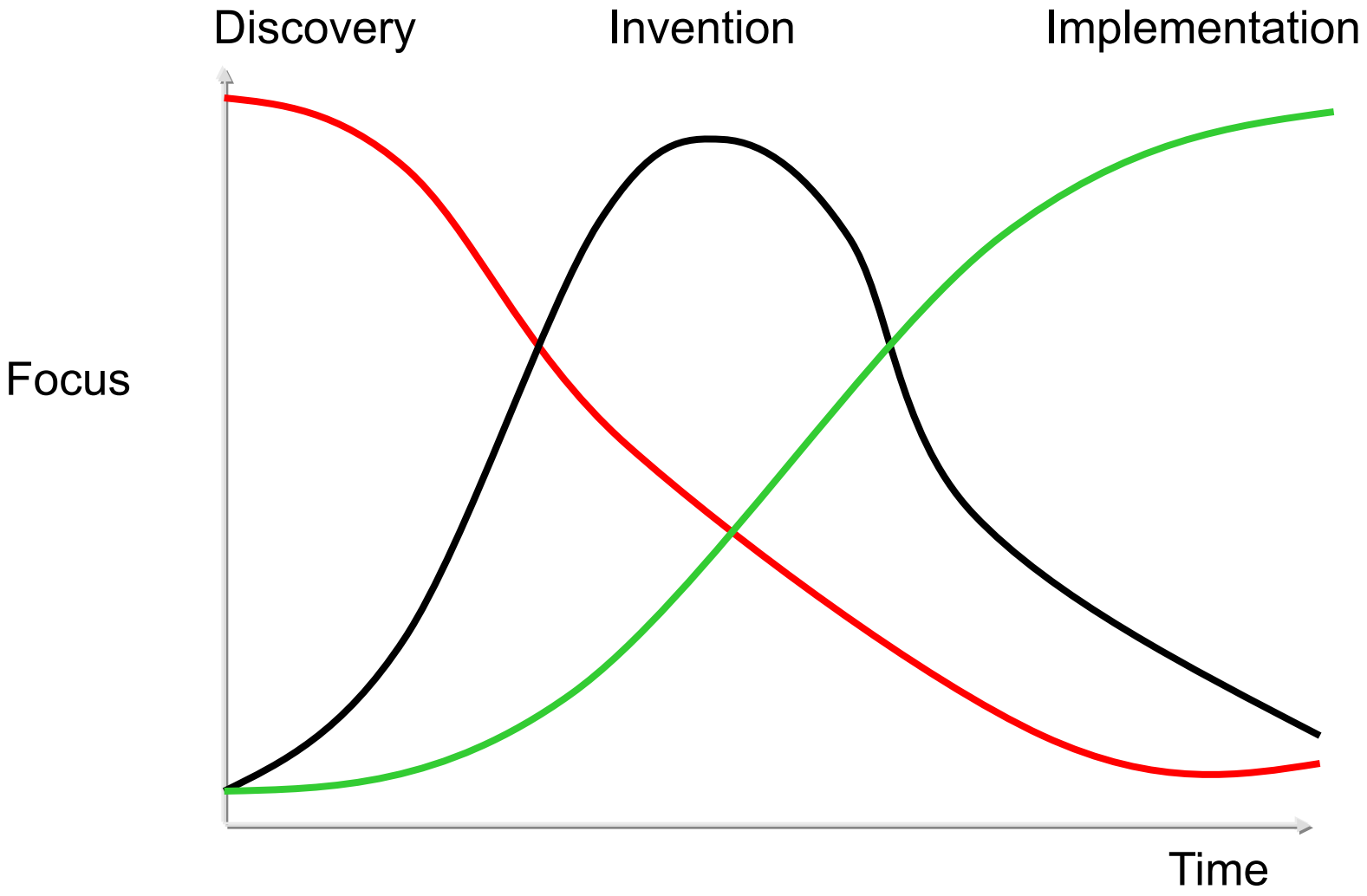


HOW:
solution structuring

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

Paris Avgeriou Keynote at SEAA 2017

Architecting is not only about the solution space, but also about the problem space: identifying, scoping, understanding the problem space.

Architecture is not only IT/technology

- Technical and non-techical issues and options are intertwined
  - Architects deciding on the type of database

