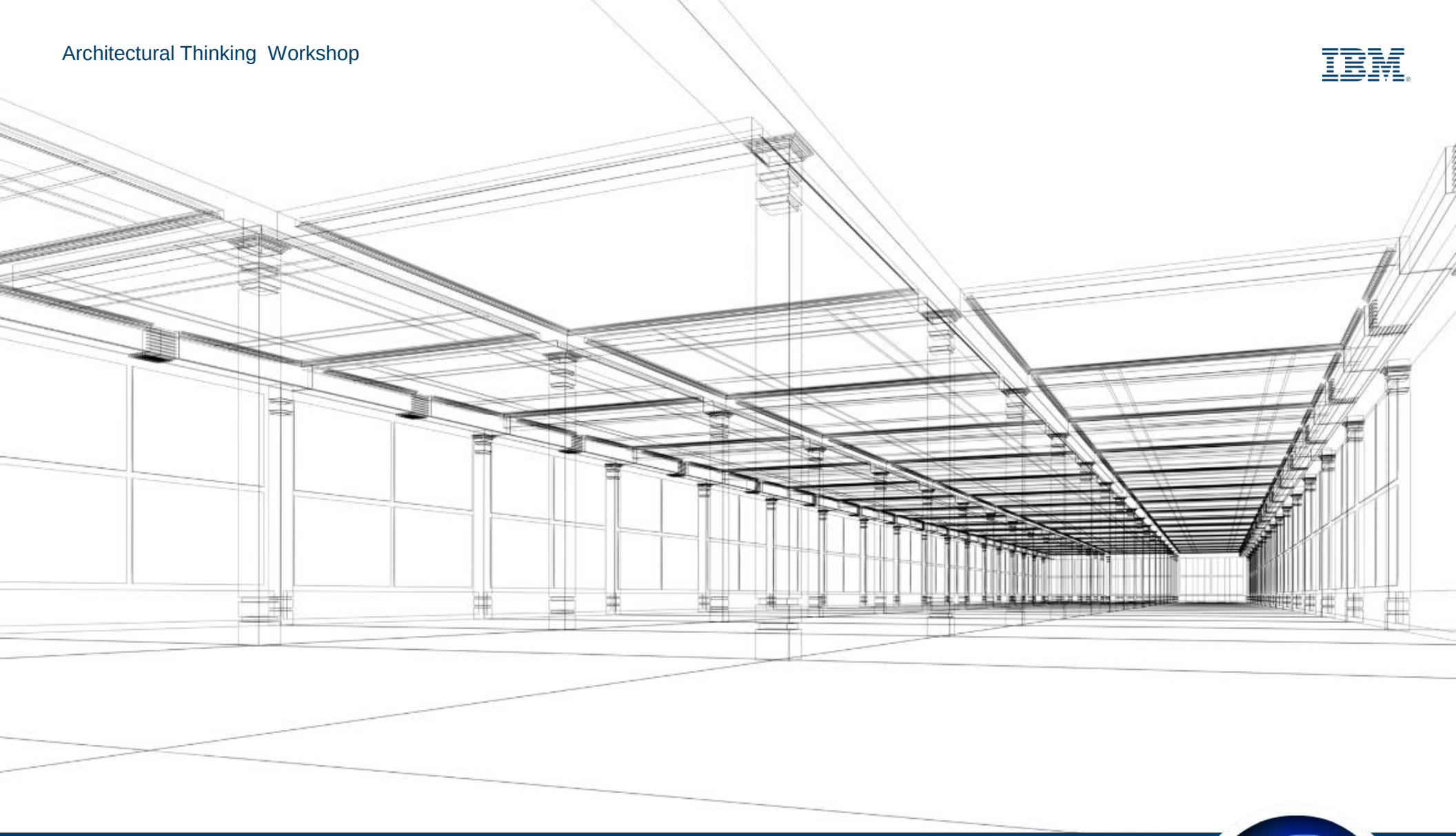


Requirements Aspect

Module outline

- **Introducing Requirements**
  - **What are Requirements?**
  - **Sourcing Requirements**
- System Context Diagram
- Functional Requirements
- Non-functional Requirements
- Constraints
- Future Requirements
- Managing Requirements
- Summary and References

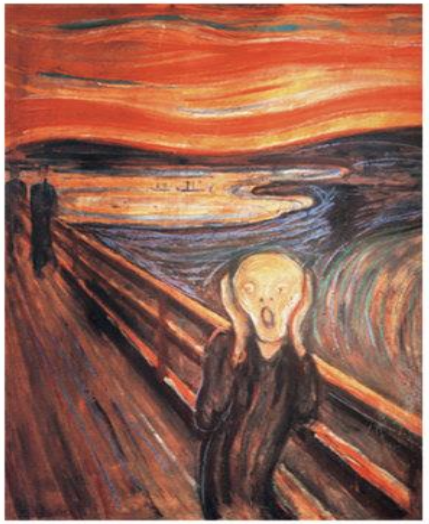
Learning Objectives

At the end of this module, you should be able to:

- Describe why requirements are important for Architects and the architecture that they develop
- Understand how requirements are classified
- Understand how to elicit requirements
- Explain which requirements are important in defining the architecture
- Show how an existing enterprise architecture, project context, and business case contribute to an understanding of the requirements
- Demonstrate how requirements can be documented
- Understand how requirements should be prioritized
- Understand the importance of a Requirements Management Plan and the associated Requirements Traceability Matrix to the successful implementation of the project

