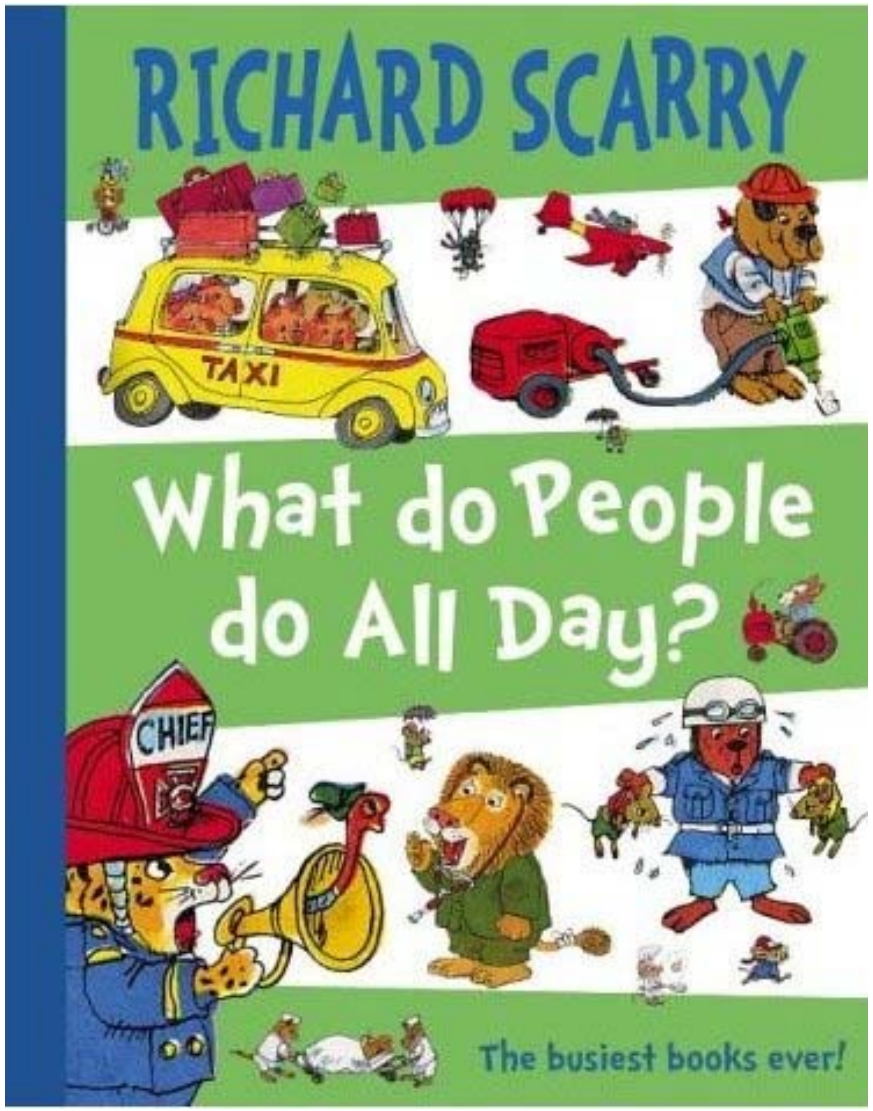
What do IT Architects do all day?

Roles and Responsibilities

Dr. Marcel Schlatter
IBM Distinguished Engineer
Member of the IBM Academy of Technology
marcel.schlatter@ch.ibm.com

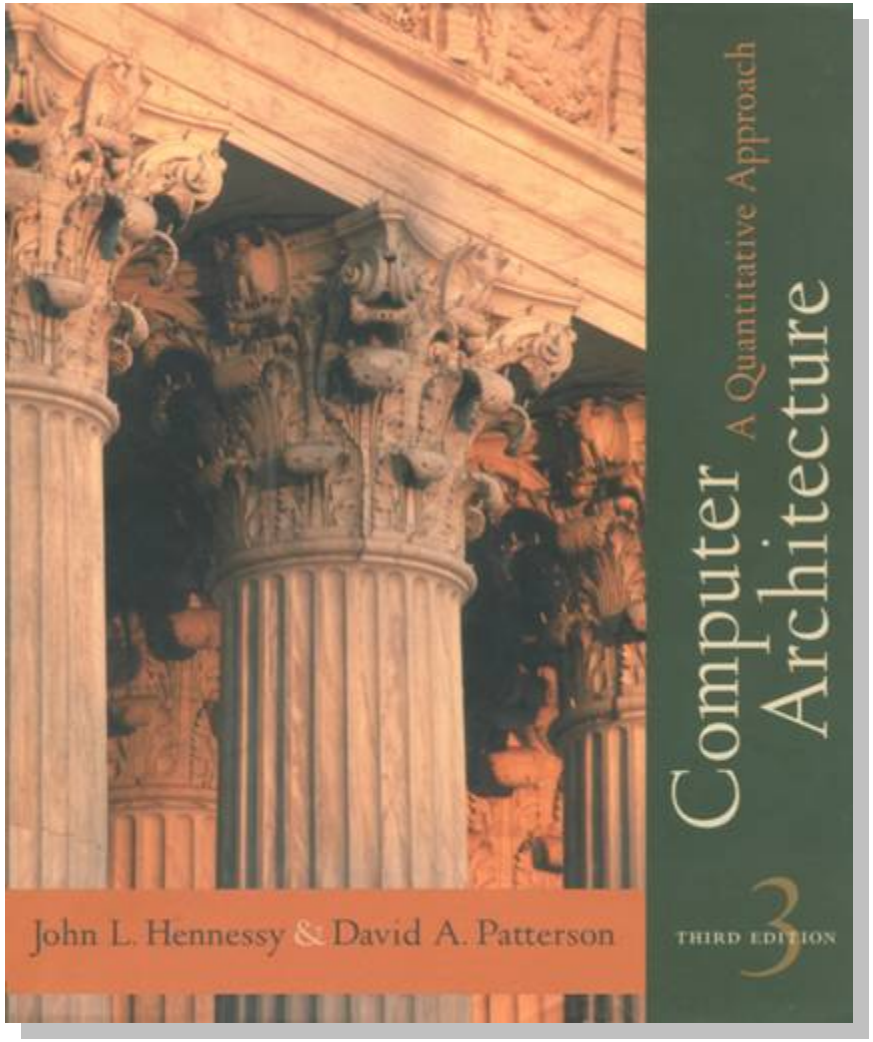

RICHARD SCARRY
TAXI
What do People do All Day?
CHIEF
The busiest books ever!

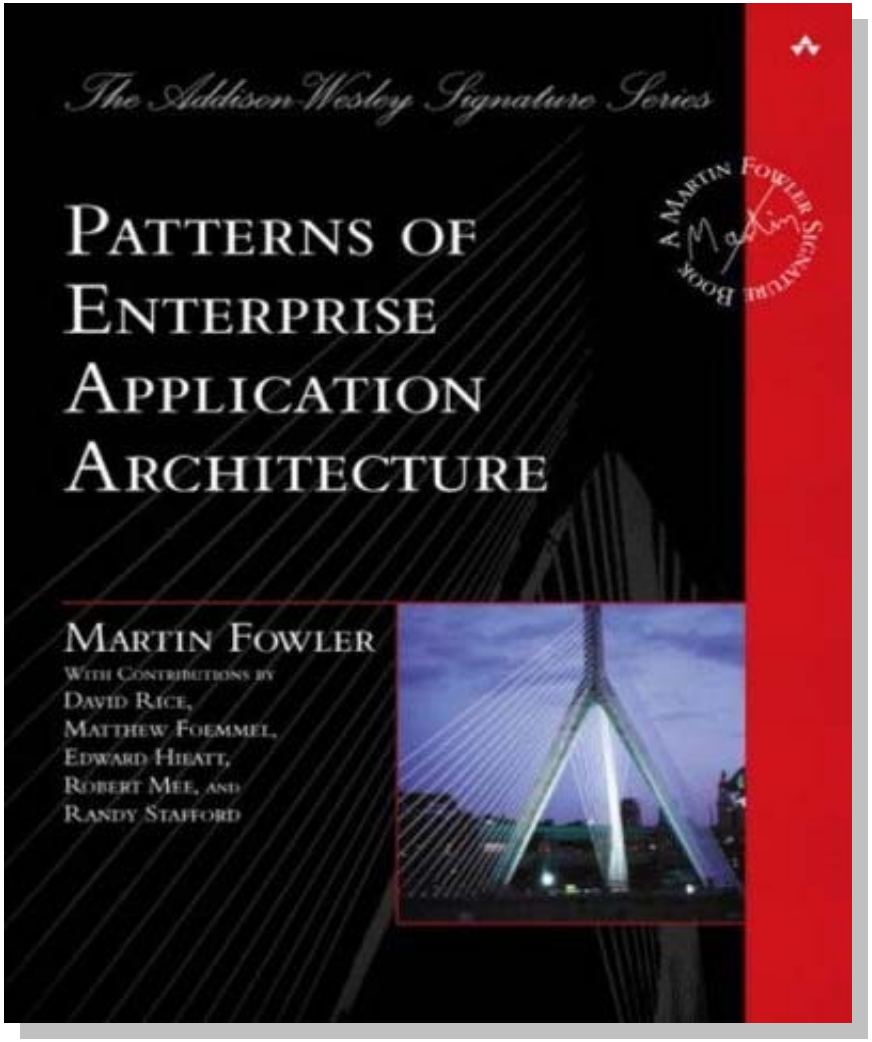
Contents

- Roles and responsibilities
- Heuristics
  - Experience
- Language
  - Vocabulary
  - Architecture Description Standard
- Method
  - Codified experience
  - Process to support Architectural Thinking
  - Reference architectures
  - Patterns
- Reviews

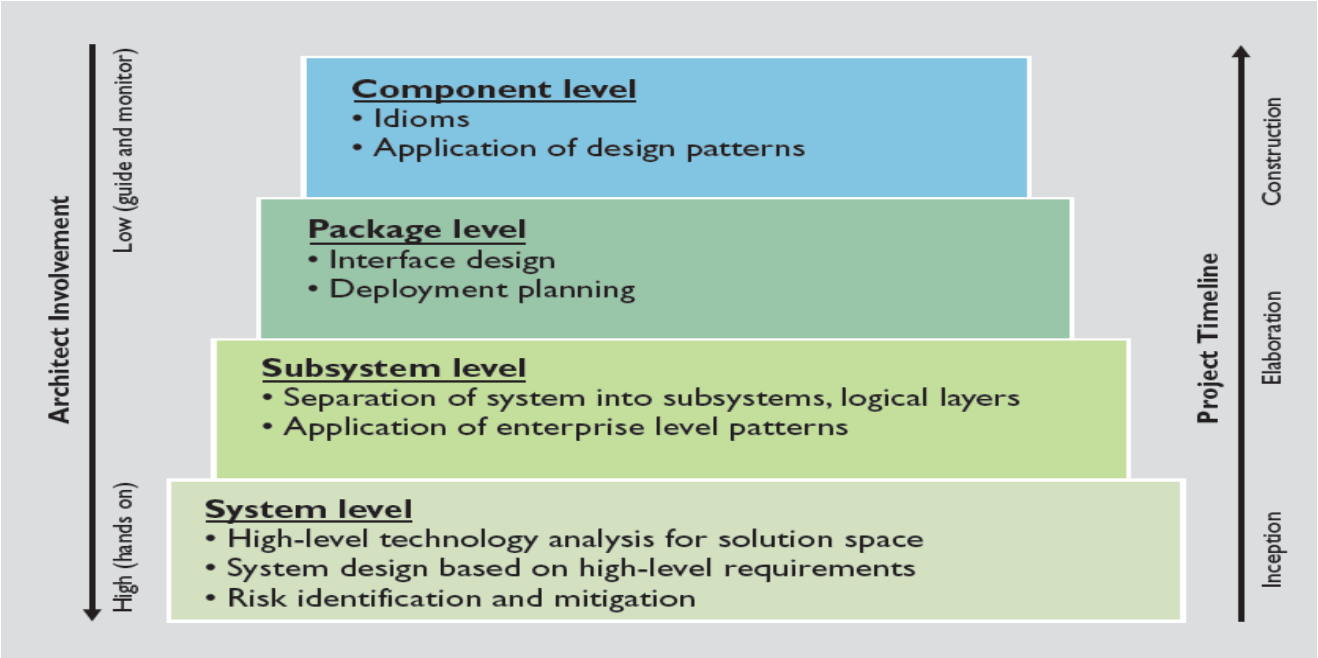
What do architects rely on

Types of IT architecture





Architects are technically competent system-level thinkers, guiding planned and economically efficient design processes to bring a system into existence.



Architects proactively focus on system- and subsystem-level issues to establish a solid foundation for detailed design, particularly for large-scale efforts.

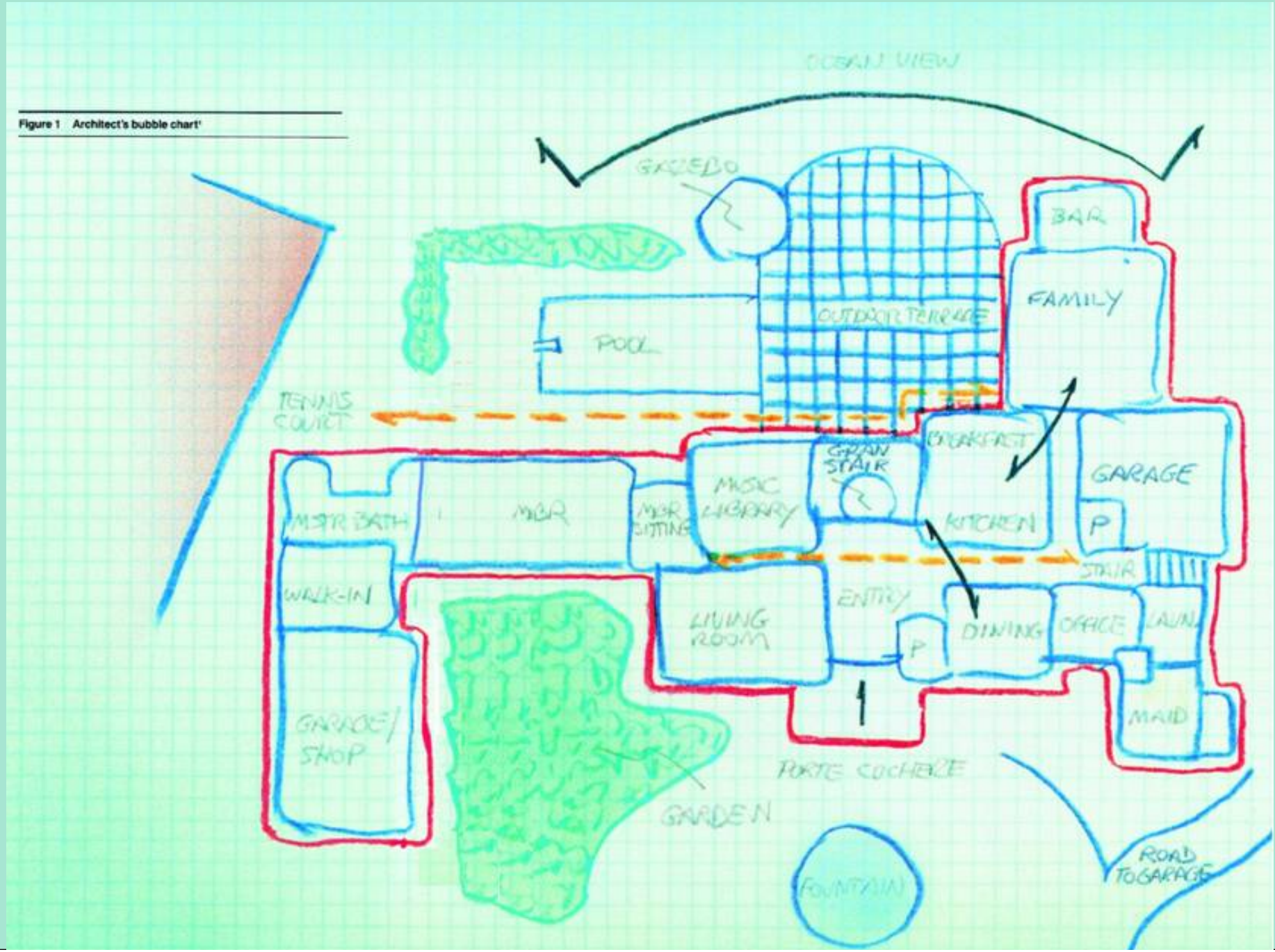
Architects must lead multiple stakeholders in a technologically challenging and sometimes politically charged environment.