  versus

  - Management deciding on new strategic partnership

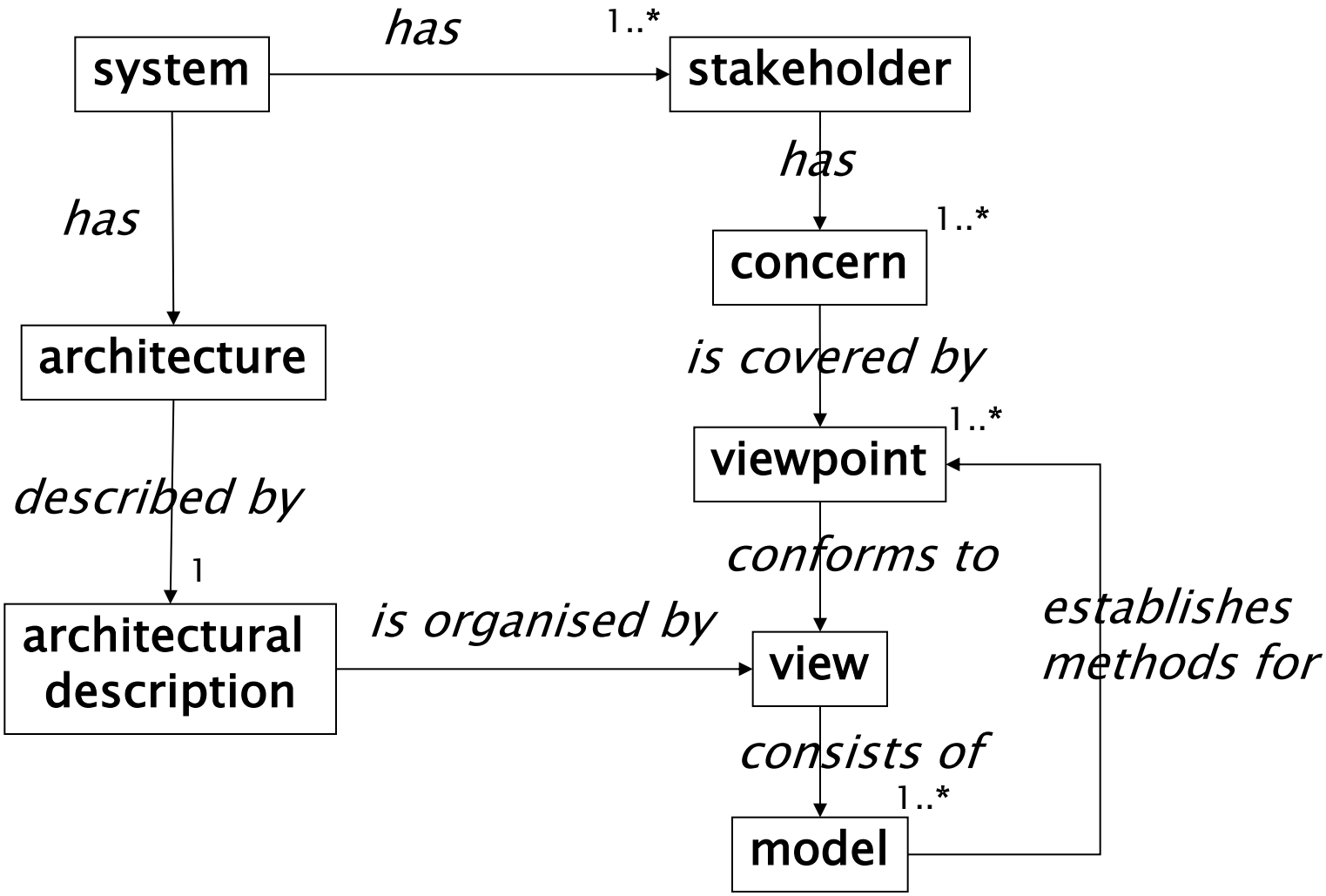
  or

  - Management deciding on budget

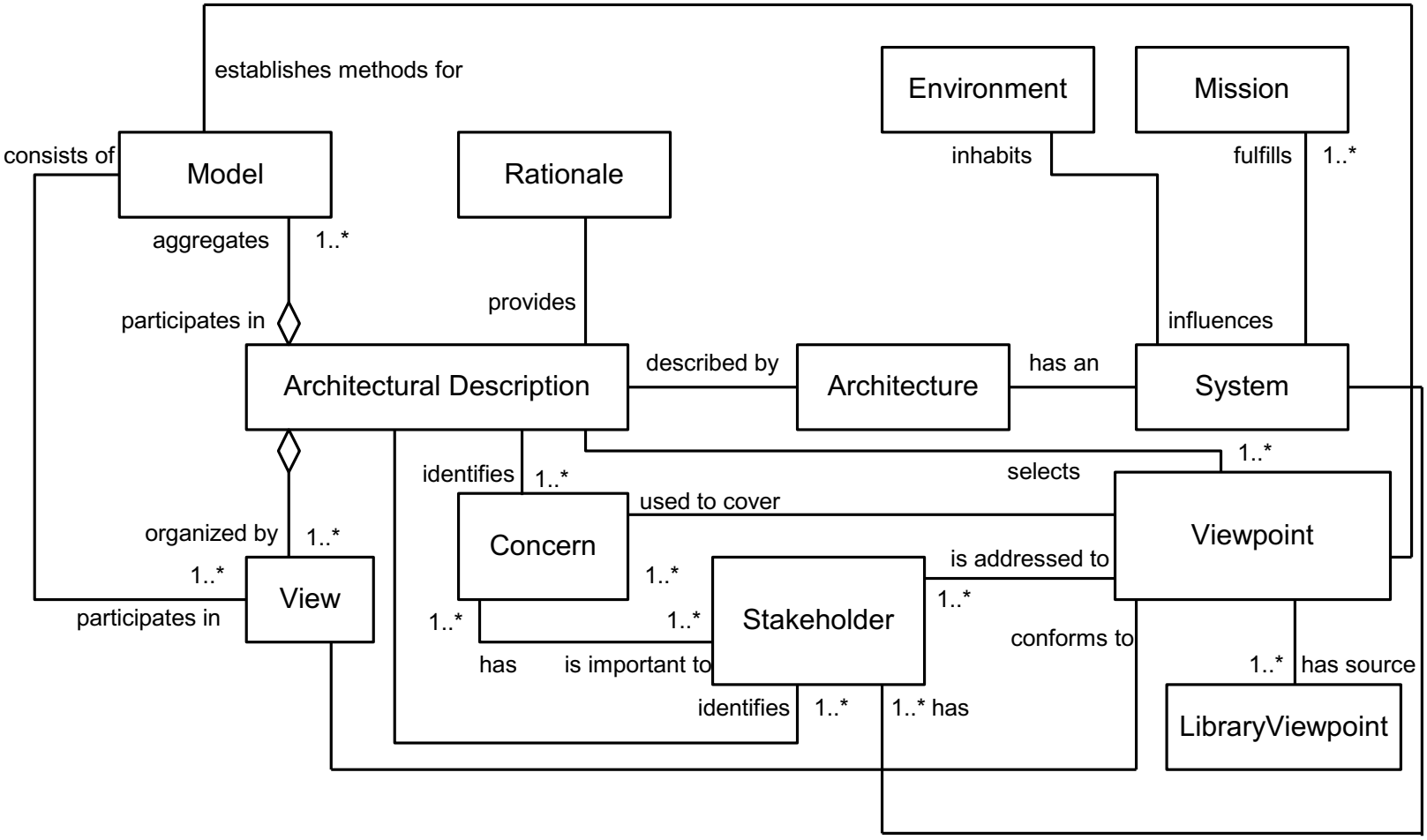
Outline

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

Overview (According to IEEE 1471)



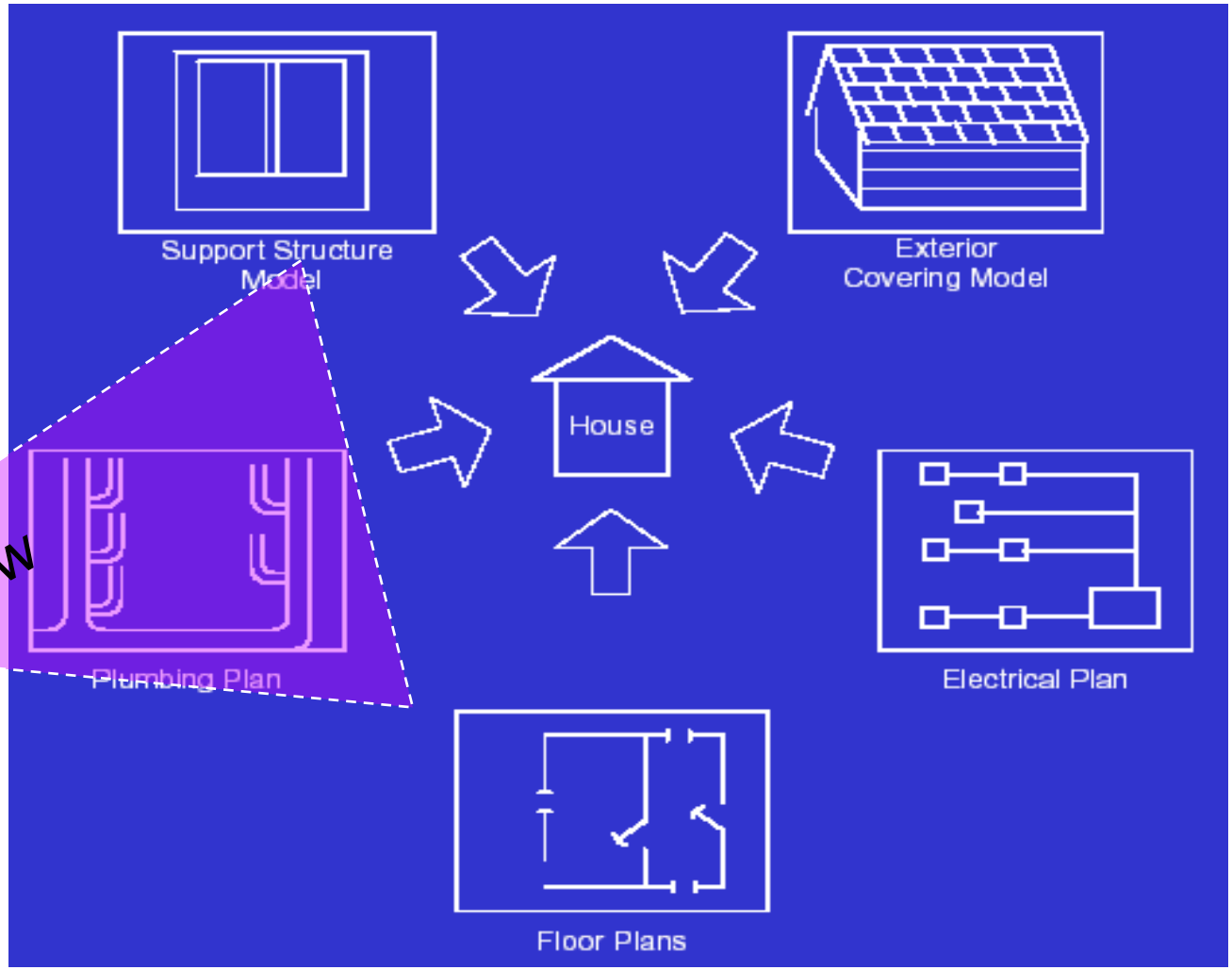
ISO/IEC/IEEE 42010:2011 Conceptual Framework