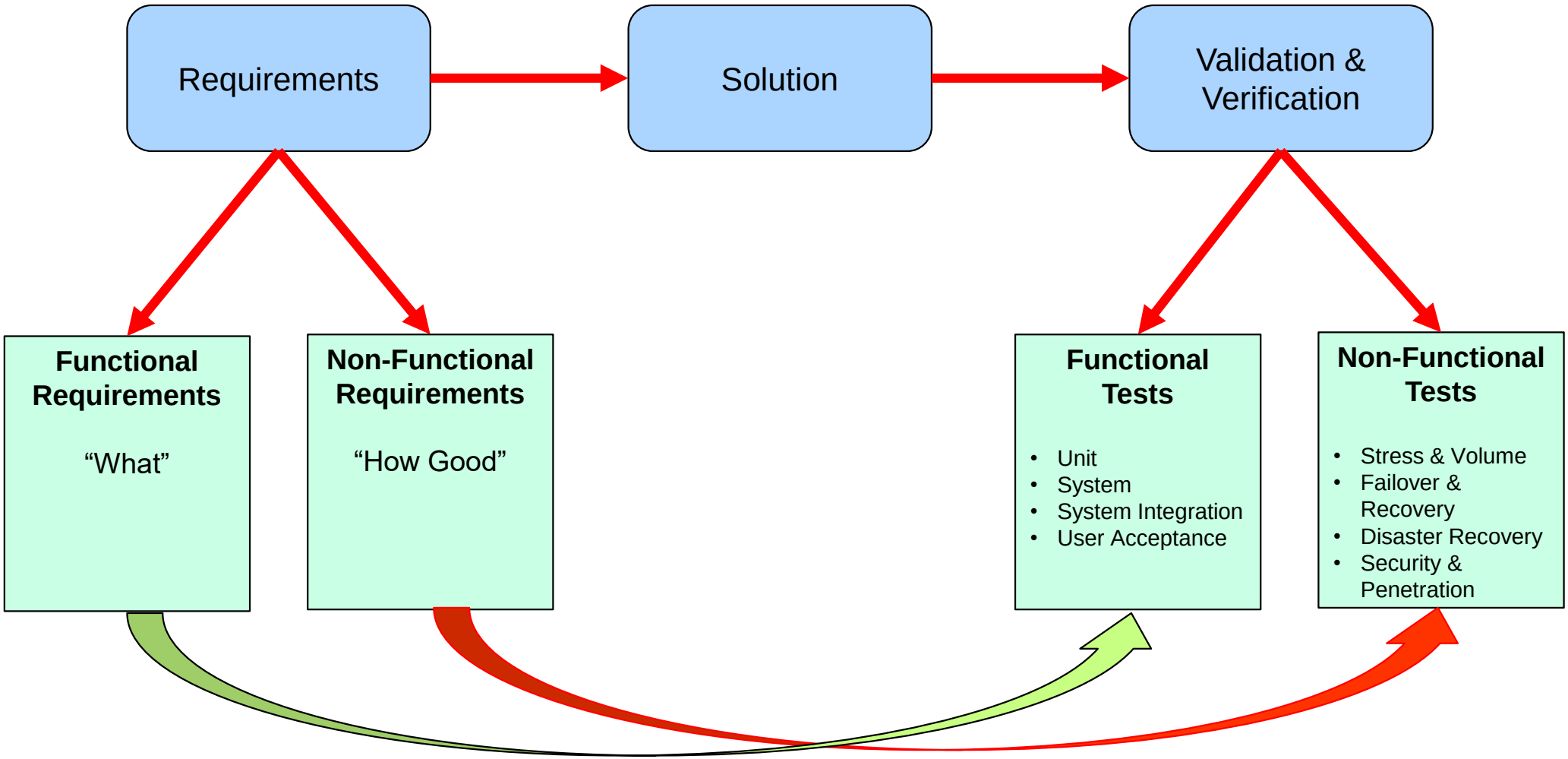
The importance of requirements management

- According to the Standish Group, only 39% of all projects are successful.[1]
- 37% of all organizations reported inaccurate requirements as the primary reason for project failure.[2]
- Rework accounting for approximately 30-40% of all software development project budgets, and roughly half of the rework on a typical project, can be attributed to missing or misunderstood requirements.[3]
- For every million dollars spent on software development, $150,000 to $200,000 is wasted because of bad requirements.[4]
- Requirements errors cost US companies $30 billion a year in total.[4]



1. *The Standish Group: "Chaos Manifesto 2013"*
2. PMI's Pulse of the Profession® (2014)
3. Leffingwell, Dean, *Managing Software Requirements*, Addison-Wesley, 2003
4. National Institute of Standards & Technology, US Department of Commerce, *The Economic Impacts of Inadequate Infrastructure for Software Testing,* 2002

Architecture Process: The three phases

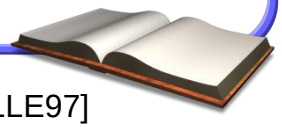


The four basic types of requirements (1 of 2)

- Functional Requirements (FRs)
- Non-Functional Requirements (NFRs)
- Constraints
- Future Requirements

Definition: *Requirements are a specification of what should be implemented.*

*They are descriptions of how the system should behave, or of a system property or attribute. They may be a constraint on the development process of the system.**



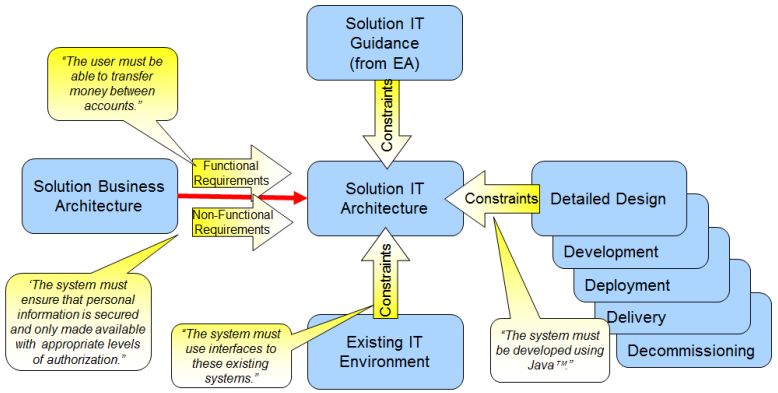
*See [SOMMERVILLE97]

The four basic types of requirements (2 of 2)

- Functional requirements:
  - Are capabilities needed by users of the IT solution to fulfill their job
  - Answers the question of "what" the solution's sponsor needs (but often not "how" it is achieved)
  - Tend to be "descriptive" or "qualitative"

- Non-Functional Requirements:
  - Define the expectations and characteristics that the IT system should support
  - Might be runtime (for example, performance or availability) or non-runtime (for example, scalability or maintainability)
  - Tend to be "prescriptive" or "quantitative"

- Constraints:
  - Given those things that cannot be changed within the scope and lifetime of the project (for example existing infrastructure)
  - Other factors, such as available skills, budget, and operational limitations
  - Architectural guidelines and standard technologies (as may be declared in the enterprise architecture)



- Future Requirements
  - Describe how the system might change in the future, whether these are functional or nonfunctional enhancements.

A solution architecture will have many stakeholders, who are potential sources of a requirement specification (1 of 2)

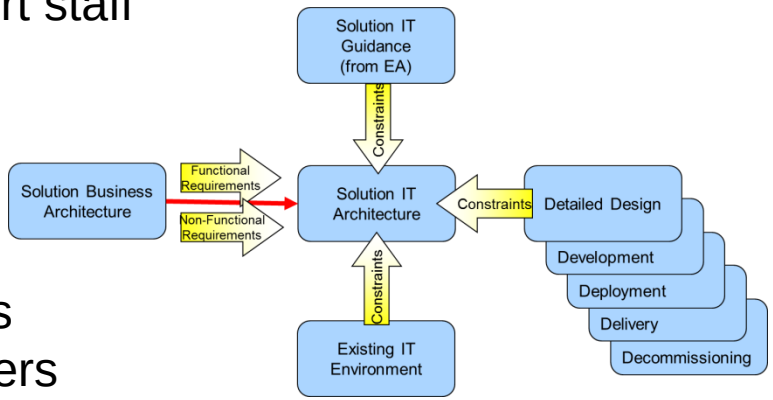
Definition: A stakeholder in a solution architecture is someone with interest in or concerns about the realization of the architecture.*

*After [ROZANSKI05]

- **Users**
  - *Functionality*
  - *(Long term) Price*
  - *Time to market*
  - *Performance*
  - *Security*

- Assessors
- Sponsors
- Support staff

- Communicators
- Business Architects
- Enterprise Architects



- Acquirers
- Maintainers
- Operations

- Systems administrators

- Suppliers
- Developers
- Testers

- Wherever they come from, they form the basis of a **requirements specification.**
  - *Probably in need of analysis, negotiation, and acceptance by the solution Architect*

The wider project environment is a rich source of requirements and constraints (2 of 2)

External Environment

● Laws and regulations ● Legal liabilities ● Social responsibilities ● Technology base ● Labor pool ● Competing products ● Standards and specifications ● Public culture

Enterprise Environment

● Policies and procedures ● Parent corporation issues ● Legacy systems ● Enterprise architecture ● Standards and specifications ● Local culture ● Guidelines ● Domain technology ● Other initiatives