The software architect
Matthew R. McBride
Communications of the ACM
Volume 50, Number 5 (2007), Pages 75-81

An example of a conversation between the architect and the prospective owner. Each question serves to pose a constraint (the lot size) or identify a requirement (number of bedrooms) in order to establish the conditions whithin which any design will take place.

"I'd like to build a building."
"What kind of building do you have in mind? Do you plan to sleep in it? Eat in it? Work in it?"
"Well, I'd like to sleep in it."
"Oh, you want to build a house?"
"Yes, I'd like a house."
"How large a house do you have in mind?"
"Well, my lot size is 100 feet by 300 feet."
"Then you want a house about 50 feet by 100 feet?"
"Yes, that's about right."
"How many bedrooms do you need?"
"Well, I have two children, so I'd like three bedrooms."

Figure 1 Architect's bubble chart[1]



Scope description

Ball park view

Model of the business

Customer's view

Model of the Information System

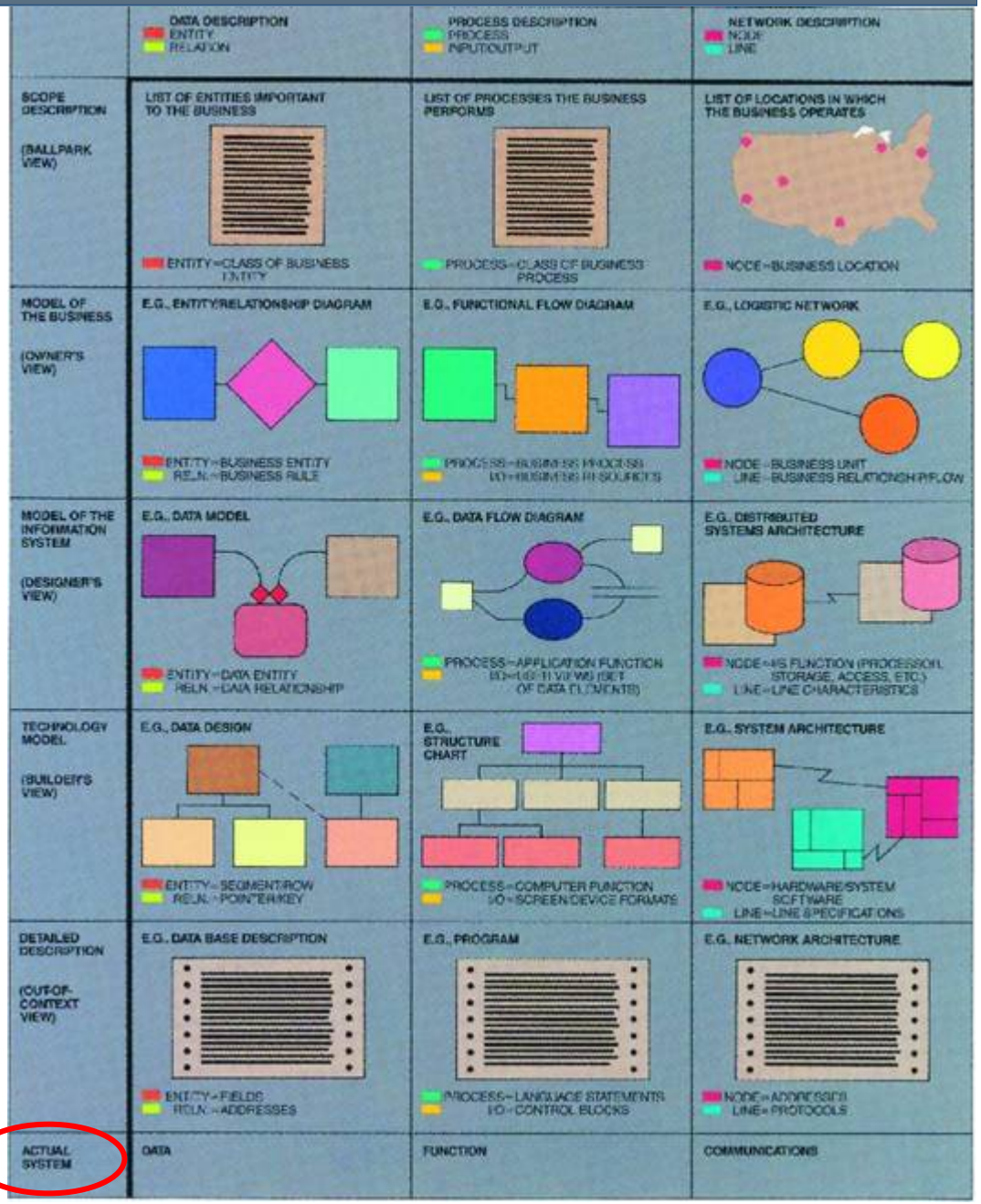
Designer's view

Technology model

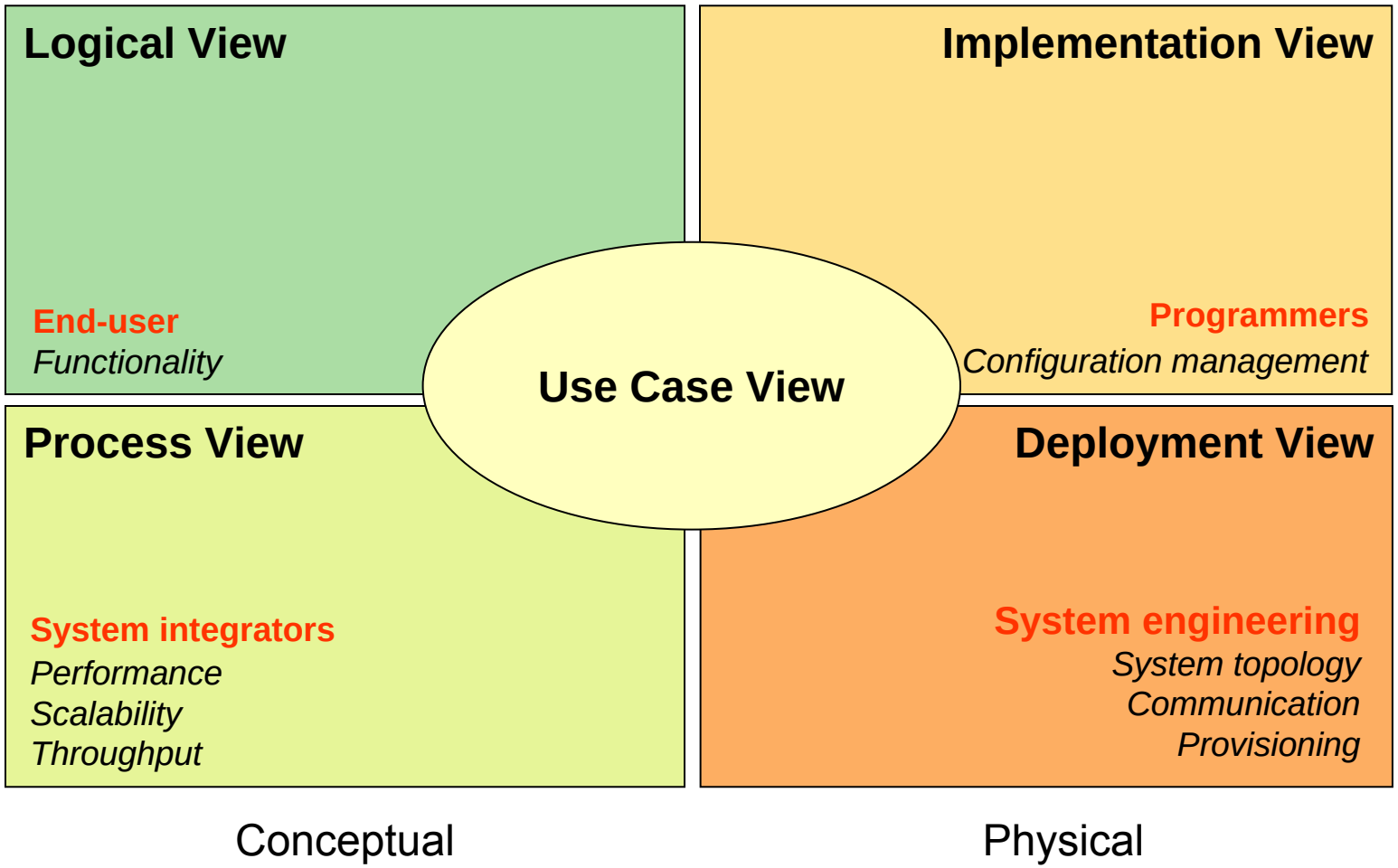
Builder's view

Detailed description

Out-of-context view

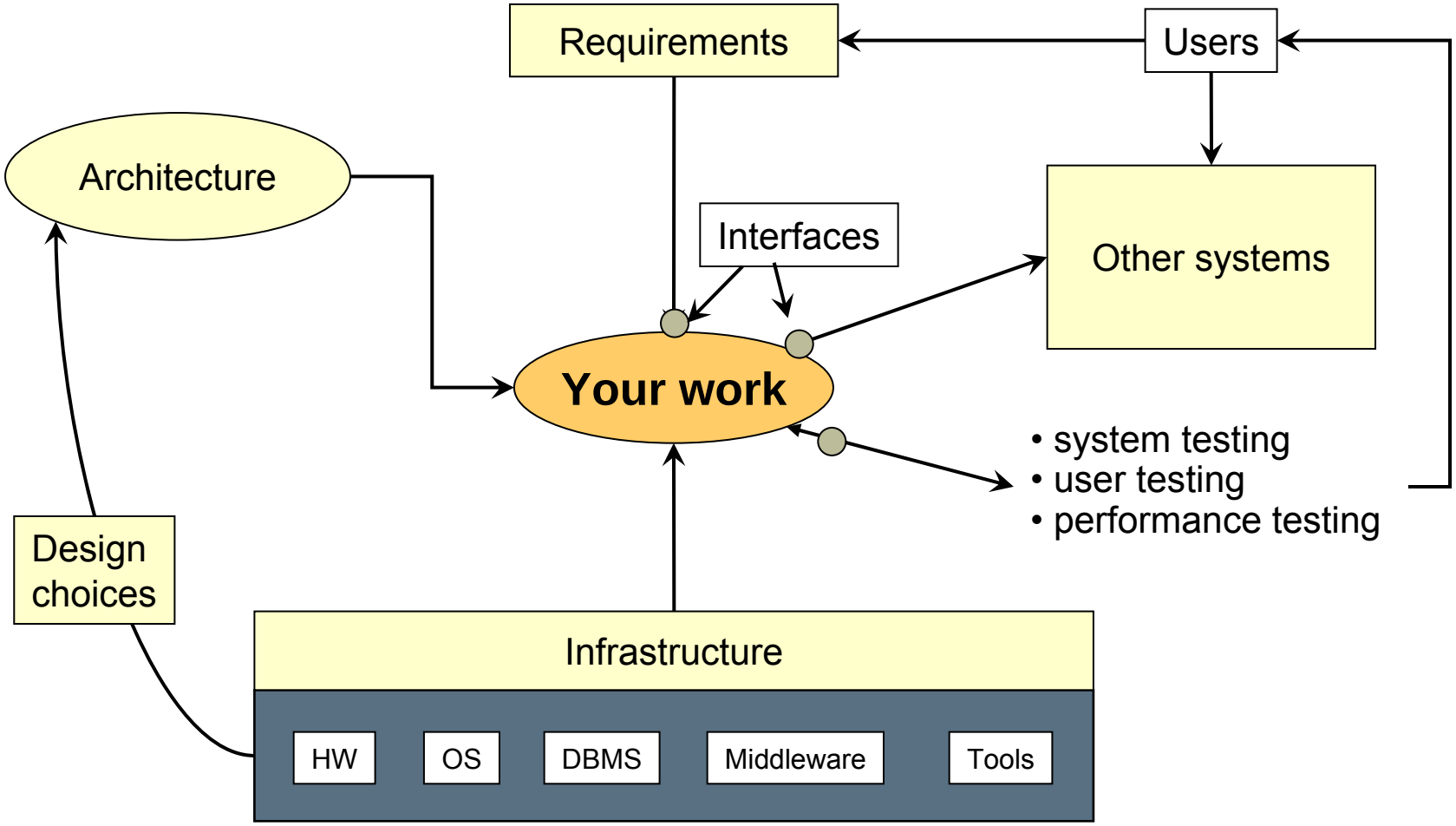


Views for representing Software Architecture

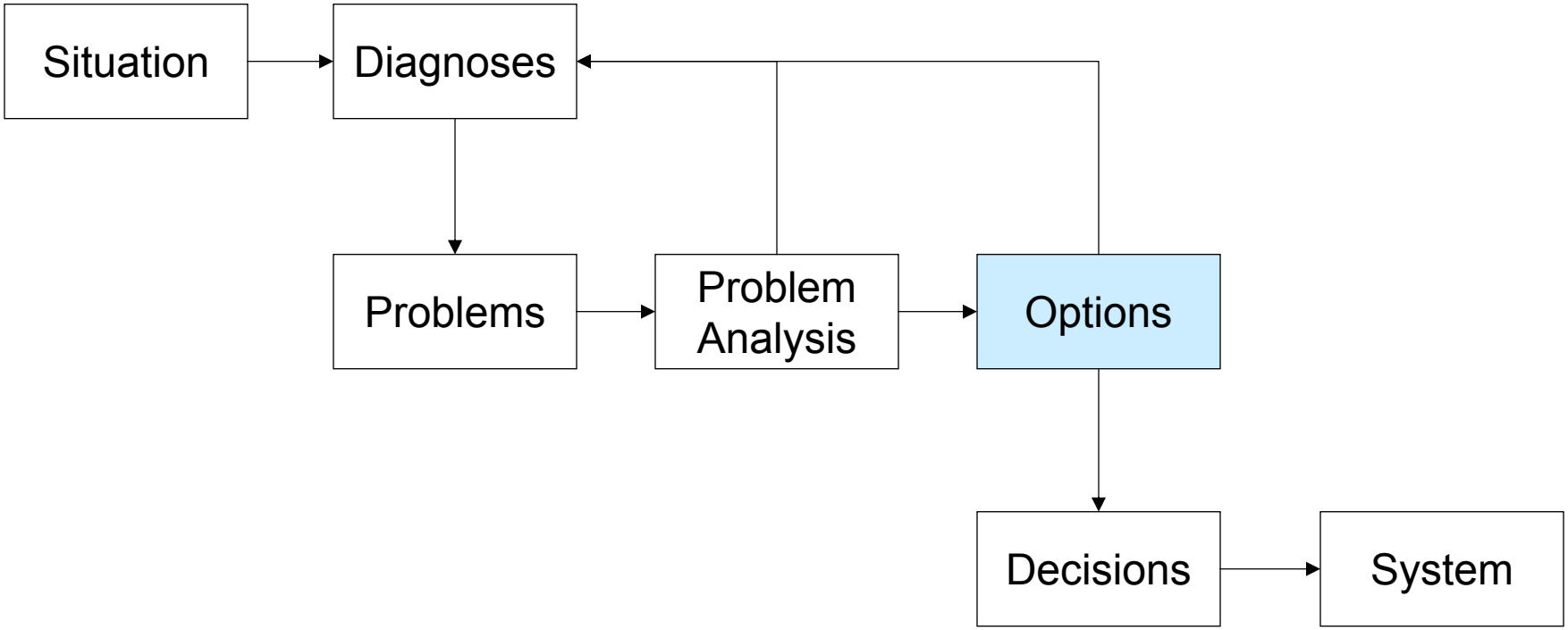


Kruchten, "The 4+1 Model View"

The wider context



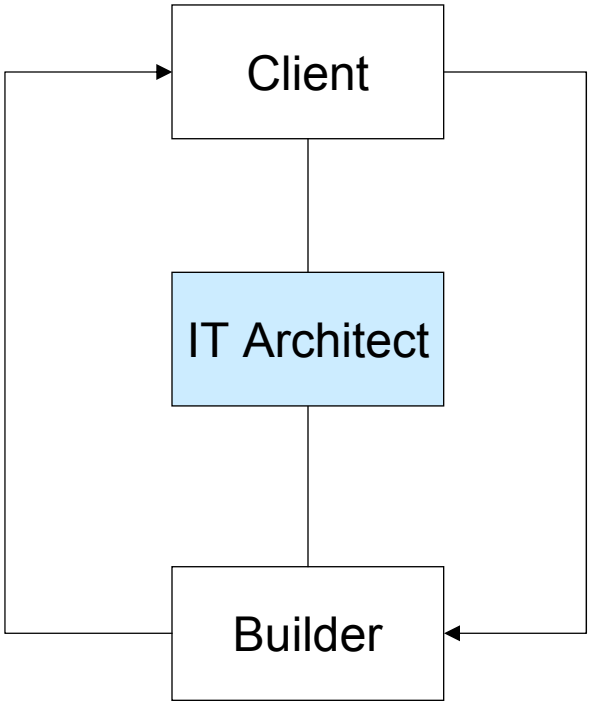
The Role of the IT Architect



- The architect should not focus on some separate part, called the „architecture“
- The architect should assume the **responsibility** that an engineered system is optimally matched to the situation
- The architect is the author of the solution, undeniably **accountable** for the effort‘s success or failure.

Options: Builder Nominates, Customer decides
Prioritizing: Trade-offs, Options, and Choices

- During initial requirements gathering, the architect established important **baselines**.
- Unstated **expectations** for the design must be identified and validated.
- Requirements may need to be **altered, added, or deleted** to deliver an optimal solution.
- The choice between architectures may well depend upon which set of **drawbacks** the client can handle best.
- If **trade-off** results are inconclusive, then the wrong selection criteria were used. Find out [again] what the customer wants and why they want it, then repeat the trade-off using those factors as the [new] selection **criteria**.
- The architect must transcend the limitations of the builder's state of the art, and **imagine what is possible, given time and budget constraints**.



Common Traits of Trusted Advisors

Seen from the client's perspective

You don't get the chance to employ advisory skills until you can get someone to trust you enough to share their problems with you.

- Understand us, and like us
- Don't try to force things on us
- Give us **options**, increase our understanding of those options, give us their recommendations, and let us chose
- Help us think things through (it's our decision)
- Help us think and separate our logic from our emotion
- Give us **reasoning** (help us think), not just **conclusions**
- Help us to put our issues in context, through the use of metaphors, stories, and anecdotes (few problems are completely unique)
- Challenge our assumptions (help us uncover the false assumptions we've been working under)
- Criticize and correct us gently, lovingly
- We can rely on them to tell us the truth
- Are consistent (we can depend on them)

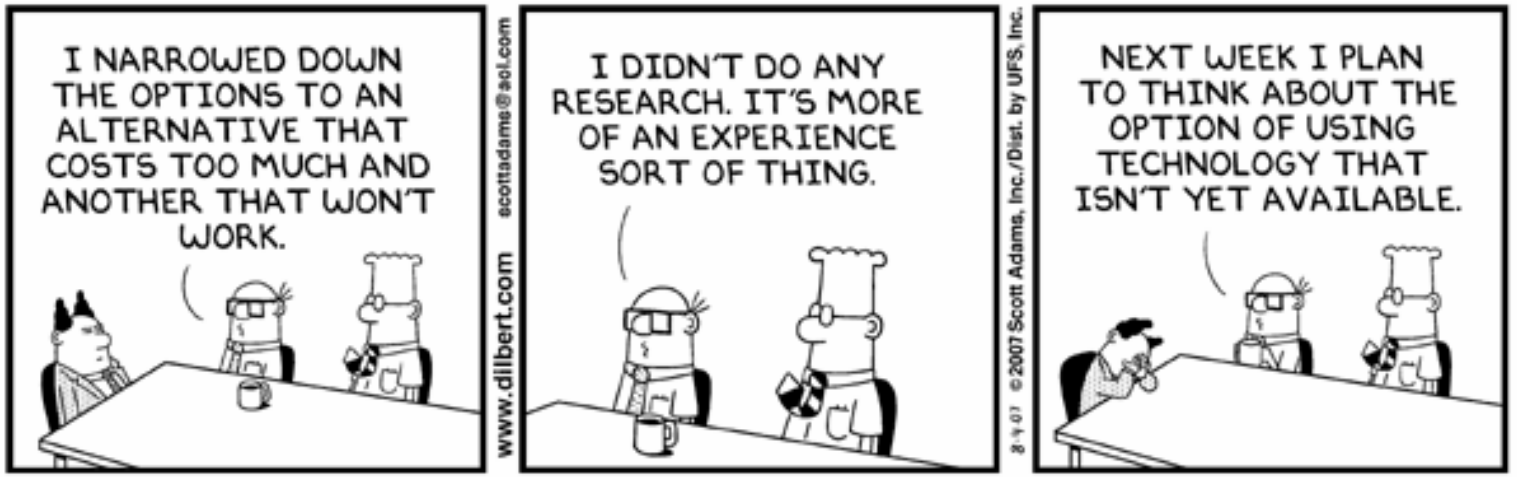
D. Maister, C. Green, R. Galford, The trusted Advisor, The Free Press, 2000.

Take a point of view (POV)

- It is useful to our clients if we articulate a Point Of View, even if it ends up being rejected or wrong.
- Two reasons:
  - It stimulates reactions
  - It crystallizes issues
- Stating a POV serves as a catalyst, a way of helping the client think
- Learn to express a POV with a simple, phrase such as:
  - Now let me just float a trial balloon here
  - Hey, who knows where this might go, but it occurs to me that ...

D. Maister, C. Green, R. Galford, The trusted Advisor, The Free Press, 2000.

Trusted Advisor and Architect Antipatterns



?

- Respond to the minutiae of an RFP rather than aggressively work relationships to manage the bigger picture with clients.
  - Risk of missing the big picture, leaving clients unconvinced that the architect's company has the capability to do anything other than implementation.
- Give the client a car when they needed a bike
  - Over-solutioning and building up complexity are a sure way to lose.

Architecting vs. Engineering

- Architecting is
  - working ***for*** a client and ***with*** a builder
  - helping determine relative requirement **priorities**, acceptable performance, cost, and schedule