Outline

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

Viewpoints & views



view point

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.

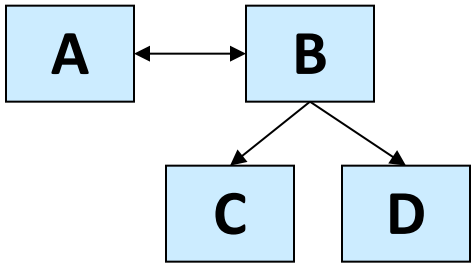
An architectural model may participate in more than one view.

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

Example 4+1 Views model

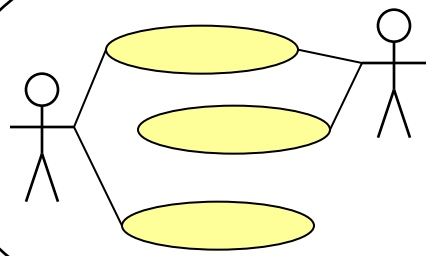
Structure view:
class/component-diagram



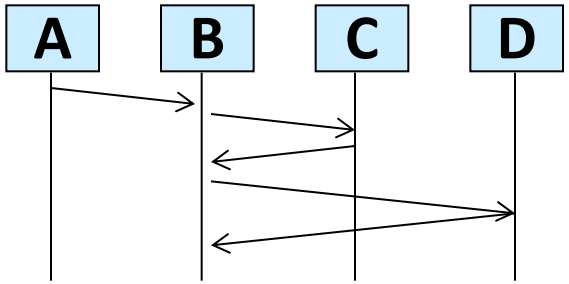
Development view

file ownership
Config. Mngnt view
versioning policies
…

Use cases

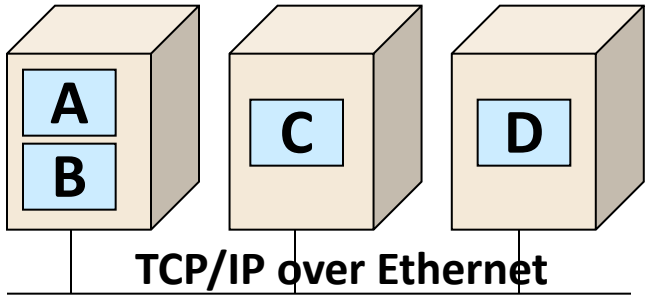


Behaviour view:
Sequence diagram



BC/WC e2e-response times, freq.

Deployment view:
physical model + mapping

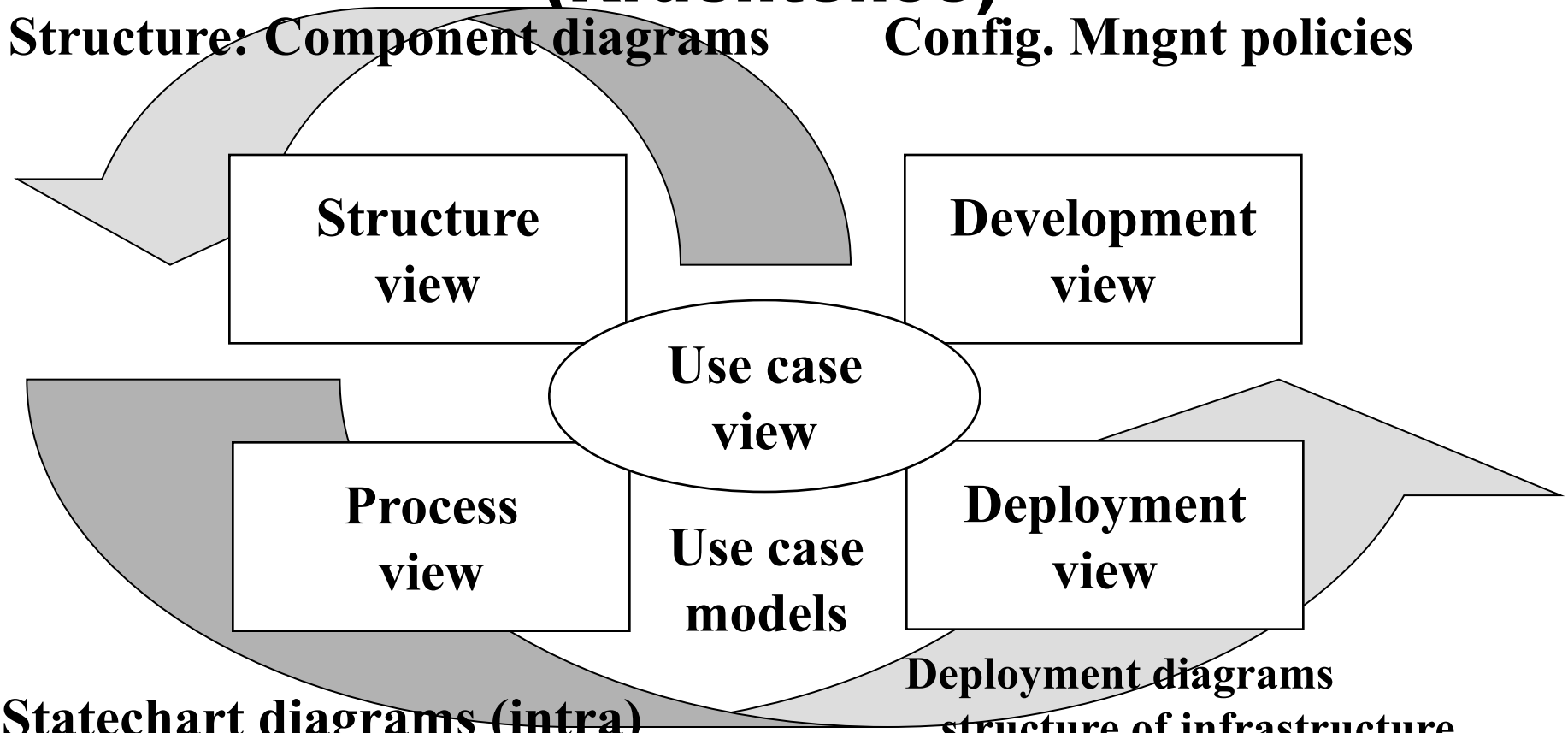


bandwidth, availability

The 4 + 1 View Model (Kruchten95)