Project Environment

● Directives and procedures ● Plans ● Tools ● Metrics

- Project management
- Technical management
- Integration, test, and deployment
- Colocated vs. virtual development team(s)
- Infrastructure
- Operations and maintenance

Project A
Project B
Project C

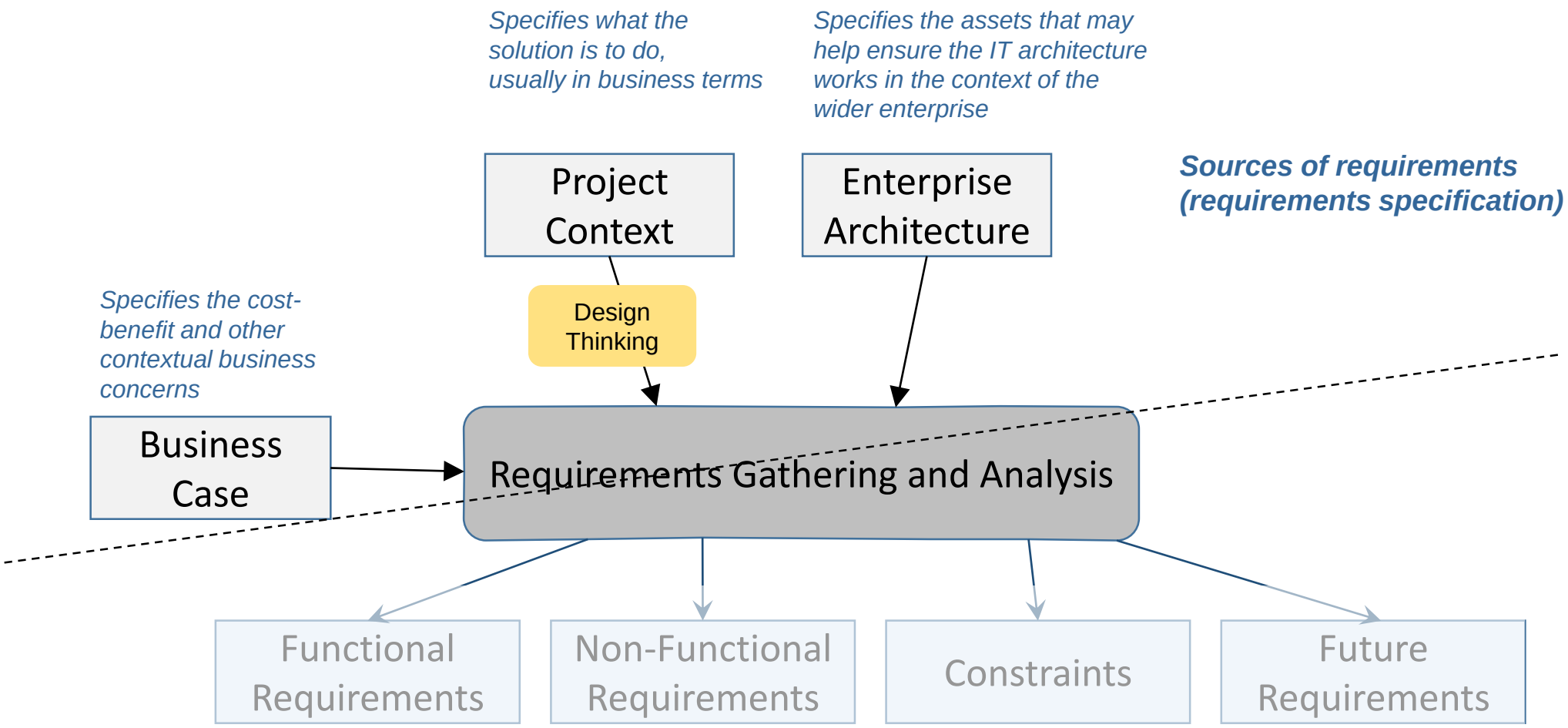
Interdependent Parallel Programs

Enterprise Support

- Resource management
- Project reporting
- Brand support
- Training
- Infrastructure

Modified from INCOSE Handbook Ver. 2

First, we need to focus on where our requirements are coming from



Let's look first at sources of requirements

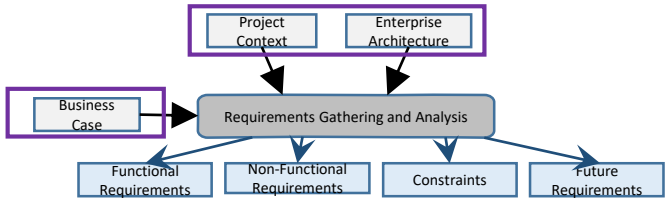
Where our requirements are coming from

An enterprise architecture defines a set of governing principles, patterns, standard building blocks (ABBs), and reference architectures, together with architecture compliance processes.

- For IT, the enterprise architecture provides:
  - The long-term strategic IT objectives that an IT solution should support
  - The principles and standards with which the solution must comply
  - Appropriate reference architectures to be followed
  - Governance structure for review and approval

Business case influence architectural decisions.

- Investment drivers will affect choice of technology.
- Key required features such as a “single customer view” to provide better customer service will affect data placement, legacy access, and so on.
- Ongoing support costs, such as a single point of contact for support, have an impact on system management capabilities and so on.



Project context is the set of objectives and assumptions used to frame the solution that IBM is developing.

Project context is important to understand:

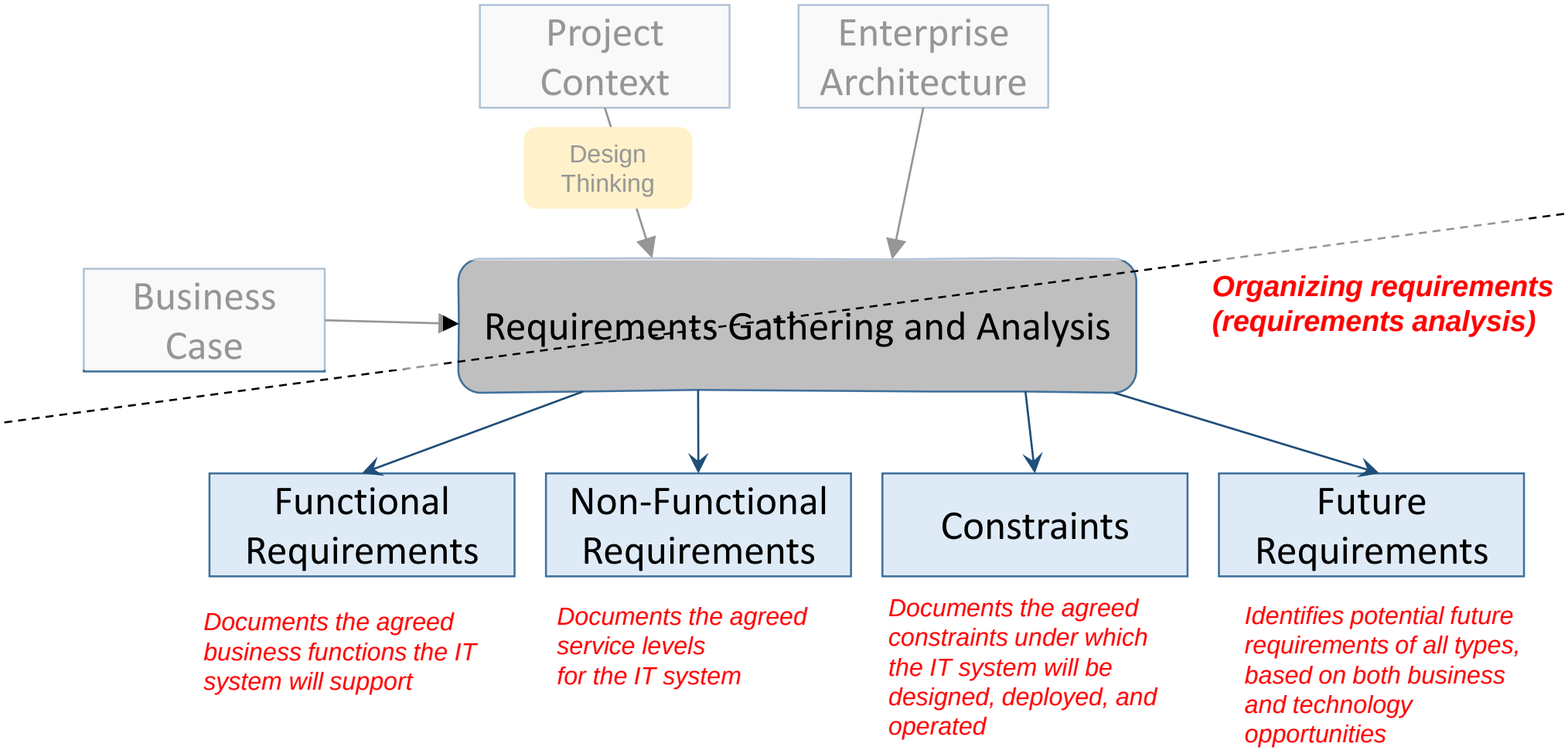
- What are the business objectives the project is supposed to meet?
- What are the baseline technology assumptions?
- What are the relationships (for example, dependencies) with other ongoing or planned projects?

Assumptions implied or stated in the proposal or Statement of Work (SOW):

- In-scope application functionality
- Service level agreements
- Customer infrastructure assumptions
- Product or vendor assumptions
- Development environment assumptions

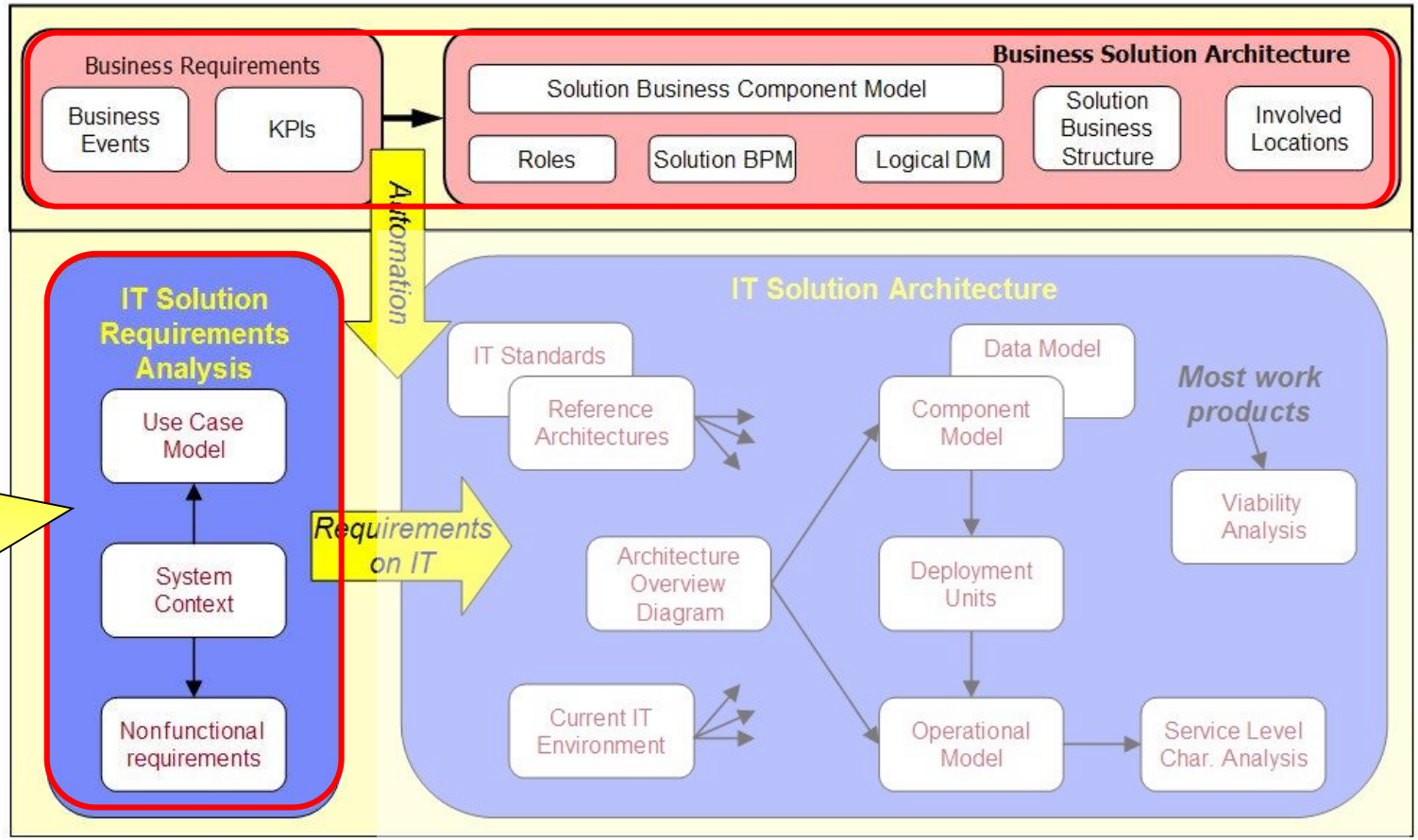
Second, how do we organize and analyze all this requirements information?

Now, let's look at requirements analysis!



Work product dependency diagram provides a useful requirements (and solution) framework

"Some of our requirements are documented directly in business terms."



"Some of our requirements are an analysis of the business requirements, documenting the interface between the IT system and its users."