  - taking into account such factors as technology **risk**, projected market size, likely competitive moves, economic trends, political regulatory requirements, project organization, and the appropriate "**illities**" (availability, operability, manufacturability, survivability, etc.)
- Engineering is
  - working ***with*** an architect and ***for*** the builder
  - applying the best engineering practices to assure **compliance** at the system level with the designated architecture and with applicable specifications, standards, and contracts.

The essence of systems is relationships, interfaces, form, fit, and function.
The essence of architecting is structuring, simplification, compromise, and balance.
The challenge is control, if not the reduction of complexity and uncertainty.

The design of complex systems must blend the art of architecture with the science of engineering

Characteristics of the Architect's Job

- No person who is not a great sculptor or painter can be an architect. If he is not a sculptor or painter, he can only be a builder.
  - John Ruskin, Lectures on Architecture and Painting (1853)
- Architect is the principal IT leader
  - In large organizations
  - In large and small projects
- Visioning and modeling
  - Bridging the Business-IT gap
  - Vocabulary, Language → Architecture Description Standard
  - Method to support Architectural Thinking
- Overseeing construction, to ensure actual implementation meets design
- Responsibility for acceptance of built system
- Multidisciplinary Synthesis
  - Technical, programmatic, managerial
  - **Heuristic**

Heuristics = Tool store for systems architecting

Bag of tricks and tools that are largely experience-based and form the intellectual framework architects use to guide decisions on a day-to-day basis

A **heuristic** is a replicable method or approach for directing one's attention in learning, discovery, or problem-solving. Originally derived from the Greek "heurisko" (εὑρίσκω), which means "I find".

The art of systems architecting / edited by Mark W. Maier, Eberhardt Rechtin.—2nd ed. Appendix A: Heuristics for systems-level architecting

- Multitask heuristics
- Scoping and planning
- Modeling
- Prioritizing (trades, options, and choices)
- Aggregating ("chunking")
- Partitioning (decompositioning)
- Integrating
- Certifying (system integrity, quality, and vision)
- Assessing performance, cost, schedule, and risk
- Rearchitecting, evolving, modifying and adapting

Not all heuristics apply to all circumstances, just most to most.

Some Heuristics

A **heuristic** is a replicable method or approach for directing one's attention in learning, discovery, or problem-solving. It is originally derived from the Greek "heurisko" (εὑρίσκω), which means "I find".

- Don’t assume that the original statement of the problem is necessarily the best, or even the right, one.
- Extreme requirements, expectations, and predictions should remain under challenge throughout system design, implementation, and operation.
- Explore the situation from more than one point of view. A seemingly impossible situation might suddenly become transparently simple.
- Success is defined by the client, not by the architect.
- The most important single element of success is to listen closely to what the client perceives as his requirements and to have the will and ability to be responsive.

A lot of times, people don't know what they want until you show it to them.
Steve Jobs

The art of systems architecting / edited by Mark W. Maier, Eberhardt Rechtin.—2nd ed. Appendix A: Heuristics for systems-level architecting

Don’t assume that the original statement of the problem is necessarily the best, or even the right one.

Security is limited by the weakest link

More Heuristics

A **heuristic** is a replicable method or approach for directing one's attention in learning, discovery, or problem-solving. It is originally derived from the Greek "heurisko" (εὑρίσκω), which means "I find".

- Ask early about how you will evaluate the success of your efforts.
- For a system to meet its acceptance criteria to the satisfaction of all parties, it must be architected, designed, and built to do so — no more and no less.
- Define how an acceptance criterion is to be certified at the same time the criterion is established.
- Regardless of what has gone before, the acceptance criteria determine what is actually built.
- If there are things a successful system or architecture cannot do, or at least not do well -- don't force it!
- The strengths of a system or architecture in one context can be its weaknesses in another. Know when and where.
- An element "good enough" in a small system is unlikely to be good enough in a more complex one.
- A system is successful when the natural intersection of technology, politics, and economics is found.
- High quality, reliable systems are produced by high quality architecting, engineering, design, and manufacture, not by inspection, test, and rework.