Structure: Component diagrams

Config. Mngt policies

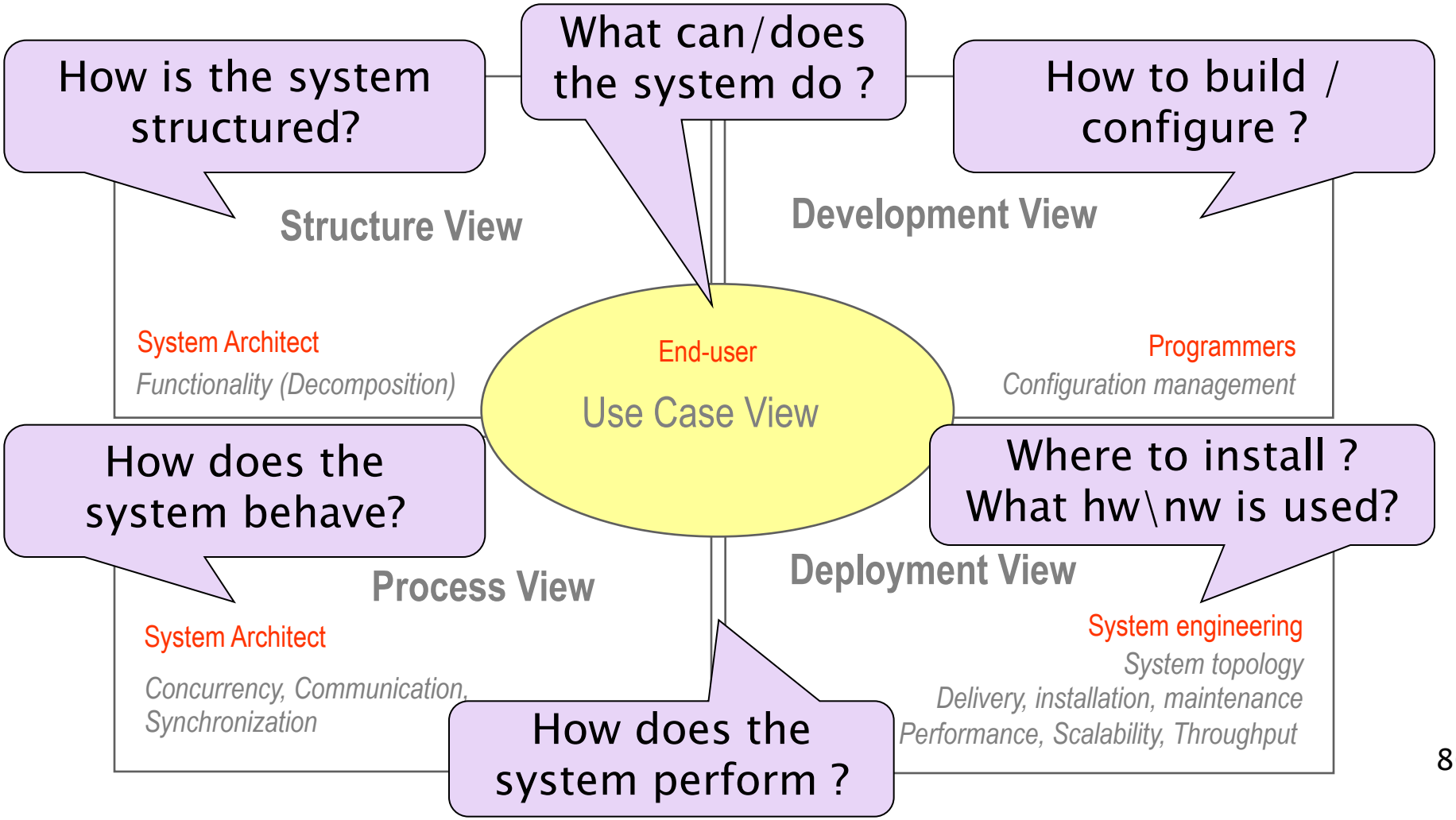


Statechart diagrams (intra)

Interaction diagrams (inter)

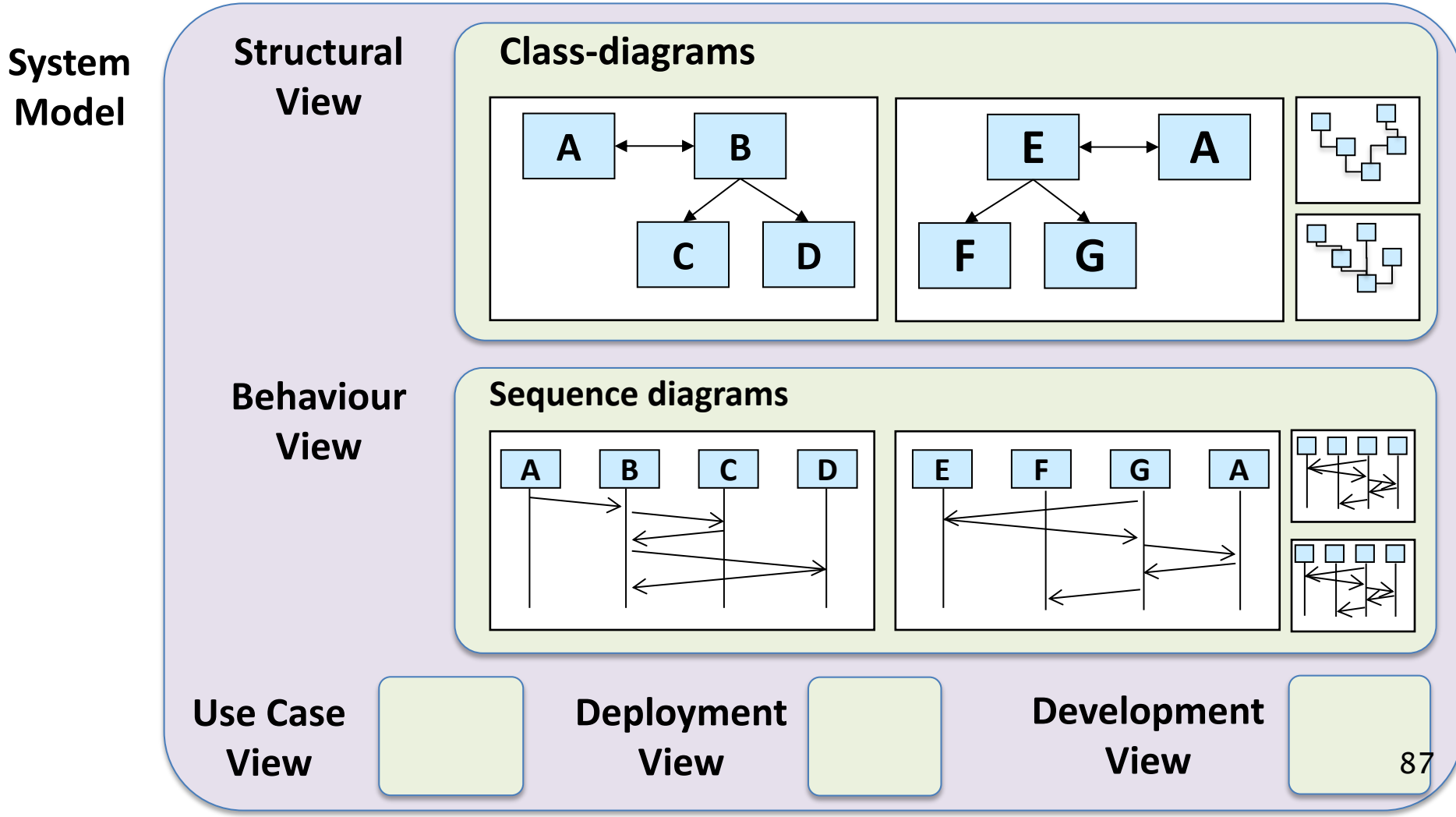
Deployment diagrams
structure of infrastructure
rules for mapping of design and process view onto infra