However, there's a much larger requirements artefact landscape than we can address here

The work products give us an overall sense of the IT system's boundary…

- **System Context**
  - Functional and operational aspects
- **Use Case Model**
  - Including change cases
- **Nonfunctional Requirements**
  - Including Constraints

- …and more specific requirements in focus areas
  - Usability Requirements
  - Network Requirements
  - IT Management Requirements
  - Security and Privacy Requirements
  - Future Organization Scope and Requirements

- Others document what the overall system is to do…
  - **User Story**
  - **Prototype**
  - Business Event List
  - Customer Wants and Needs
  - Process Definition
  - Subject Area Model
  - Stakeholder Requests

- …and help us understand the project's overall context
  - **Business Direction**
  - **Project Definition**
  - Business Case
  - Decision Model
  - **Technical Environment**
  - **Standards**
  - **Current Organization Description**

- As the project moves forward, additional work products help us track why the solution looks the way it does:
  - **Requirements Matrix**
  - Requirements Traceability Matrix

Requirements sources (1 of 3)

Time for a question

What is the source of the following requirement? *"This project depends on project X going live."*

A. Enterprise architecture

B. Business case

C. Project context

Requirements sources (2 of 3)

Time for a question

What is the source of the following requirement? *"The proposed solution must exploit packages at all times."*

A. Enterprise architecture

B. Business case

C. Project context

Requirements sources (3 of 3)

Time for a question

What is the source of the following requirement? "*The proposed solution must pay for itself within 3 years.*"

A. Enterprise architecture

B. Business case

C. Project context

Whatever artefacts we use to analyse and organize our requirements, they need to be SMART

Individual requirements have to be*:

- **S**pecific
  - Unambiguous, consistent, and at the appropriate level of detail
- **M**easurable
  - Possible to verify that a requirement has been met, so include success criteria
- **A**ttainable
  - Technically feasible and within the art of the possible
- **R**ealizable
  - Realistic given all the constraints (e.g., resources, skills, time, infrastructure)
- **T**raceable
  - Linked from conception through specification, design, implementation, and test
- As a group, requirements have to be:
  - Non-conflicting
  - Prioritized

*See [Bredemeyer, Dana and Malan, Ruth. *Defining Non-Functional Requirements*, Bredemeyer Consulting, March 2001]

Seven general good requirements rules of thumb

1. A requirement must form a complete sentence.
2. Avoid using abbreviations unless they are defined.
3. Make consistent use of one of the following verbs:
   - *Shall*, *will*, or *must* to show mandatory requirements
   - *Should* or *might* to show optional requirements
   - *Could or would* to show desirable requirements
4. They must contain the success criteria and be measurable and testable.
5. A unique reference ID must be provided; for example, NFR-PERF-001.
6. Refer to supporting material; do not duplicate information.
7. Avoid:
   - Ambiguity (instead, write as clearly and explicitly as possible)
   - Rambling, long sentences with arcane language
   - Wishful thinking (such as “100% reliable,” “totally safe,” and so on)

MOSCOW:

Must (so will)
Should (and probably will)
Could (but might not)
Would (but won’t)

But, of course, this represents a perfect world…

A collection of requirements pitfalls

- The “this is too technical for me” attitude
  - Which really means “*I do not know how to describe what I want*” or “*I’m embarrassed that I do not know what I want.*”
- The “all requirements are equal” fallacy
  - Which is a bit like saying “*all or nothing!*”
- The “park it in the lot” problem
  - Which probably means “*it’s not my idea; I’d rather have something else, please.*”
- The “requirements that can't be measured” syndrome
  - Such as “*make it better than before*” or “*I’ll know what I want when I haven’t got it.*”
- The “not enough time” complaint
  - A multimillion-dollar project probably has complicated requirements, and complicated requirements take time to work out!
- The “lack of ownership” problem
  - Who specifies, and who analyzes? The person paying or the person supplying?
- The “all stakeholders are alike” misconception
  - Is a functional requirement more or less important than a constraint?
- The “too general” tendency
  - Architectural requirements need to be more abstract than those for software development, but not so abstract that they are meaningless.

Requirements should be SMART (1 of 3)

Time for a question

1. Is the following an example of a SMART requirement?

"The proposed solution must be better than the current system."

Requirements should be SMART (2 of 3)

Time for a question

2. Is the following an example of a SMART requirement?

The proposed solution must decrease customer callbacks by 10%, based on recorded call center statistics.

Requirements should be SMART (3 of 3)

Time for a question

3. Is the following an example of a SMART requirement?

The proposed solution must completely eliminate the need for customer support.

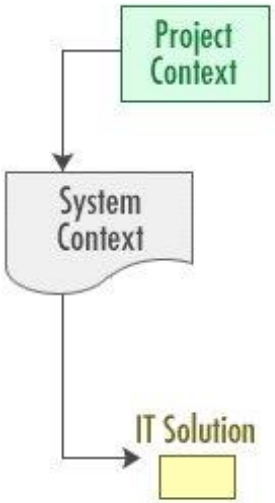
Module outline

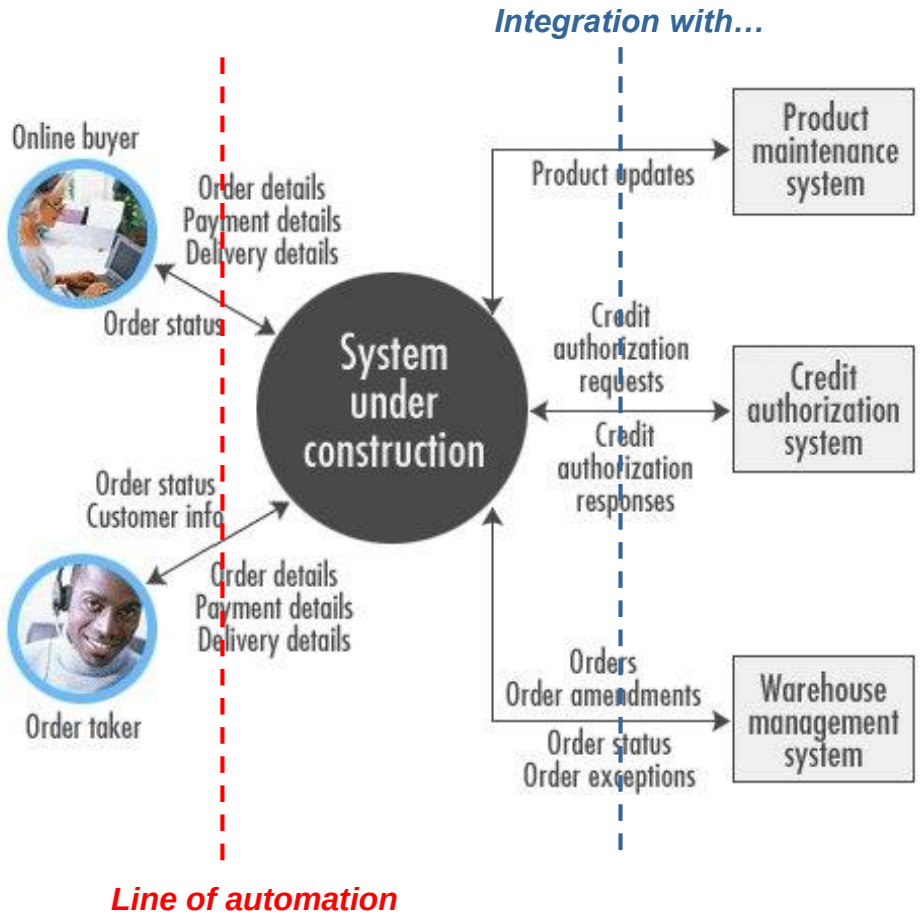
- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- **System Context Diagram**
- Functional Requirements
- Non-functional Requirements
- Constraints
- Future Requirements
- Managing Requirements
- Summary and References

System's boundary - captured in the system context artifact

The system context:

- Clarifies the environment in which the system has to operate
- Puts bounds on the system
- Identifies external interfaces (with human users and other IT systems)