Adapted from: The art of systems architecting / edited by Mark W. Maier, Eberhardt Rechtin.—2nd ed. Appendix A: Heuristics for systems-level architecting

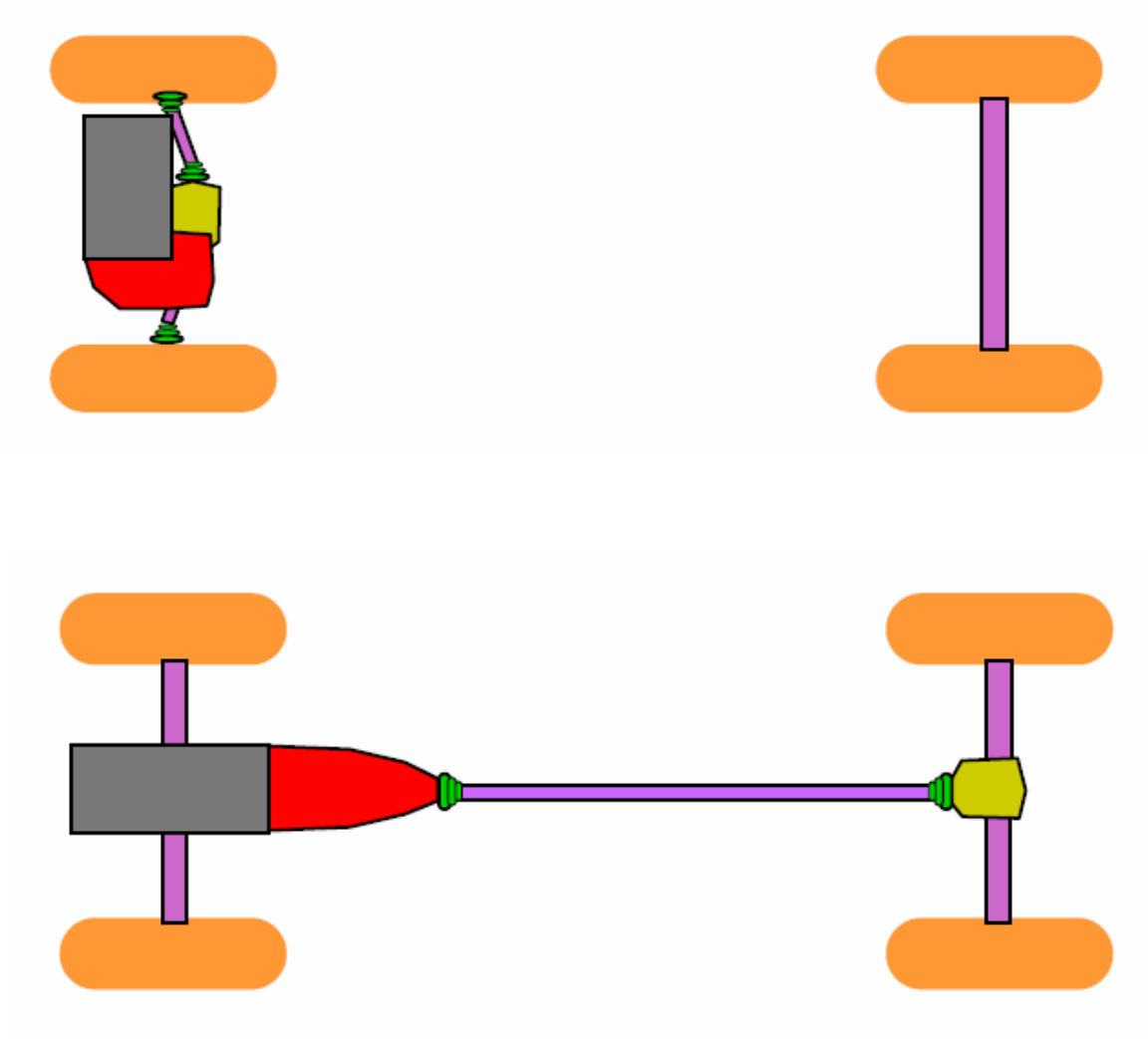
Partitioning (decompositioning) and Integrating

A **heuristic** is a replicable method or approach for directing one's attention in learning, discovery, or problem-solving. It is originally derived from the Greek "heurisko" (εὑρίσκω), which means "I find".

- Do not slice through regions where high rates of information exchange are required.
- The principles of minimum communications and proper partitioning are key to system testability and fault isolation.
  - To be tested, a system must be architected and designed to be tested.
- The greatest leverage in architecting is at the interfaces.
- The greatest dangers are also at the interfaces.
- Logical layering and careful interface definition improve the overall design.
- Guidelines for a good quality interface specification: they must be simple, unambiguous, complete, concise, and focus on substance.
- It is inadequate to architect up to the boundaries or interfaces of a system; one must architect across them.
- Relationships among the elements are what give systems their added value.
- Be sure to ask the question: “What is the worst thing that other elements could do to you across the interface?

The art of systems architecting / edited by Mark W. Maier, Eberhardt Rechtin.—2nd ed. Appendix A: Heuristics for systems-level architecting

Front and Rear Wheel Drive Architectures
Same elements, arranged differently



Aggregating (“chunking”)

A **heuristic** is a replicable method or approach for directing one's attention in learning, discovery, or problem-solving. It is originally derived from the Greek "heurisko" (εὑρίσκω), which means "I find".

- Group elements that are strongly related to each other, separate elements that are unrelated.
- Subsystem interfaces should be drawn so that each subsystem can be implemented independently of the specific implementation of the subsystems to which it interfaces.
- Choose a configuration with minimal communications between the subsystems.
- Choose the elements so that they are as independent as possible; that is, elements with low external complexity (low coupling) and high internal complexity (high cohesion). (Christopher Alexander, 1964 modified by Jeff Gold, 1991)
- Never aggregate systems that have a conflict of interest; partition them to ensure checks and balances.
- Aggregate around “testable” subunits of the product; partition around logical subassemblies.
- Iterate the partition/aggregation procedure until a model consisting of 7 ± 2 chunks emerge.
- System structure should resemble functional structure.

The art of systems architecting / edited by Mark W. Maier, Eberhardt Rechtin.—2nd ed. Appendix A: Heuristics for systems-level architecting

Things to keep in mind: Partitioning and Aggregating

- Why architects do it
  - To simplify - break a complex problem into smaller ones that are more easily solved
  - For speed - the sub-problems can be solved in parallel
- The challenges
  - finding the most suitable set of sub-problems
  - combining the separate subsystems into an overall solution may be difficult.
- Basic approach
  - Decompose the problem into basic elements
  - Analyze the interactions between the elements
  - Aggregate the elements into logical groups or partitions

These are critical!!

- Considerations
  - Use and Re-use
    - Commercial software packages
    - Re-use libraries
  - Organizational
    - End-user oriented
    - Development & delivery oriented
  - Integration & certification approach
    - Plan for subsystem and system integration & test
    - Qualification and required supporting environment

Examples:

- In partitioning, choose elements so that they are as independent as possible; that is, elements with low external complexity and high internal complexity (or cohesion).
- Group elements that are strongly related to one another, separate elements that are unrelated.

Partitioning focus: Developing and Integrating the System

- Developing:
  - Consider who will be developing each piece of the system
  - Design the partitions such that each developing organization is supplying a whole, separate, testable piece.
  - Minimize the number of “hops” any piece of the system takes before it reaches you
    - Any piece of software should go through only one or two other organizations before it reaches final integration.
  - As you refine the requirements for each piece, note how each should be tested for compliance and acceptance.
    - Is special test data or equipment needed by that development organization? Who will provide it?
- Integrating (Assembling):
  - Consider how the components and elements of the system will be integrated
    - Staged integration
    - Big Bang
  - What requirements will that integration approach drive?
    - Test hooks/modes in the software
    - Integration environments
    - Test environments
    - Special test equipment or test data
    - Interface drivers
  - How will the system be deployed and activated?
    - Any requirements driven by deployment plans?
    - If replacing legacy system, do you need a special mode for parallel operations?

Pay attention to nonfunctional requirements

- Nonfunctional requirements are observable characteristics of the system as a whole
  - While functional requirements are primarily concerned with the stated needs of the problem domain, **nonfunctional requirements represent capabilities that are orthogonal to functional requirements**
- The architect is accountable for ensuring that system performance, i.e. availability, throughput, security and scalability meet user expectations
  - If the architect does not actively address them early on, they could be forgotten or ignored until significant rework must be performed to address them

Murphy's Law: **"If anything can go wrong, it will."**

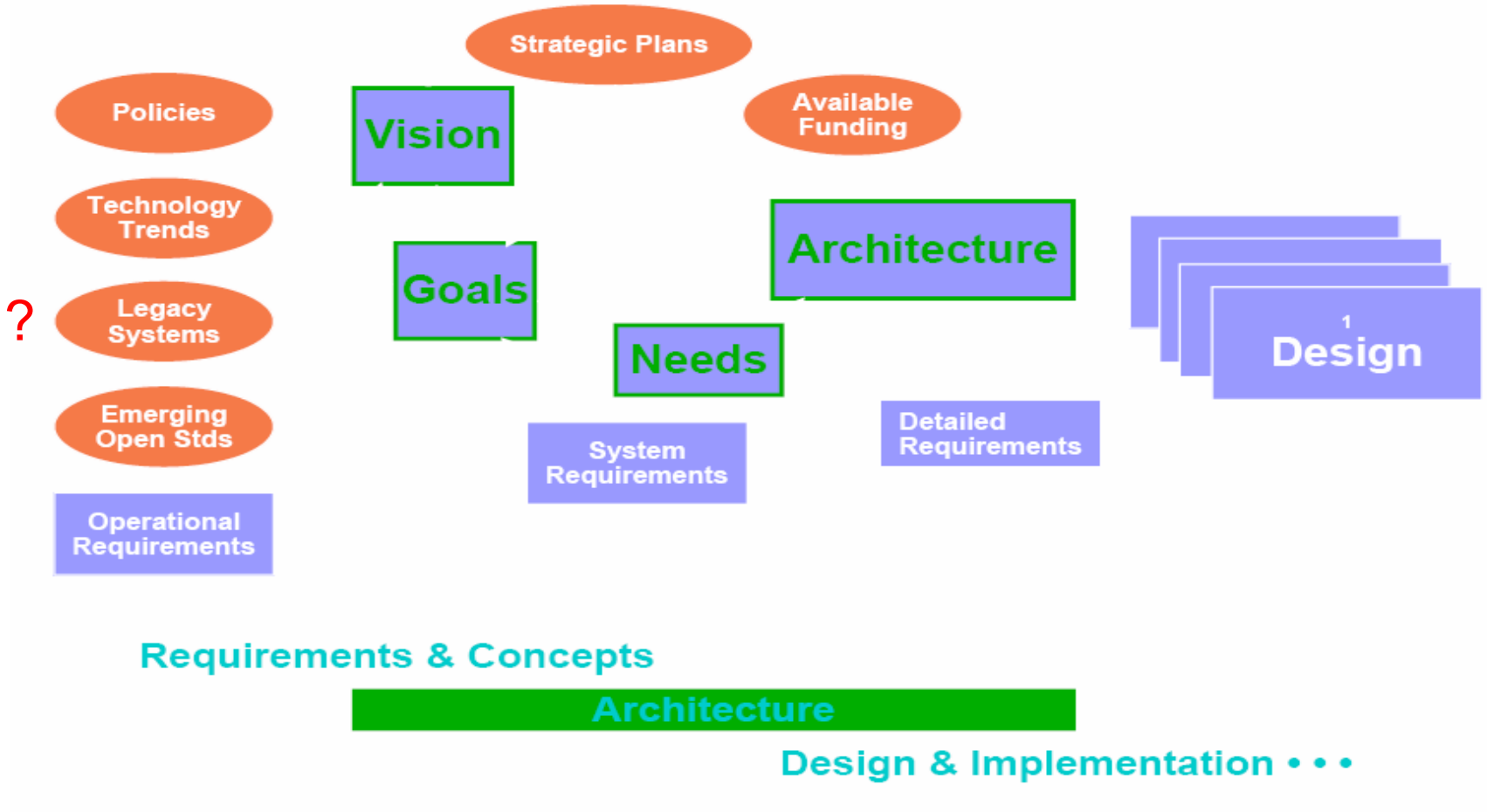
- **The first line of defense against complexity is simplicity**
- Simplify. Simplify. Simplify.
- Simplify, combine, and eliminate.
- **The most reliable part on an airplane is the one that isn't — because it isn't needed. [DC-9 Chief Engineer, 1989]**
- Simplify with smarter elements.
- **If you can't explain it in five minutes, either you don't understand it or it doesn't work.**
- Next to interfaces, the greatest leverage in architecting is in aiding the recovery from, or exploitation of, deviations in system performance, cost, or schedule.

There is a class of problems better avoided than solved

Prevention is better than cure, in particular if the illness is unmastered complexity, for which no cure exists.

E. W. Dijkstra, „The Tide, not the Waves", in „Beyond Calculation", P.J. Denning, R.M. Metcalfe (ed.), Springer, 1997.

The Architect's Job: Visioning and Separating Concerns Amid Complexity

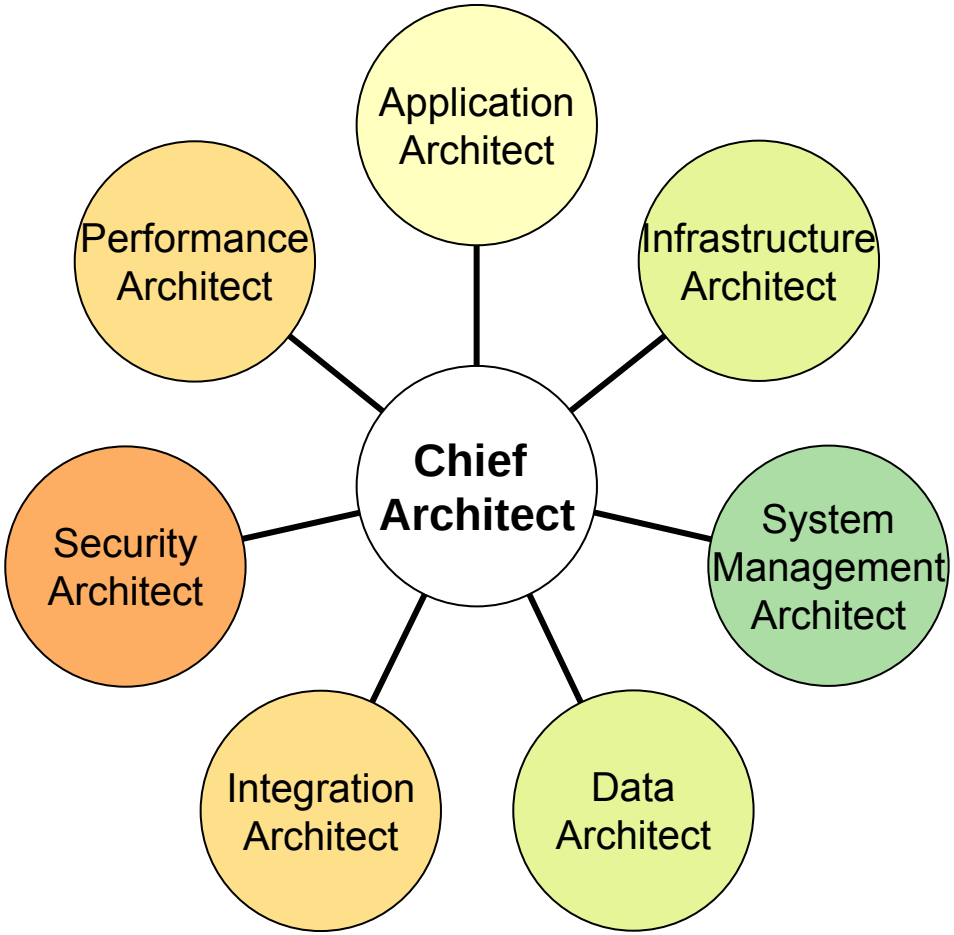


Managing complexity is a key role of the architect

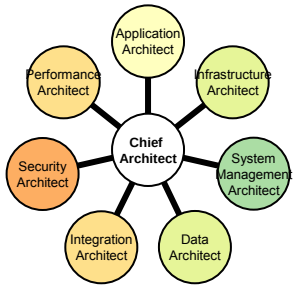
Take a fresh look -- Sometimes, complexity is just a point of view.

- The solar system became no less complex when Copernicus proclaimed that the Earth revolved around the Sun, rather than the reverse.
- But this description made it easier to explain natural phenomena and make predictions.
- This eliminated barriers to imagination and creativity that had stood for most of history.
- Objectively, there is no right or wrong view, but simpler models tend to be more useful.