4+1 Views Representation of System Architecture



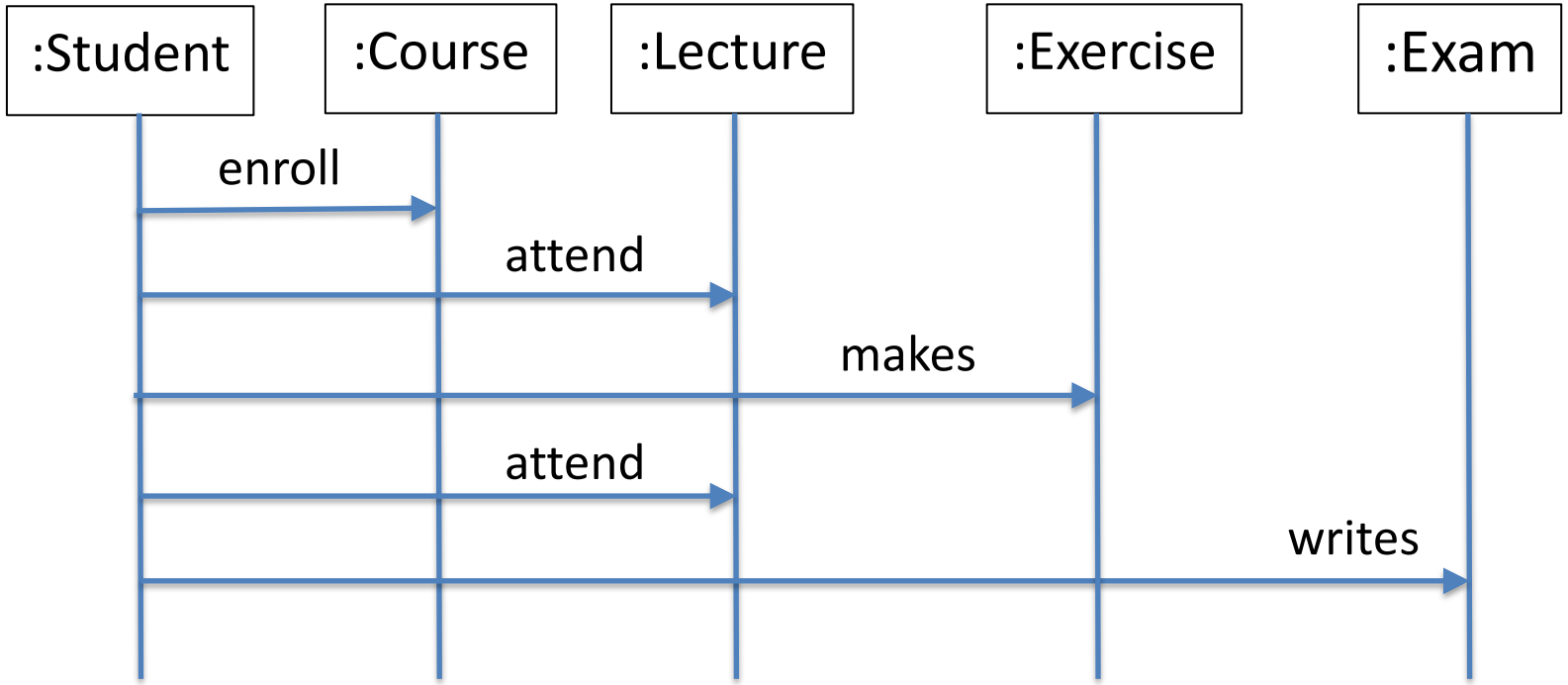
1 Model = union of multiple views

each view has one or more diagrams



Behaviour View!

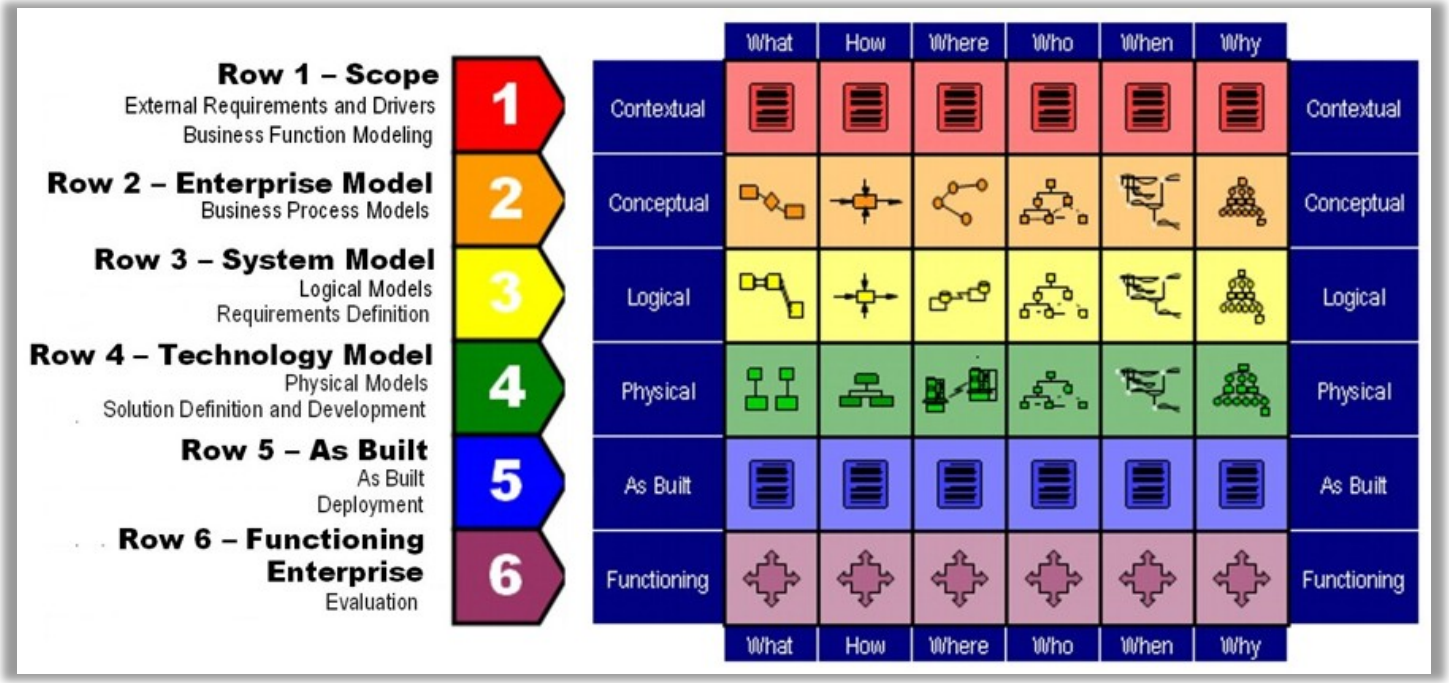
Most illustrations of software architecture us structural views, but the behavioural views as just as important!