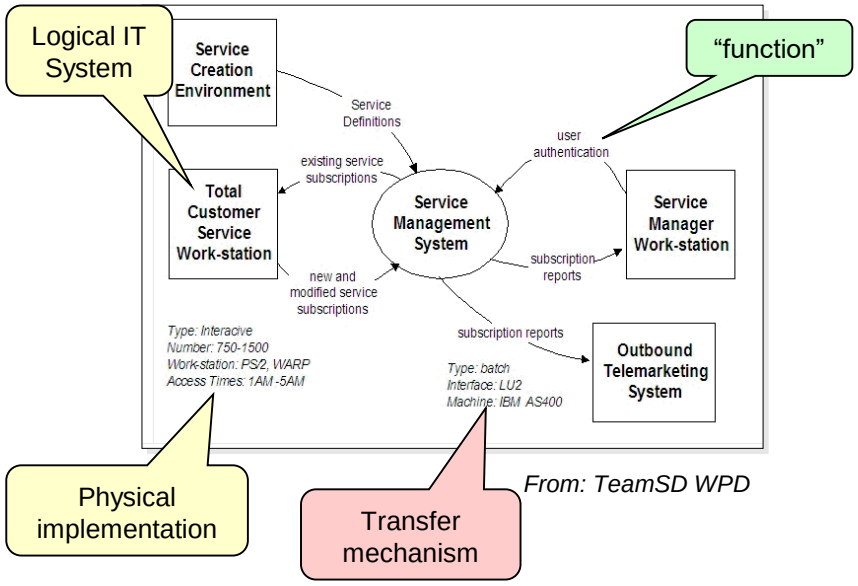




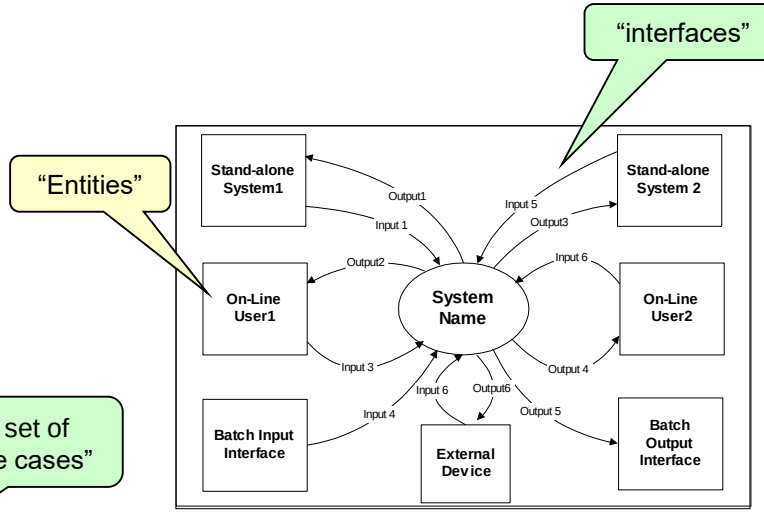
The system context

The system context can be produced as a “free form” sketch in which the meaning of the relationships between the system and its external agents is ambiguous.

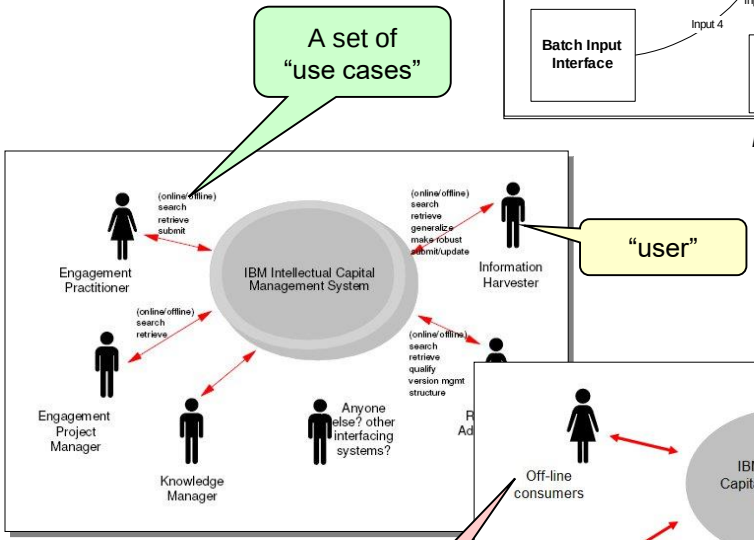
- Who, or which, is interacting with the target system?
- What are they doing?
- How are they doing it?



From: TeamSD WPD



From: SCD Artifact template



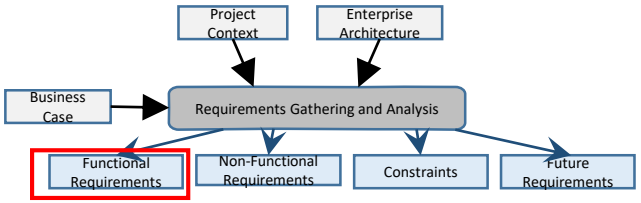
From: AT Class Case Stu… in…

From: AT Class Case Study sample answer

Module outline

- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- System Context Diagram
- **Functional Requirements**
- Non-functional Requirements
- Constraints
- Future Requirements
- Managing Requirements
- Summary and References

Functional requirements



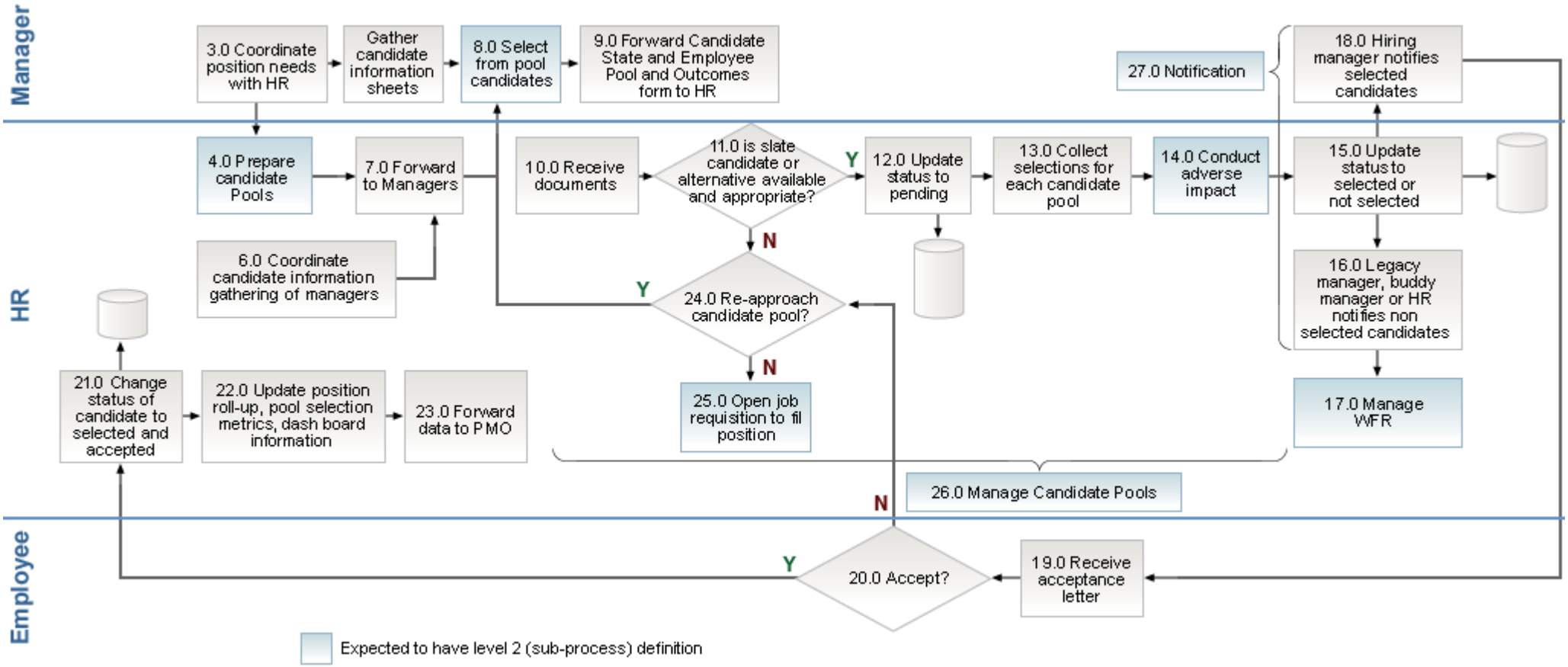
- Functional requirements describe what the system will do from a number of perspectives – each with its own architectural implications.