Architect Job Roles → The design authority team



Design Authority – Two basic roles



Technical Accountability

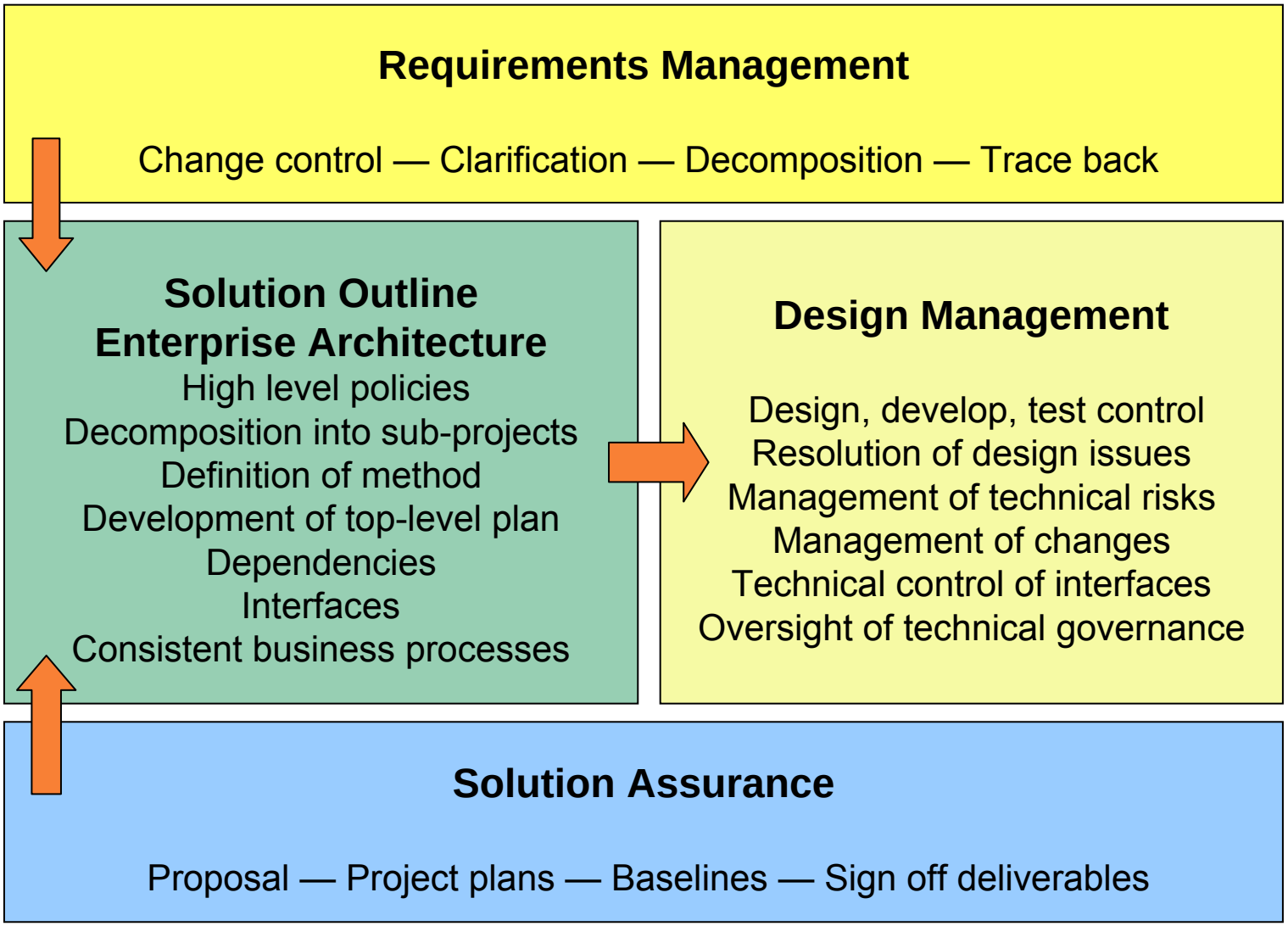
Will it work?

Design Authority

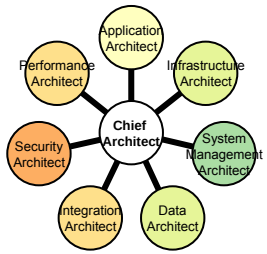
What do we do?

Technical Control

The design authority role – under the leadership of the Chief Architect

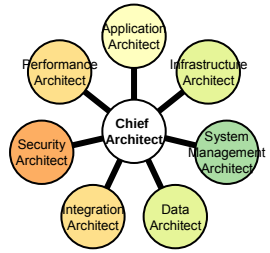


Chief Architect Role



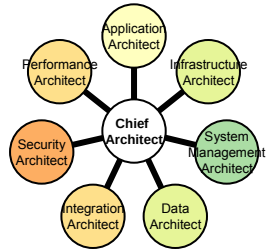
- Carries **technical responsibility**
  - Maintains **overall vision**
  - Needs to carry the whole scope and solution in his/her head
- **Leads** Design Authority staff from technical perspective
- Key roles:
  - Explaining technical issues to management
  - Technical management of Requirements, Issues, Risks & Changes
  - Definition of the Architectural Principles
  - Ownership of Architecture Overview Work Product
  - Managing **reviews**
    - Work products and deliverables
    - Co-ordinating external reviewers, Quality Assurance
  - Developing relationships with **stakeholders**

Application Architect Role



- Defines what the solution does
- Responsible for the **Functional Aspects** of the system
- Key responsibilities
  - Understands how the business requirements can be met using application software, and defines what **application software packages** and / or **bespoke code** is needed
    - It is likely that the main application software will be existing or chosen early on, but the full application picture is often complex
  - Understands the business applications, their capabilities and limitations
  - Develops and maintains application architectures and strategies and to ensure the design integrity of the application subsystem and that it meets the agreed requirements
  - Defines **high level data flows** between applications
  - Leads any **bespoke application development**
  - Leads the **configuration of the application software**

Infrastructure (or Technical) Architect Role

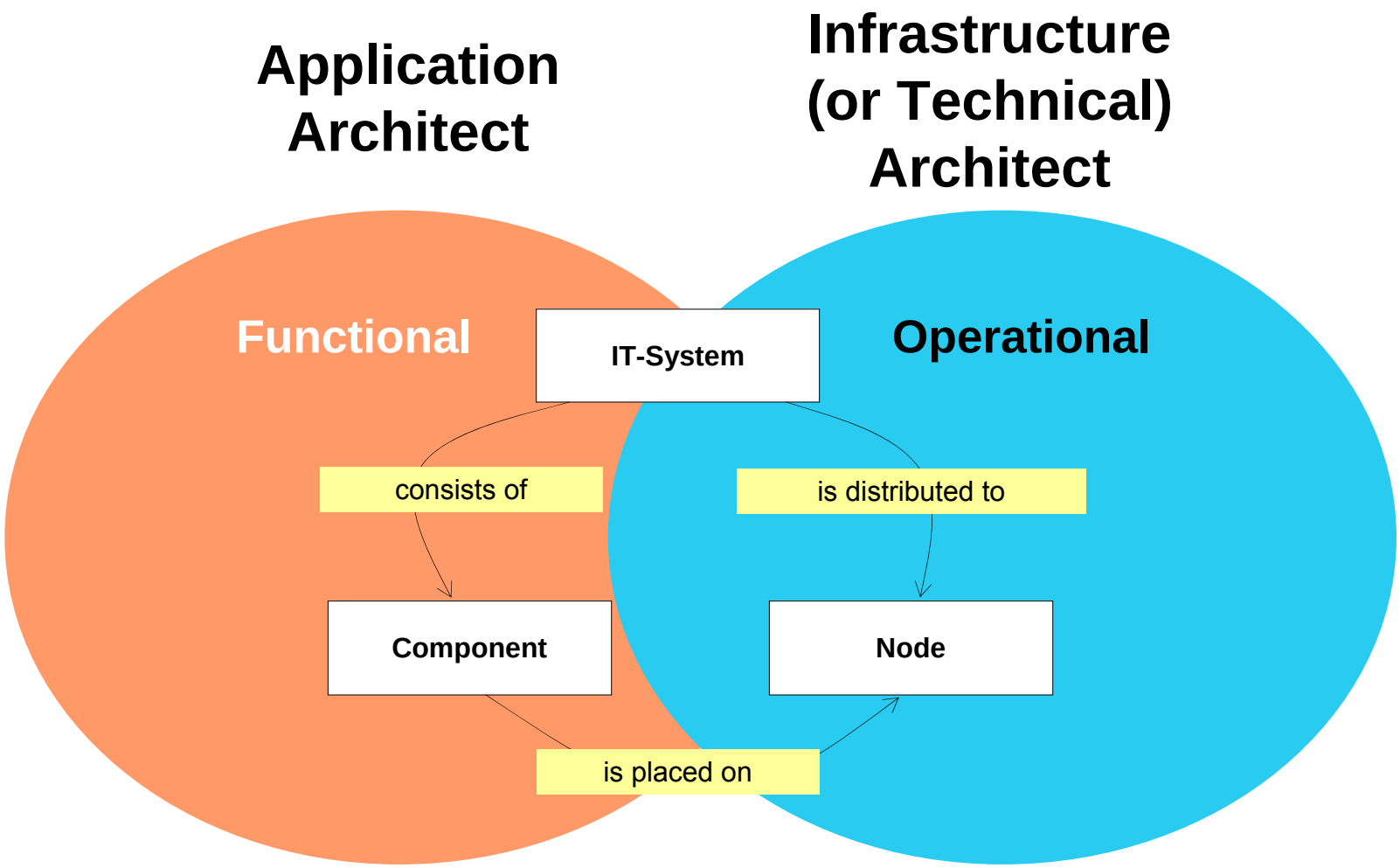


- Defines the overall system shape
  - What the **building blocks** are from which the solution will be made
  - How the data and functionality will be **placed**
- Responsible for the **Operational Aspects** of the system
- Key responsibilities
  - Establishes **non-functional and technical infrastructure requirements**
  - Defines the **infrastructure** solution
    - Networking, hardware configurations, system software, middleware
  - Performance, Capacity, Scalability
  - Availability, Recoverability
  - Systems Management, Service Levels

Non-Functional Requirements

Characteristics of the Architect's Job

- No person who is not a great sculptor or painter can be an architect. If he is not a sculptor or painter, he can only be a builder.
  - John Ruskin, Lectures on Architecture and Painting (1853)
- Architect is the principal IT leader
  - In large organizations
  - In large and small projects
- Visioning and modeling
  - Bridging the Business-IT gap
  - **Vocabulary, Language → Architecture Description Standard**
  - Method to support Architectural Thinking
- Overseeing construction, to ensure actual implementation meets design
- Responsibility for acceptance of built system
- Multidisciplinary Synthesis
  - Technical, programmatic, managerial
  - Artistic, Heuristic



Architectural Description Standard

Meta-Model that describes the concepts that are used to create architectural models, or architectural descriptions

The Application Architect is responsible for the Functional Aspects, which include these key concepts:

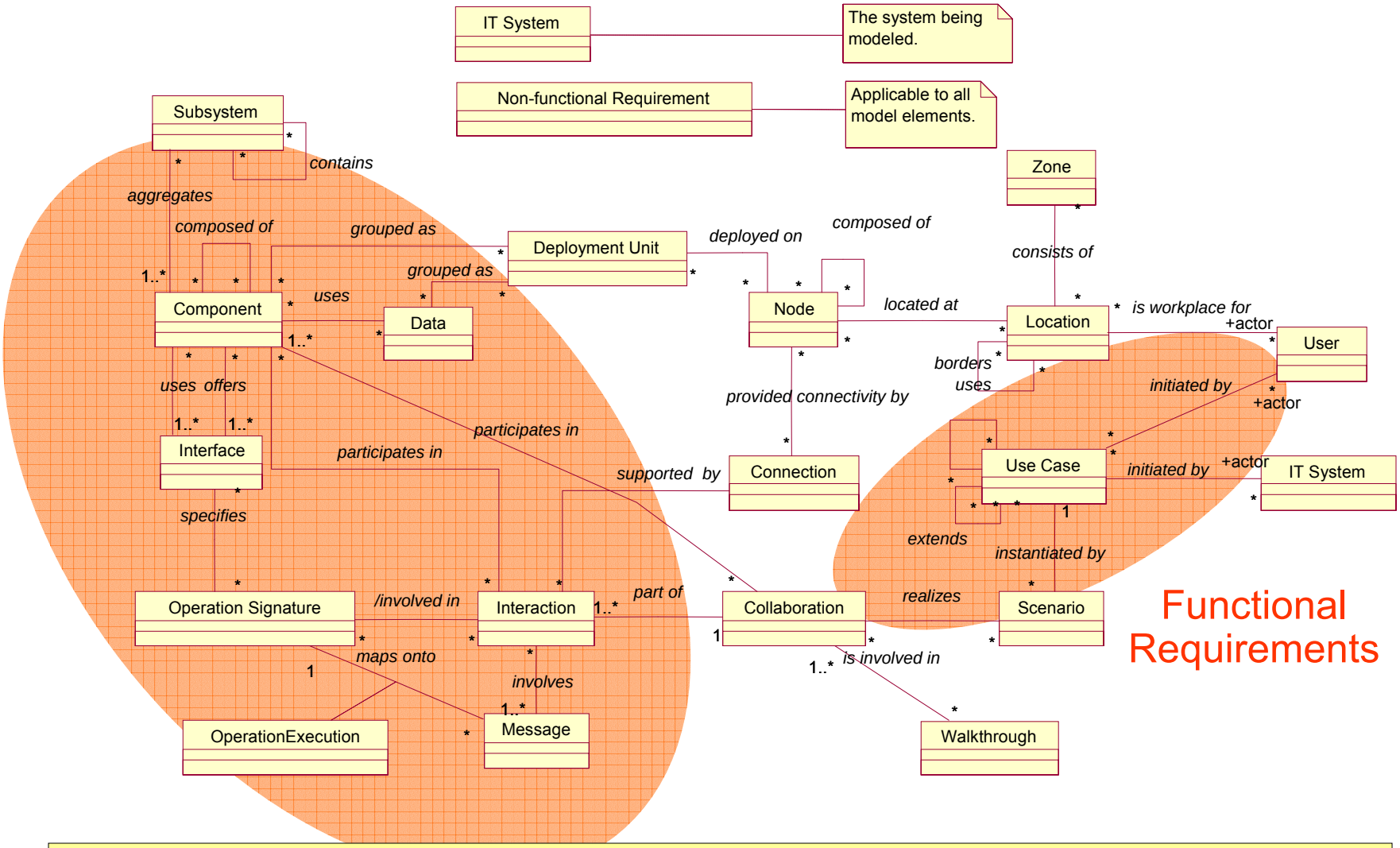
- **Component**
  - modular unit of functionality which makes this functionality available through an interface
- **Subsystem**
  - any grouping of components in IT system
- **Interaction and Collaboration**
  - collaboration between components
  - sequence of component operations
  - exchanges between two components
  - interface usage contract / protocol