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

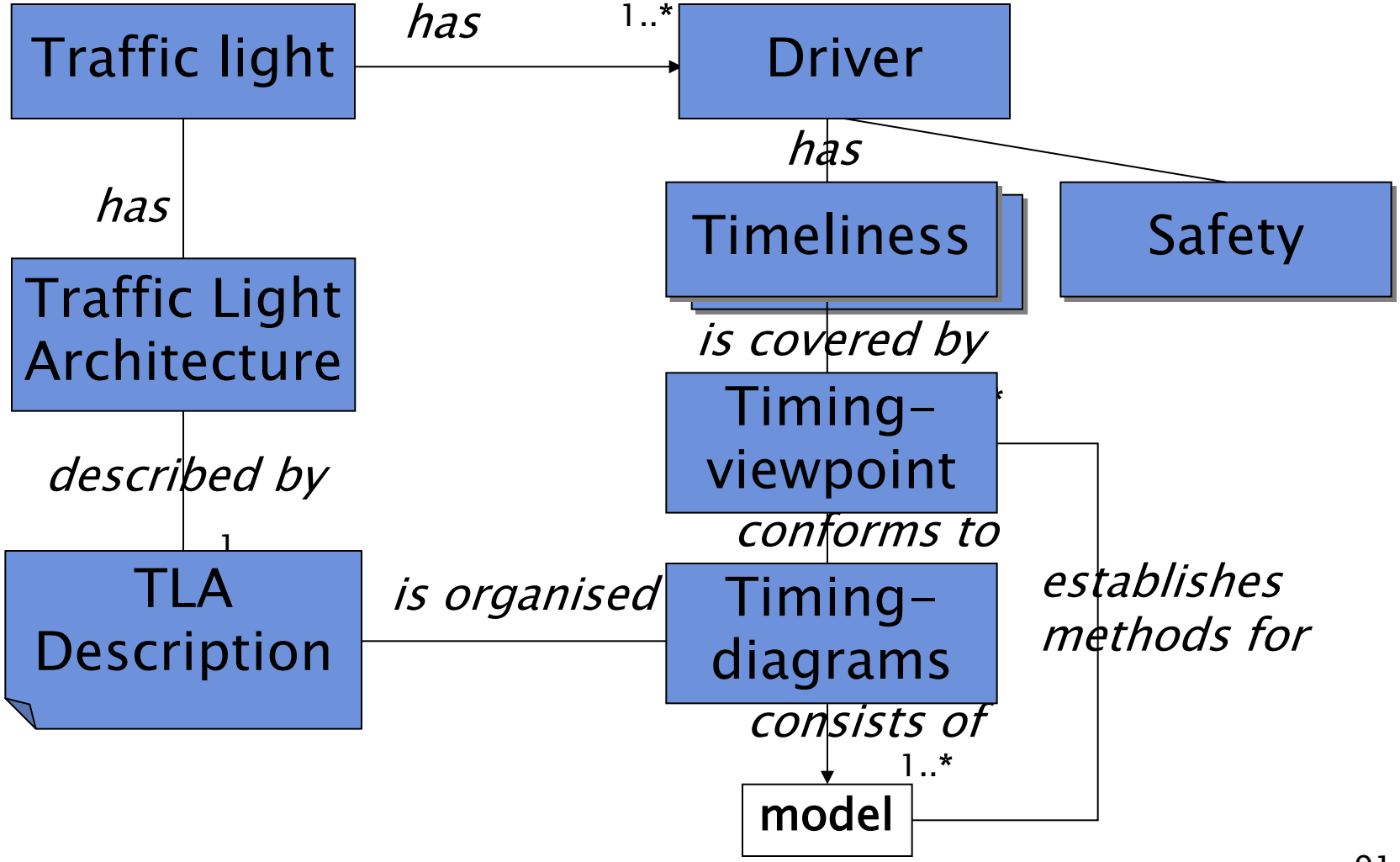
Other 'Views'-paradigms exist

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

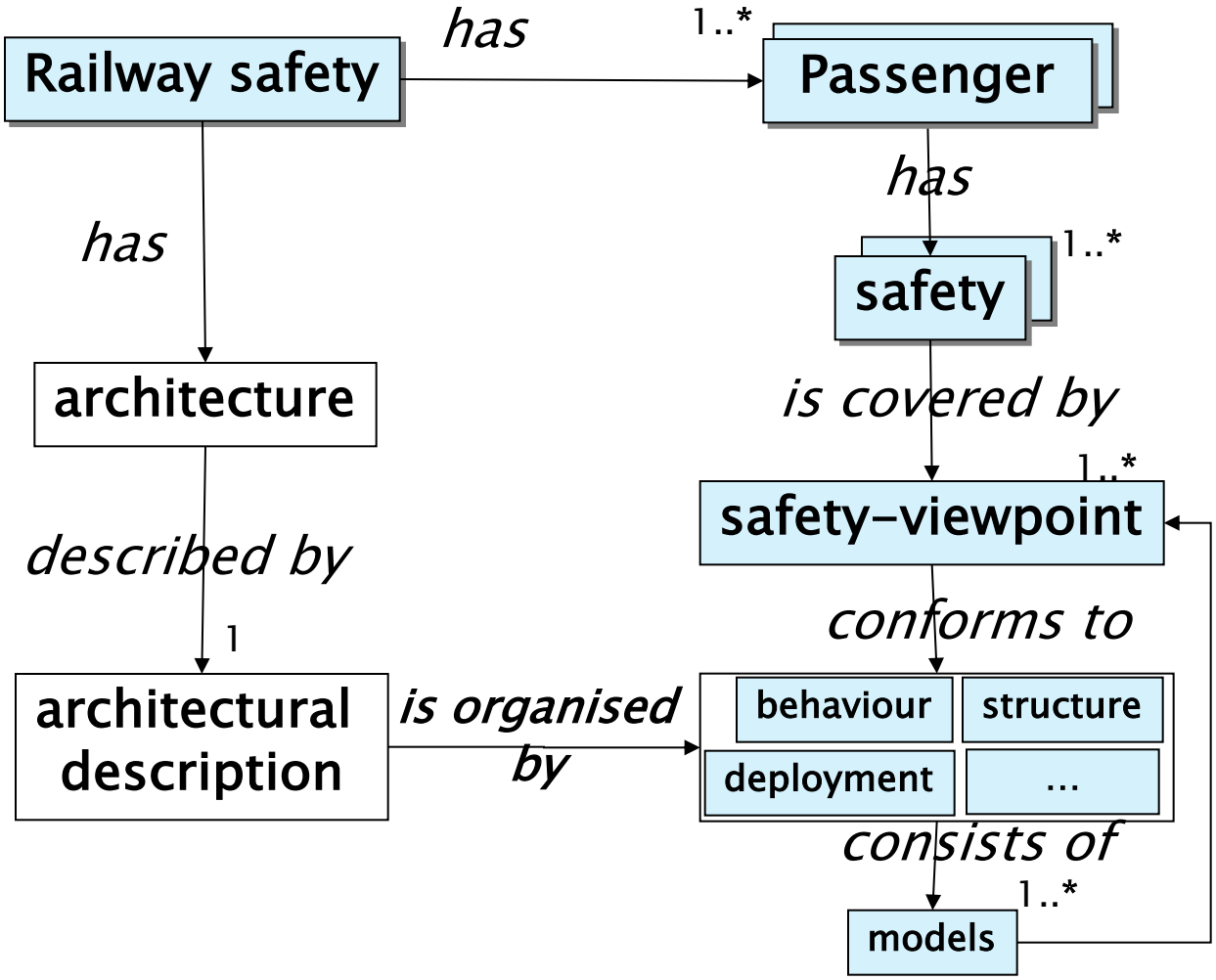