- Business functionality - *What the system has to do "inside"*
  - Customer Wants & Needs
  - Business Process Model
  - Use Case Model
  - Use Case Specification
  - Business Rules
  - Epics and User Stories

- Users – *What the system's users can do, and where they are*
  - Design Thinking
  - Business Roles (Personas) and Locations
  - User Groups

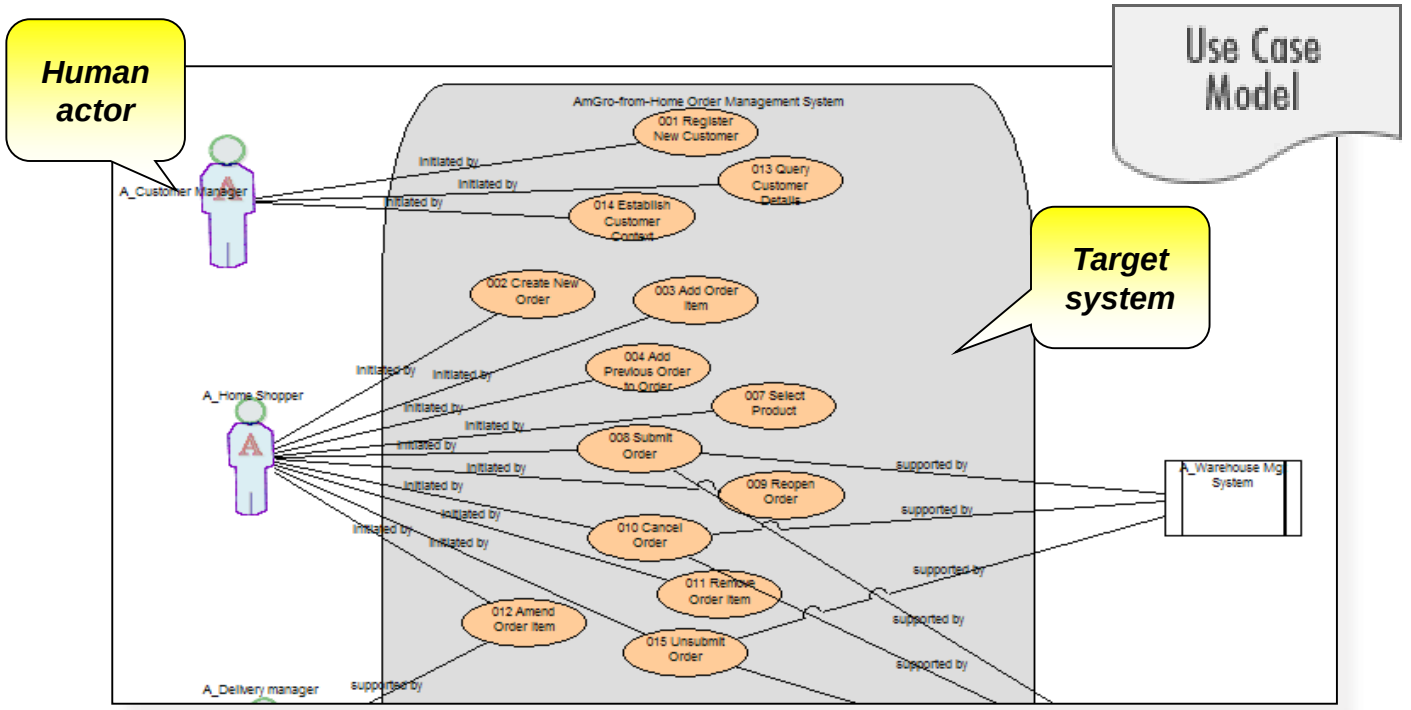
Business processes are a way of documenting functional requirements without concern for automation

Business process modeling, system context, and use case modeling help to bridge between business architecture and IT architecture.



Use Case Model

- The Use Case Model is a way of grouping all the use cases in scope for a particular solution (or solution domain for large solutions) and presenting them as a single model (or a reasonably small number of models)
- **A Use Case Model can be seen as a summary of the Use Cases in scope**



A use case consists of a main scenario together with one or more alternatives

Sample use case

- Main scenario
  - The system requests the person's details.
  - The user supplies the address and phone number.
  - The system validates the address and phone number.
  - The use case ends successfully.

- Alternatives

  3a. Non-unique name or town supplied at 2
  - 3a1. The system supplies a list of matching full names and/or post codes.
  - 3a2. The user selects an entry from the list.
  - 3a3. The system supplies the address and phone number.
  - 3a4. The use case ends successfully.