Link between Use Cases, and Components

Use Case Realizations

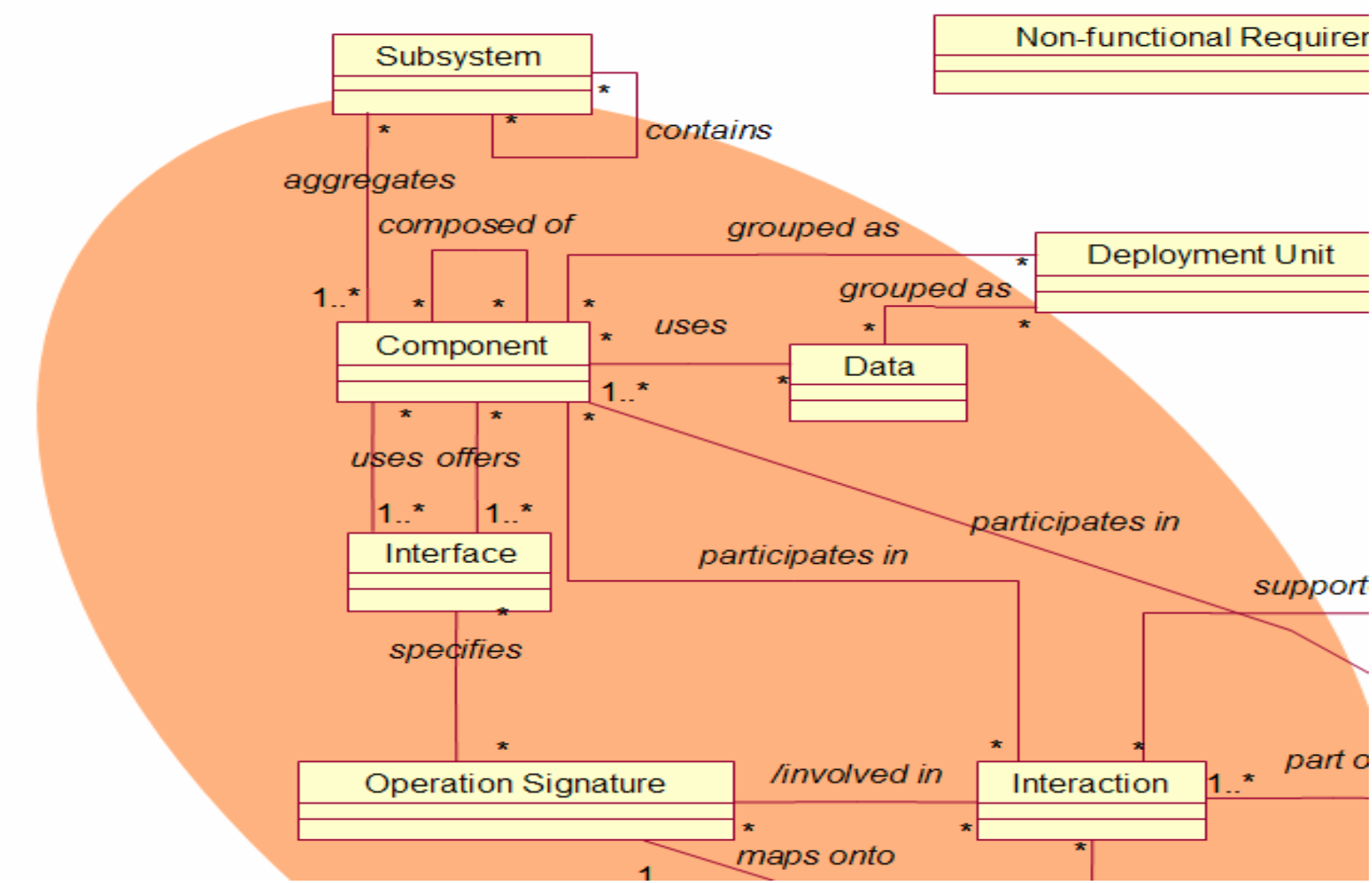
- **Data**

The Application Architect is responsible for the Functional Aspects, which include these key concepts:

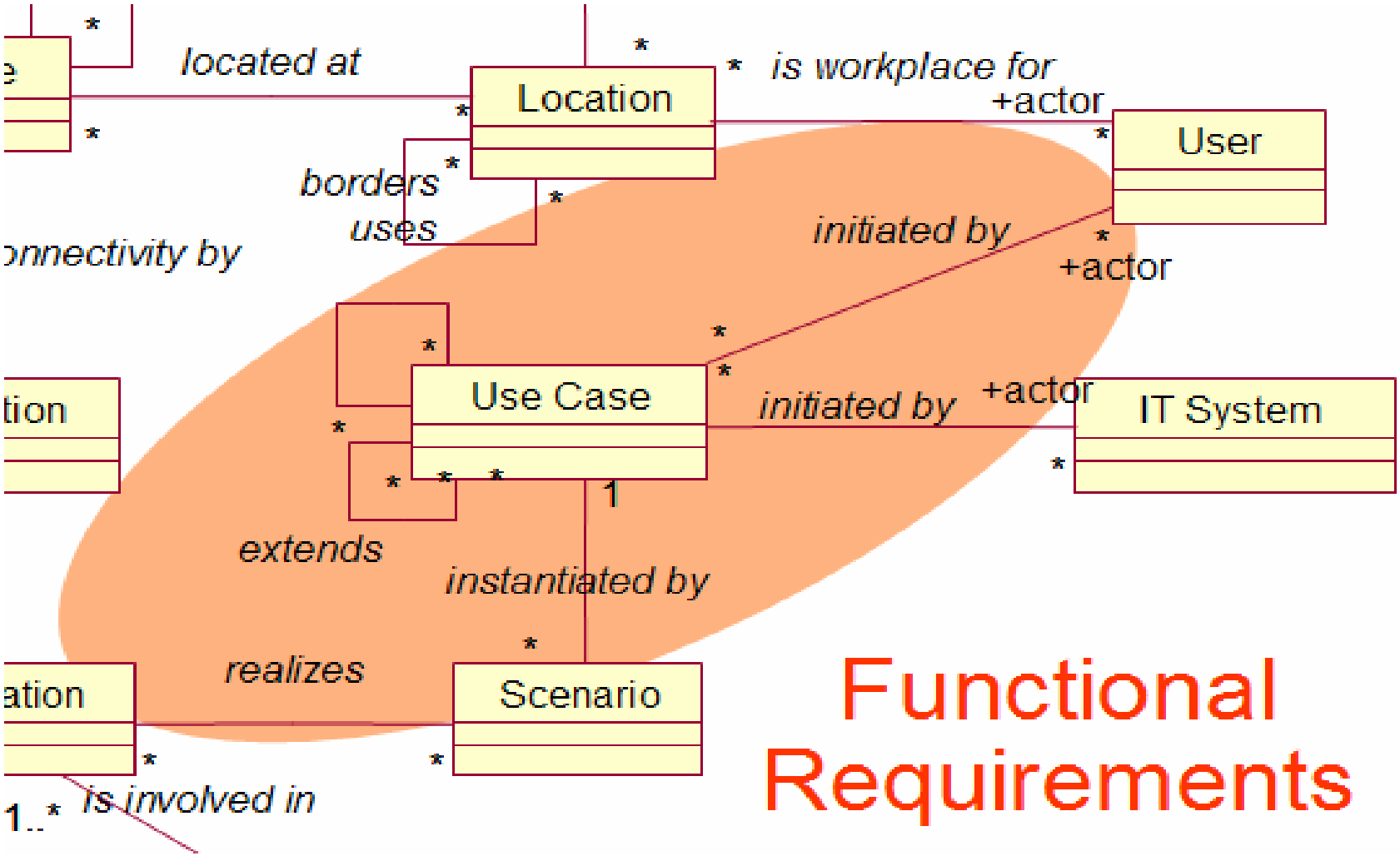


Goal: Define Responsibility for Development, Reuse, Enhancements, etc.

The Application Architect is responsible for the Functional Aspects, which include these key concepts:



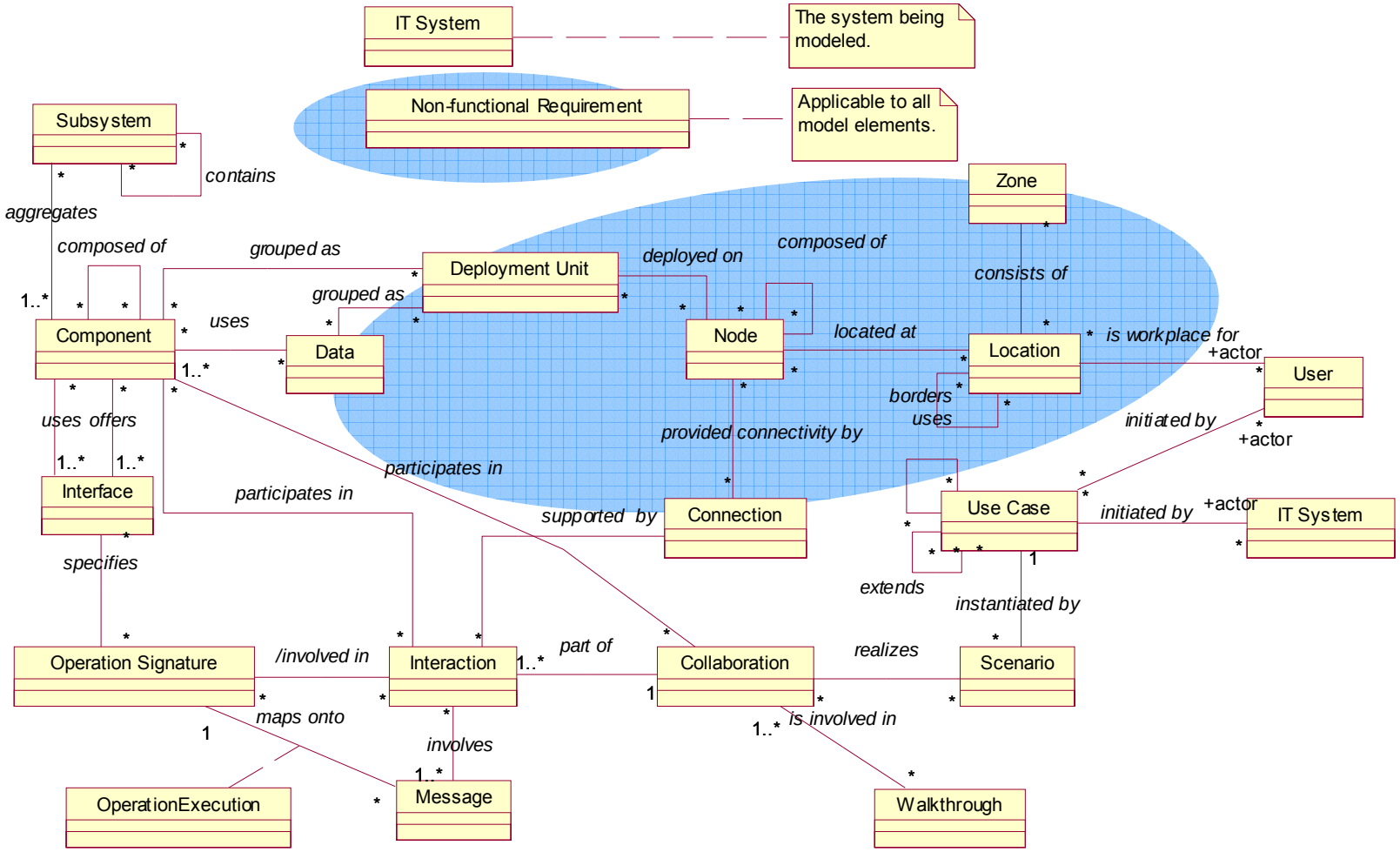
The Application Architect is responsible for the Functional Aspects, which include these key concepts:



The Infrastructure Architect is responsible for the Operational Aspects, which include these key concepts:

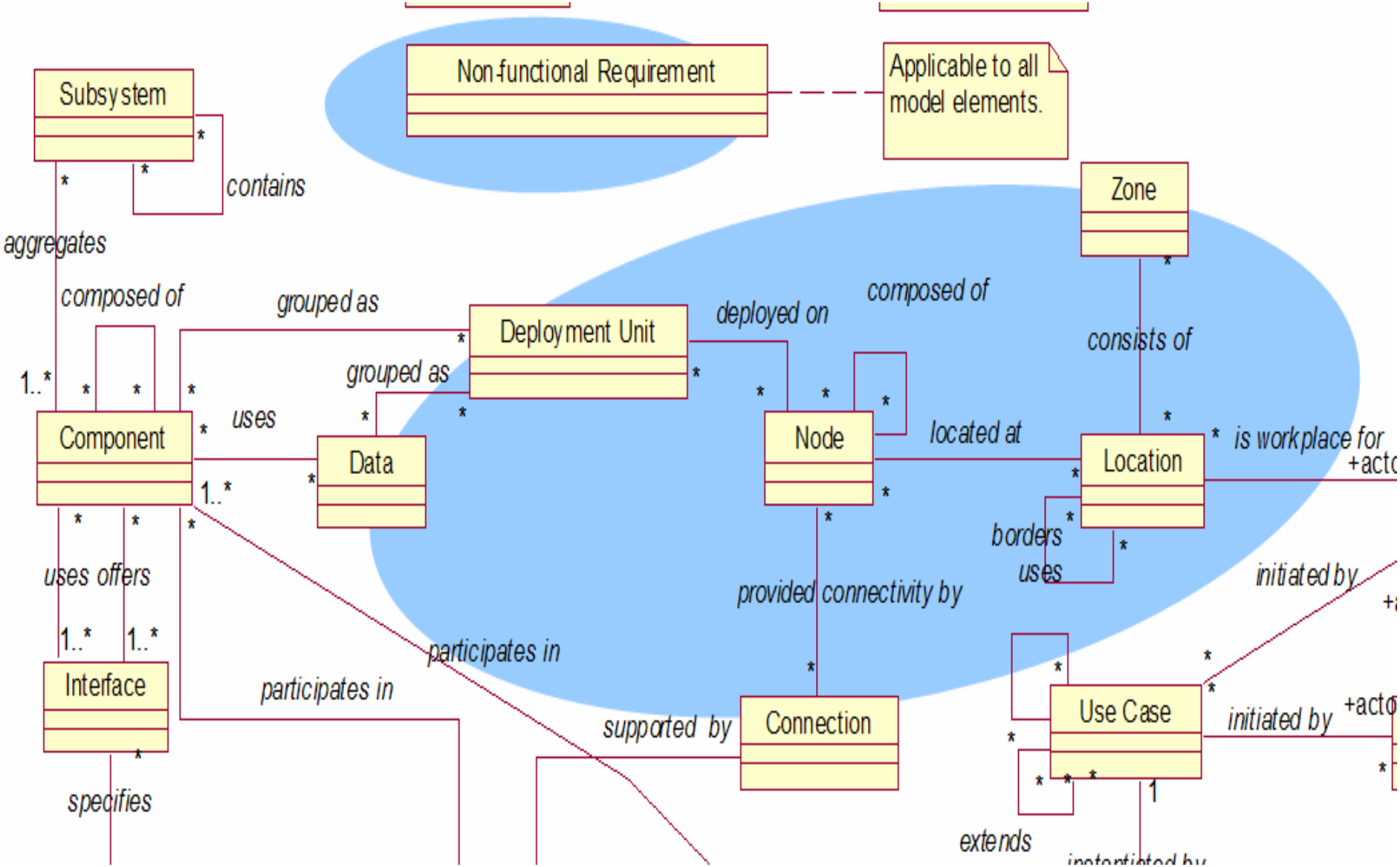
- **Node**
  - platform on which software executes
- **Location**
  - type of geographical area or position
- **Zone**
  - an area for which a common set of non-functional requirements can be defined
- **Connection**
  - physical data path between nodes (LAN, WAN, dial-up etc)
- **Deployment Unit**
  - one or more components placed together on a node
- **Non-functional Requirements (NFRs)**
  - Service Level Requirement (SLR) like performance, availability, etc.
  - Constraints: business / geography, IT Standards, current Infrastructure, etc.
- **Walkthrough**
  - description of the flow of a scenario starting from a user all the way through the system and back to the user