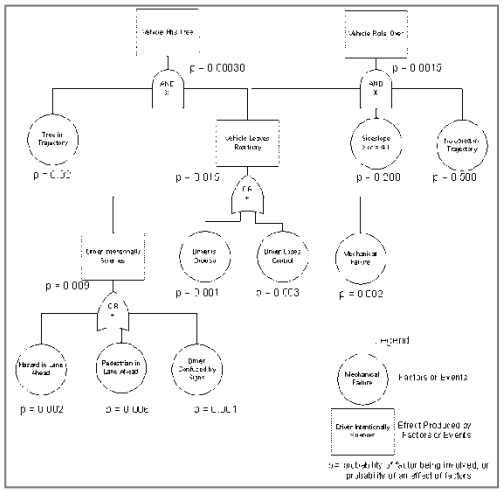
https://en.wikipedia.org/wiki/Zachman_Framework

Overview - example



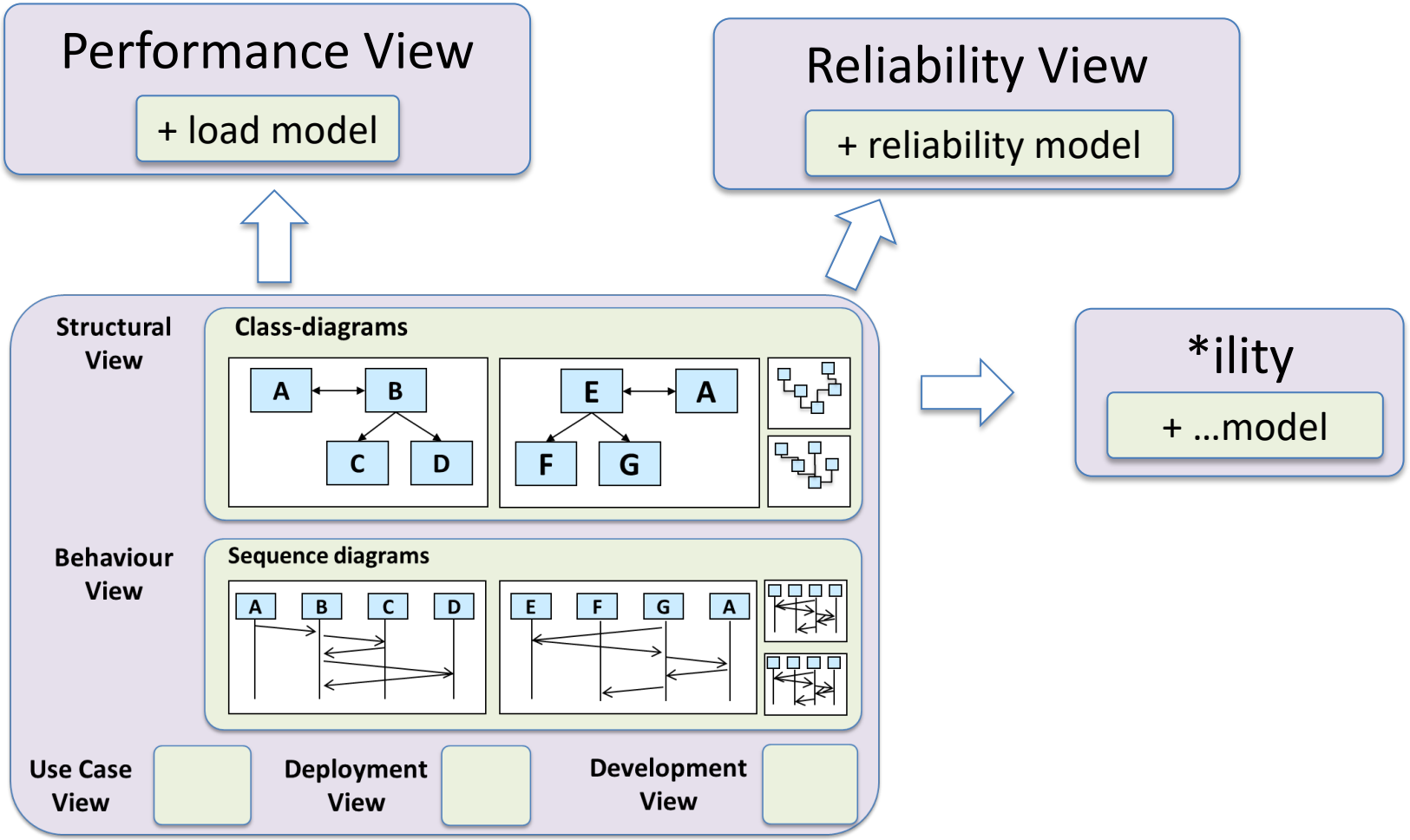
Example (According to IEEE 1471)