The Infrastructure Architect is responsible for the Operational Aspects, which include these key concepts:

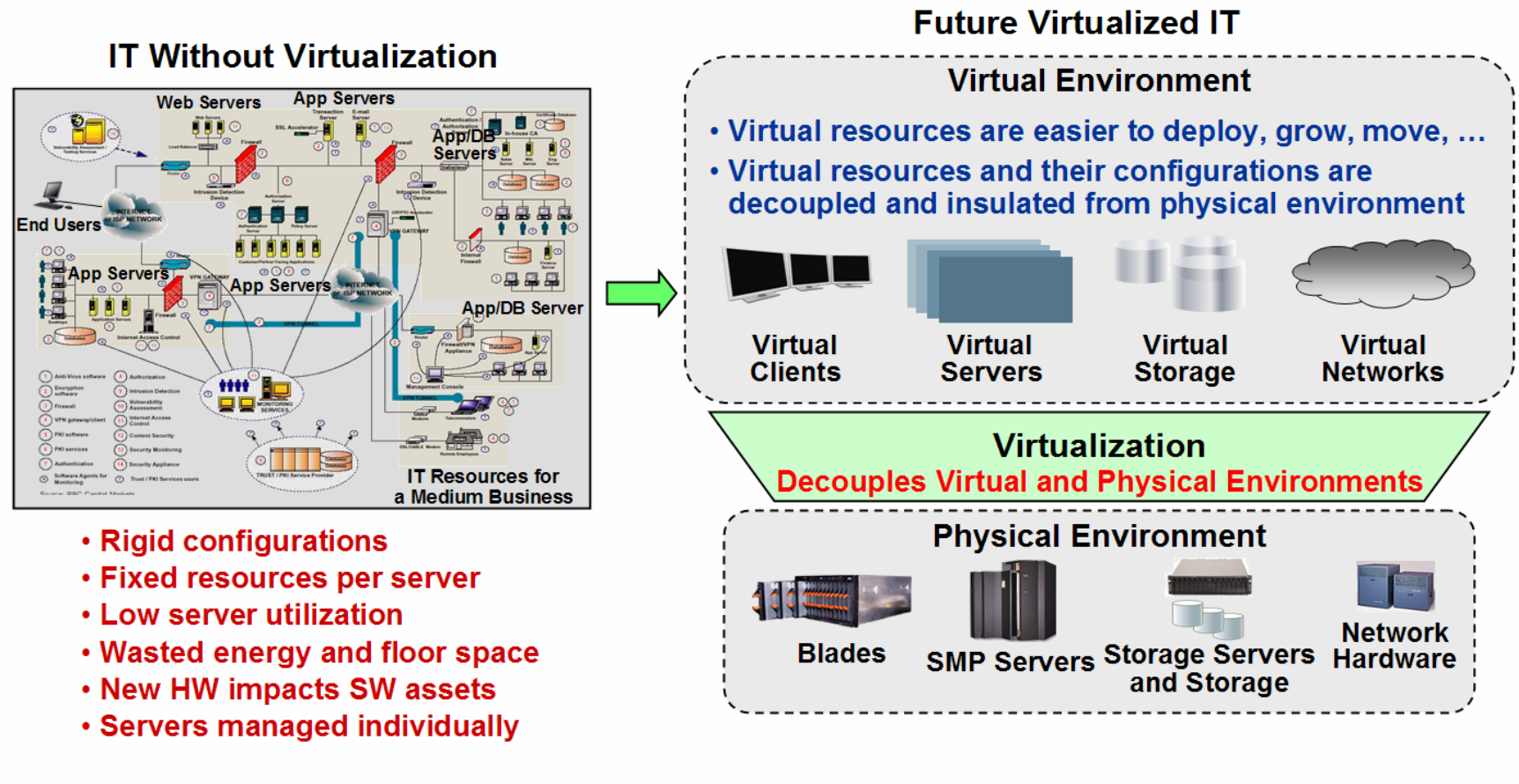


Goal: Fulfill Non-functional Requirements (Service-Level, Manageability, etc.)

The Infrastructure Architect is responsible for the Operational Aspects, which include these key concepts:



IT With Future Virtualization And Mgmt. Software



- A new era is dawning in which virtualization technologies with new management software will significantly reduce IT costs and fulfill “on demand” / SOA needs.
- This will play out via incremental enhancements to existing data centers.

Virtualization Gives Users Idealized Resources

Architected Fishing Interface

Virtual Ice
Has better RAS than real ice

From the Merriam-Webster Online Dictionary:

Main Entry: vir·tu·al
Function: *adjective*
Etymology: Middle English, possessed of certain physical virtues, from Medieval Latin *virtualis*, from Latin *virtus* strength, virtue
1 : being such in essence or effect though not formally recognized or admitted <a *virtual* dictator>

Characteristics of the Architect's Job

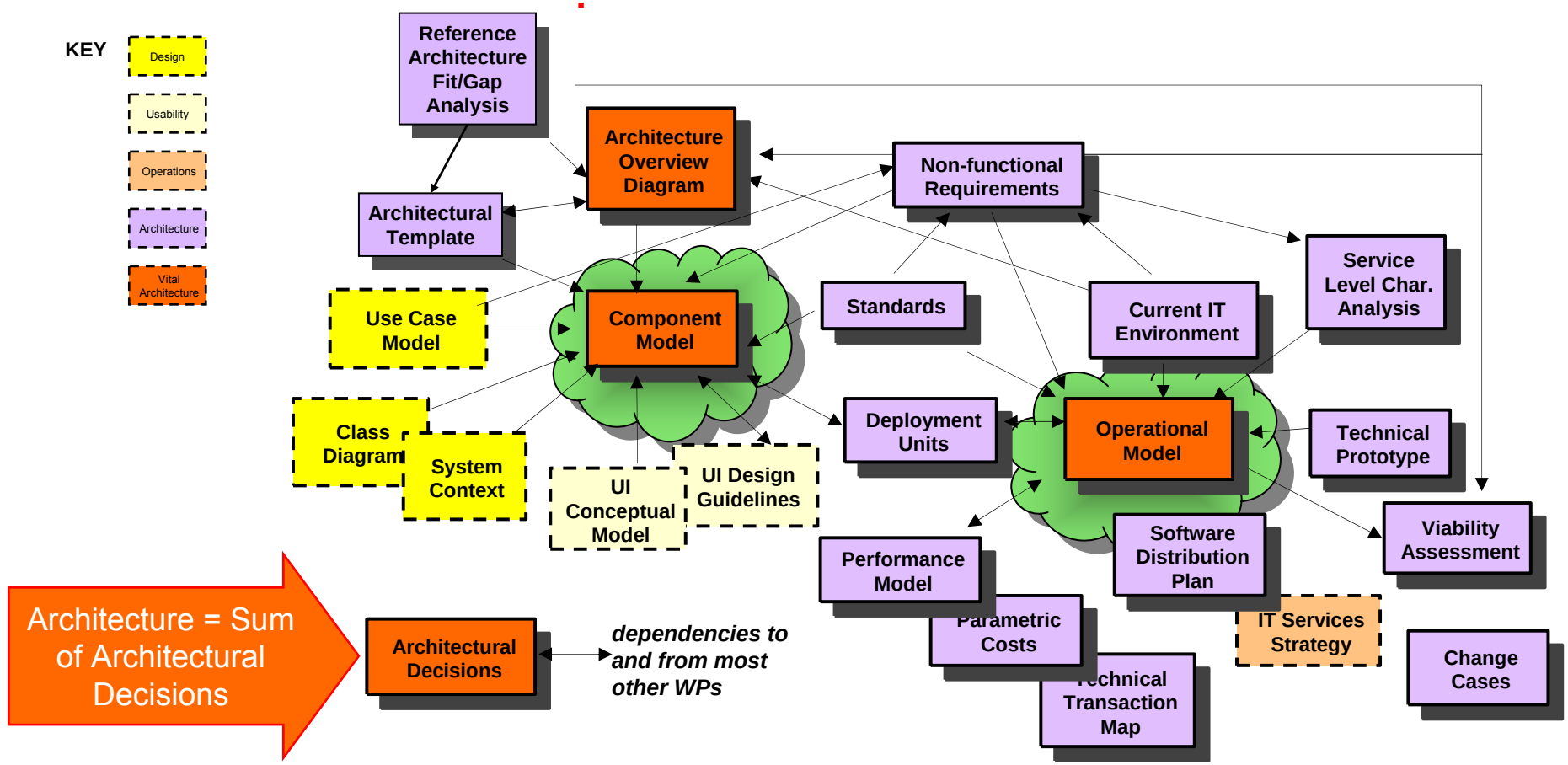
- No person who is not a great sculptor or painter can be an architect. If he is not a sculptor or painter, he can only be a builder.
  - John Ruskin, Lectures on Architecture and Painting (1853)
- Architect is the principal IT leader
  - In large organizations
  - In large and small projects
- Visioning and modeling
  - Bridging the Business-IT gap
  - Vocabulary, Language → Architecture Description Standard
  - **Method to support Architectural Thinking**
- Overseeing construction, to ensure actual implementation meets design
- Responsibility for acceptance of built system
- Multidisciplinary Synthesis
  - Technical, programmatic, managerial
  - Artistic, Heuristic

Method to support Architectural Thinking

Work Product Descriptions and Templates

Technique Papers, WBS Guidance for various project types

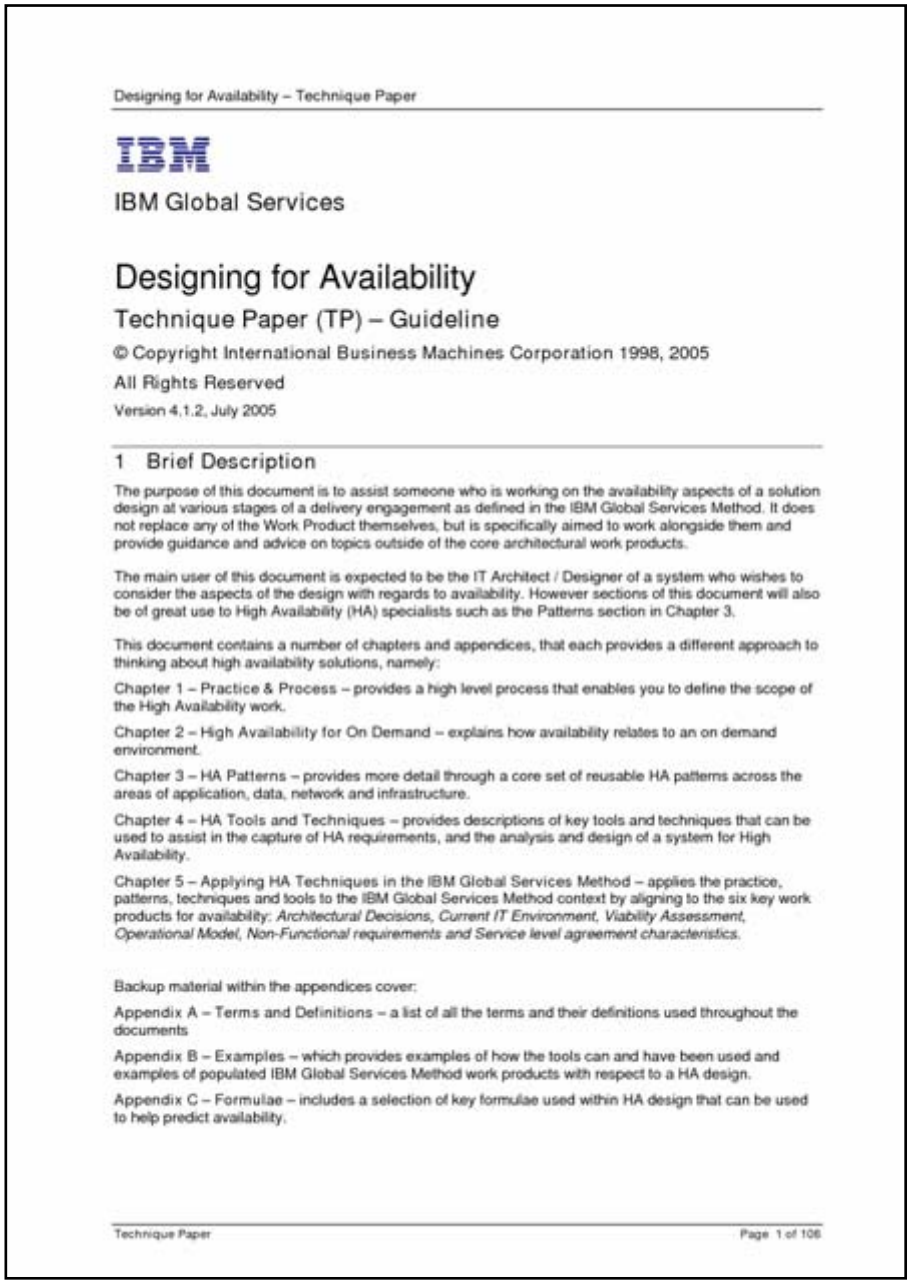
?



Method = codified experience / codified heuristics

Technique papers

- Written by practitioners for practitioners
- **Method organizes your work — Technique Papers actually help you to do it**
- Balance of theory and practical advice



Designing for Availability – Technique Paper

IBM

IBM Global Services

Designing for Availability

Technique Paper (TP) – Guideline

Version 4.1.2, July 2005

1 Brief Description

The purpose of this document is to assist someone who is working on the availability aspects of a solution design at various stages of a delivery engagement as defined in the IBM Global Services Method. It does not replace any of the Work Product themselves, but is specifically aimed to work alongside them and provide guidance and advice on topics outside of the core architectural work products.

The main user of this document is expected to be the IT Architect / Designer of a system who wishes to consider the aspects of the design with regards to availability. However sections of this document will also be of great use to High Availability (HA) specialists such as the Patterns section in Chapter 3.

This document contains a number of chapters and appendices, that each provides a different approach to thinking about high availability solutions, namely:

Chapter 1 – Practice & Process – provides a high level process that enables you to define the scope of the High Availability work.

Chapter 2 – High Availability for On Demand – explains how availability relates to an on demand environment.

Chapter 3 – HA Patterns – provides more detail through a core set of reusable HA patterns across the areas of application, data, network and infrastructure.

Chapter 4 – HA Tools and Techniques – provides descriptions of key tools and techniques that can be used to assist in the capture of HA requirements, and the analysis and design of a system for High Availability.

Chapter 5 – Applying HA Techniques in the IBM Global Services Method – applies the practice, patterns, techniques and tools to the IBM Global Services Method context by aligning to the six key work products for availability: *Architectural Decisions, Current IT Environment, Viability Assessment, Operational Model, Non-Functional requirements and Service level agreement characteristics.*

Backup material within the appendices cover:

Appendix A – Terms and Definitions – a list of all the terms and their definitions used throughout the documents

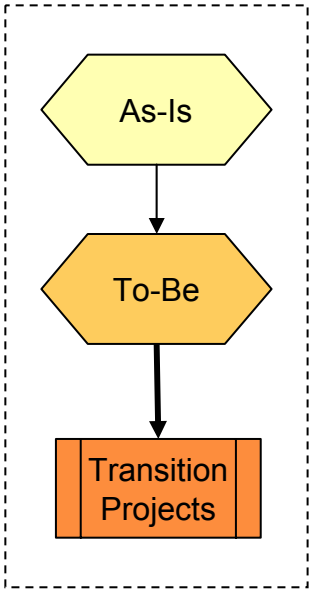
Appendix B – Examples – which provides examples of how the tools can and have been used and examples of populated IBM Global Services Method work products with respect to a HA design.

Appendix C – Formulae – includes a selection of key formulae used within HA design that can be used to help predict availability.

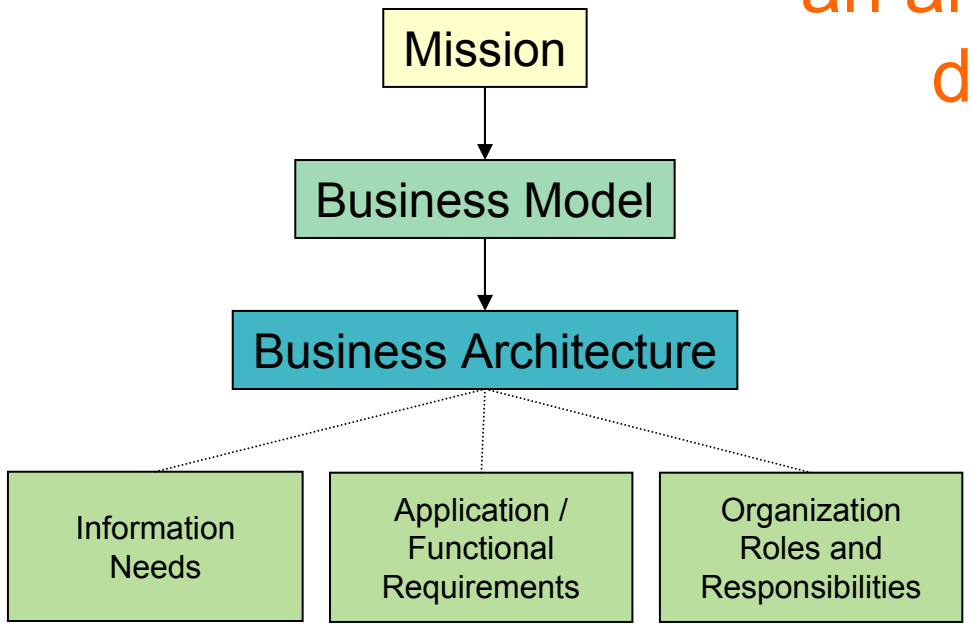
Technique Paper Page 1 of 106

Method → Process of architecture

What does an architect do?



Documentation



L0 Architecture Overview

Information **Systems** Architecture

L1 Standards Guidelines

Information **Technology** Architecture

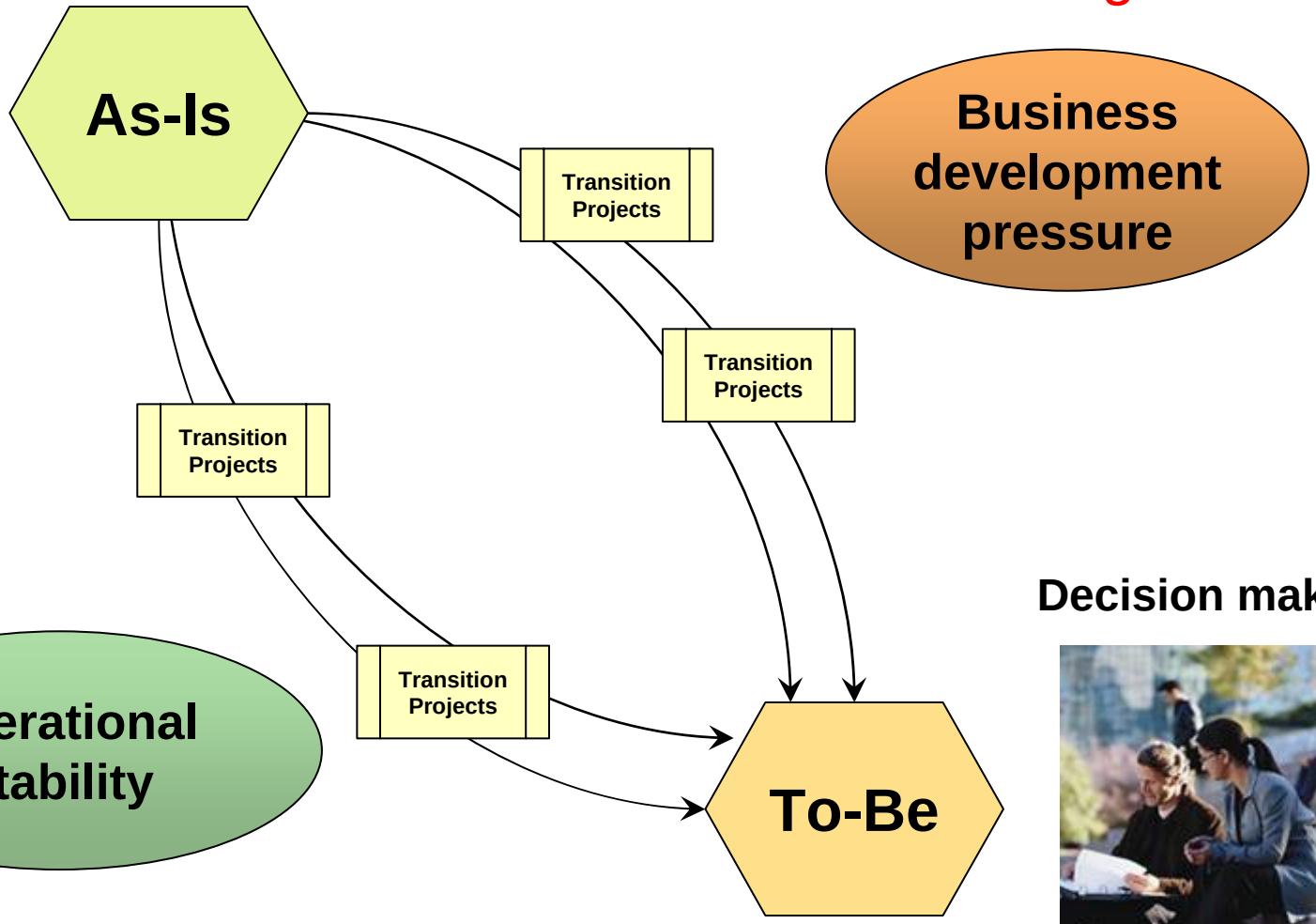
L2 Detailed designs and projects

IT architecture proved to be successful in providing an **analytical and decision-making framework** for a sequence of new initiatives or **changes**, at the same time ensuring **design integrity** and stability

http://www.research.ibm.com/journal/

Analysis

Change



Decision making

Design integrity

Once again: As the author of the solution, the architect is undeniably accountable for the effort's success or failure.

Reviews cannot take that responsibility away from the architect, but reviews can help to identify risks that the architect may have overlooked, things that may be missing, or bad architectural decisions.

- Topics that the Lead Architect typically covers in a presentation at the beginning of a review
- Qualities that are typically reviewed
- Areas of risk that are typically investigated in a review

Topics that the lead architect typically covers in a presentation at the beginning of the review (1 of 3)

- **Driving architectural requirements**
  - Measurable quantities associated with these requirements
  - Any existing standards / models associated with these requirements
  - Any existing standards / models / approaches for meeting these requirements
- **High-level architectural views**
  - Other systems with which the system must interact
  - Key events which the architecture must enable
  - System overview
  - Inter-dependencies of the sub-systems
  - Technical constraints such as an operating system, hardware, or middleware prescribed for use
  - Architectural approaches used to meet quality attribute requirements
  - Key workproducts, covering requirements, performance, availability, capacity, security, manageability, development, testing, and implementation
- **Architectural issues / risks** with respect to meeting the driving architectural requirements
- **Functional view**: Functions, Business Rules, Data Flows

Topics that the lead architect typically covers in a presentation at the beginning of the review (2 of 3)

- **Service Model**
  - Service identification: Online requests, non-interactive requests, Batch “Mass Requests”
  - Relationship between business processes and services
  - Service specification: Service Versioning, Message Versioning and Schema Validation
  - Service catalog
  - Service hierarchy
  - End-to-end view: showing how information flows from system border (service interface) to the backend and vice versa – bridging all system, layer, protocol, technology, etc. borders
- **Code:** describe the system’s decomposition of functionality
  - Subsystems
  - Layers
  - Components
  - Modules (Objects, procedures, and functions that populate the modules)
  - Relationships among modules (procedure call, method invocation, callback, containment)
  - Concurrency (Processes, thresds, synchronizatoin, dataflow, events that connect processes and threads)
- **Physical:** CPU’s, Storage, Networks and communication devices that connect them
- **Architectural approaches**, styles, patterns, or mechanisms employed
  - What quality attributes are addressed by these approaches, styles and patterns
  - Description of how the approaches address those attributes

Topics that the lead architect typically covers in a presentation at the beginning of the review (3 of 3)

- **Trace** of 1 – 3 of the most important use case scenarios
  - including the run-time resources consumed for each scenario
- **Trace** of 1 – 3 of the most important growth or other change scenarios, describing the change impact
  - estimated size / difficulty of the change in terms of the changed components, connectors, services, and interfaces
- **Process description**
  - Problem management
  - Change management
  - Operational change control
  - Defect management
  - Release management
  - Test and integration
  - Deployment
  - Acceptance
  - Document management
- **Assessment** of the lead architect: challenges, issues, and areas of concern
  - Technical risk register and technical roadmap
  - Is the system architecture documentation fit for the purpose in terms of completeness, level of detail, currency and structure?

Qualities that are typically reviewed (1 of 2)

- **Functionality**: Ability of the system to do the work for which it is intended
  - Requires that the system's components work in a coordinated manner to complete the job
- **Performance**: Responsiveness of the system
  - Time required to respond to stimuli (events)
  - Number of events processed in some interval of time
- **Reliability**: Ability of the system to keep operating over time
  - Measured by mean time to failure
- **Availability**: Proportion of time the system is up and running
  - Measured by the length of time between failures, and how quickly the system is able to resume operation in the event of failure
- **Security**: Ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users
  - Categorized in terms of the types of threat that might be made to the system

Qualities that are typically reviewed (2 of 2)

- **Modifiability**: Ability to make changes to a system quickly and cost effectively
  - Ability to make isolated changes
  - Measured by using specific changes as benchmarks and recording how expensive those changes are to make
- **Conceptual integrity**: underlying theme or vision that unifies the design of the system at all levels
  - The architecture should do similar things in similar ways
  - The architecture should exhibit consistency
  - The architecture should have a small number of data and control mechanisms
  - The architecture should use a small number of patterns to get the job done
- **Operation of technical management** across the system life-cycle
  - Requirements traceability (functional and nonfunctional)
  - Architectural governance
  - Implementation governance

Areas of risk that are typically investigated in a review (1 of 3)

If the answer to any of the questions is „no“ then this indicates an area of risk which needs to be mitigated

- Requirements
  - Have likely changes been identified and recorded?
- Architecture
  - Is it possible to map, or decompose, requirements to individual elements in the architecture → Functional requirements, and Quality attributes: Modifiability, Performance, Security, Availability, Reliability
  - Can the architecture accommodate the identified changes?
- Detailed design and implementation
  - Are notations and programming systems used which facilitate verification that the software meets its requirements?

Areas of risk that are typically investigated in a review (2 of 3)

If the answer to any of the questions is „no“ then this indicates an area of risk which needs to be mitigated

- Verification
  - Are all specifications, items of software, test scripts, etc. reviewed by people other than their authors?
  - Have the designs and items of software been analysed by tools that can detect errors and inconsistencies?
  - Are individual software components tested by people other than their authors?
  - Is there a systematic way of ensuring that all the software in individual components is thoroughly tested?
  - Is there a way of systematically testing the software as it is progressively integrated to produce the complete system?
  - Is there a means of regression testing following changes (to ensure that changes have not caused unintended changes in system behavior)?
- Traceability
  - Is all software traceable to a top-level requirement?
  - Are all requirements traceable to the software which implements them?

Areas of risk that are typically investigated in a review (3 of 3)

If the answer to any of the questions is „no“ then this indicates an area of risk which needs to be mitigated

- Configuration management
  - Is it possible to identify every software item in a particular build, or release, of the software system?
  - Is it possible to re-build any previous release of a software system?
  - Is it possible to know to which item of software a user problem report relates?
- Change management
  - Is it possible to determine the impact on the software and all associated information (requirements, architecture, tests) of any proposed change before implementing it?
  - Are changes grouped to minimise their impact and to ease their verification?
- Things missing (NFR)
  - Plausibility checks?
  - Dealing with unexpected input, unexpected return codes from the infrastructure, etc.?
  - Application monitoring instrumentation?
  - Queue monitoring?
  - etc.

Exercises on Heuristics

- Choose a system, software product, or software development process with which you are familiar and assess it using heuristics
  - What was the result?
  - Which heuristics are or were particularly applicable?
  - What further heuristics were suggested by the system chosen?
  - Were any of the heuristics clearly incorrect for this system?
  - If so, why?
- Try to spot heuristics and insights in the technical literature.
  - Some are easy; they are often listed as principles or rules.
  - The more difficult ones are buried in the text but contain the essence of the article or state something of far broader application
- Try to create a heuristic of your own — a guide to action, decision making, or to instruction of others.