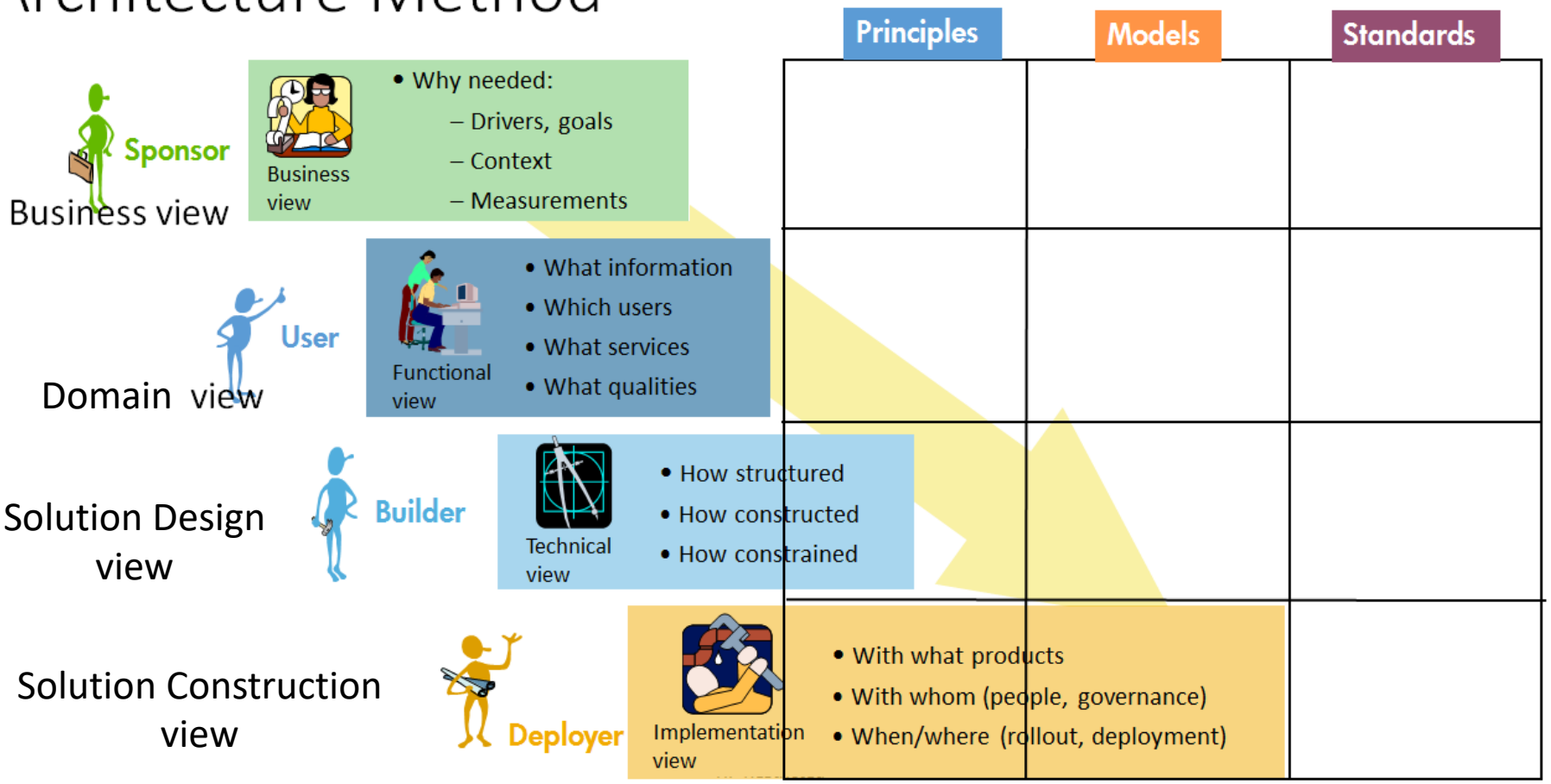
Method: Fault trees

Views for Extra-Functional Properties



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

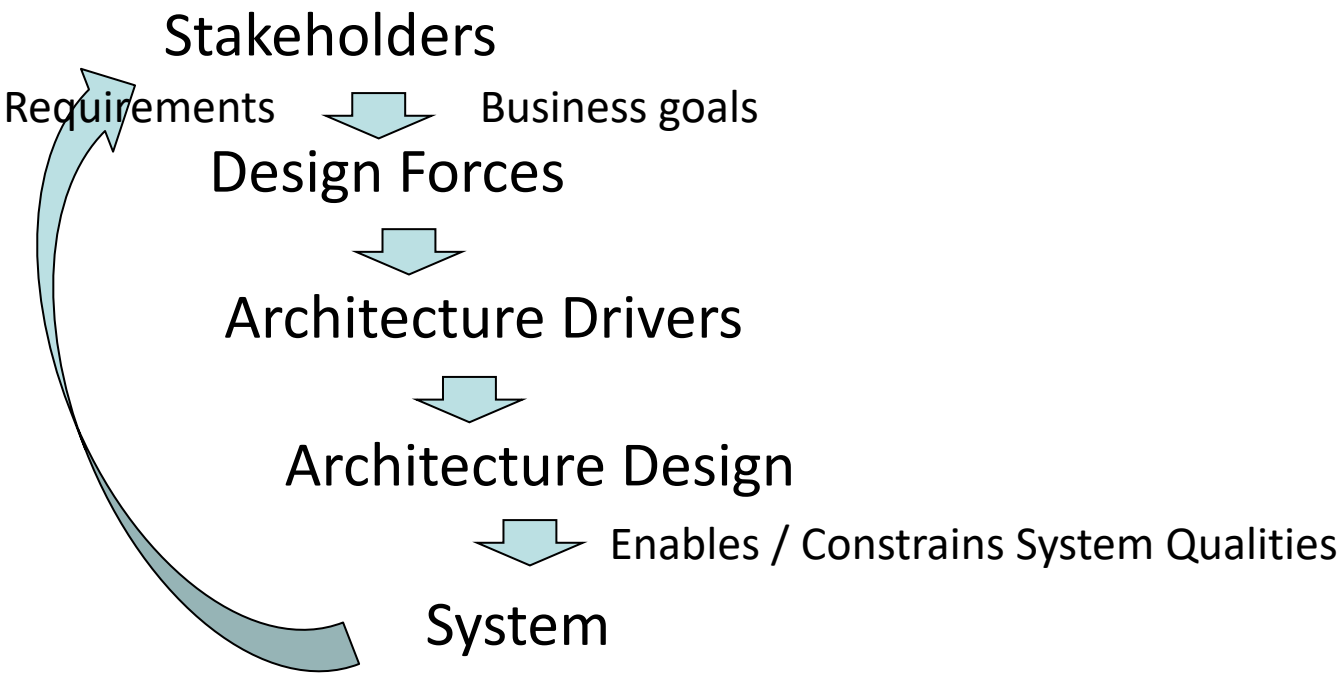
Architecture Method



Discussion

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

Summary - 1


Stakeholders
Requirements
Business goals
Design Forces
Architecture Drivers
Architecture Design
Enables / Constrains System Qualities
System

Summary - 2

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