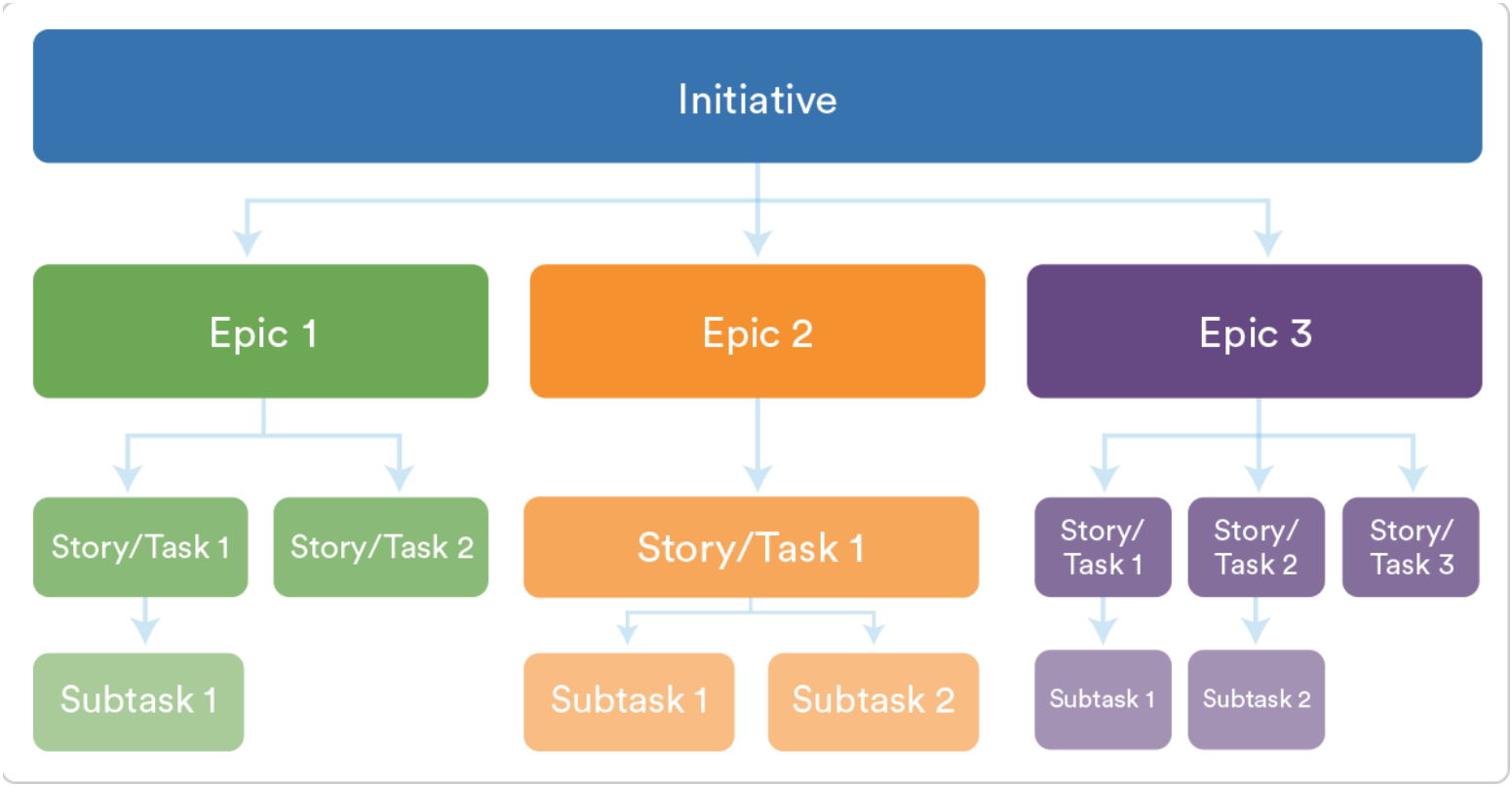
Good use cases often adopt this "conversational style":

I say, you say…

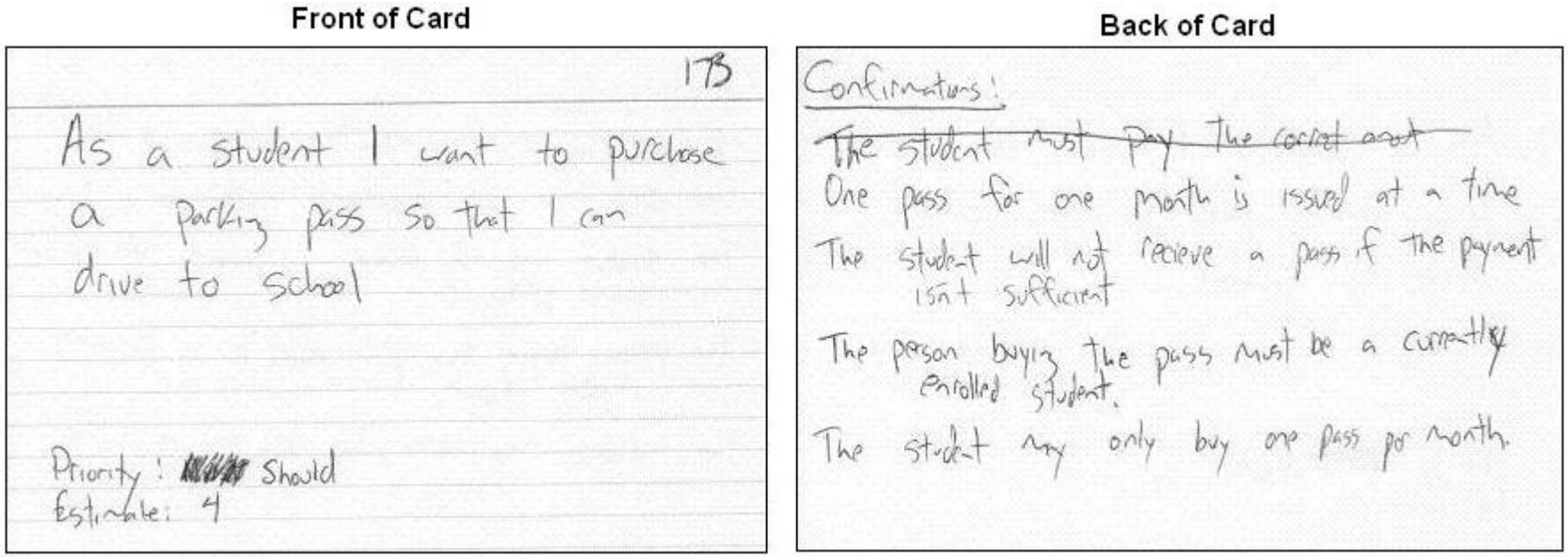
Do not confuse each step (what each party does) with the message that flows from party to party between the steps.

Functional requirements in agile world

- Functional requirements under various agile methods are likely to be defined in terms for Epics, User Stories and Themes
  - Epics are single, large stories that usually represent complex requirements that need to be broken down before development.
  - A user story is a brief description of a system requirement from the user's or stakeholder's perspective.
  - Themes are collections of user stories that are grouped together for prioritization and planning purposes.



User Stories – Formal User Story Card with Confirmations



Use cases vs user stories

- No agreement if and when use cases and user stories are both applicable or when one versus the other should be used.
- However, one can successfully argue that:
  - User Stories are a way to describe requirements.
  - Use Cases can be used as a way to detail the User Stories.

Design Thinking – Framework to solve user problems

Empathy Map

As- Is Scenario Map

Explore Ideas

Stakeholder Playback

- From the IBM Design Thinking Field Guide –

  IBM Design Thinking is our approach to applying design thinking at the speed and scale the modern enterprise demands. It's a framework for teaming and action. It helps our teams not only form intent, but deliver outcomes – outcomes that advance the state of the art and improve the lives of the people they serve.

- The framework has us:
  - Work with **sponsor users** to understand and empathize with those who will be most impacted by whatever we are designing
  - Ideate to generate big ideas and then prioritize ideas that we have agreement will be impactful while within the bounds of feasibility
  - Establish a common statement (intent) that helps to align the entire team [our **Hills**]
  - And along the way continue to involve all sponsor users and stakeholders in the process through **Playbacks** to insure meaningful impact of the outcome of the design

The Architecture Process: From client need to client value

Checkpoint: (1) How does a solution look like from the Client Customer's perspective? (2) What are the main ideas underpinning the approach to solving the business problem?

Checkpoint: (1) What is the solution to the business problem? (2) Is it feasible? (3) What are the risks? (4) How much will it cost? (5) How long will it take to deliver?

Project Planning → Analysis → Design → Impleme ntation → Operation

IBM Design Thinking

Empathy Map | Needs Statement | Storyboard / Prototype

As Is Scenario | Ideation / Prioritization | Hills / Playback 0

Delivery Playbacks (Agile Methodology)

Key Questions: What are the needs of the Client or Client's Customers? How can the Client's business astonish its customers and address their core needs in a new and innovative way?

Key activities: Understand users, their as-is experience, & pain points. Explore creative to-be opportunities. Prototype to-be concepts to visualize them. Evaluate prototypes with users.

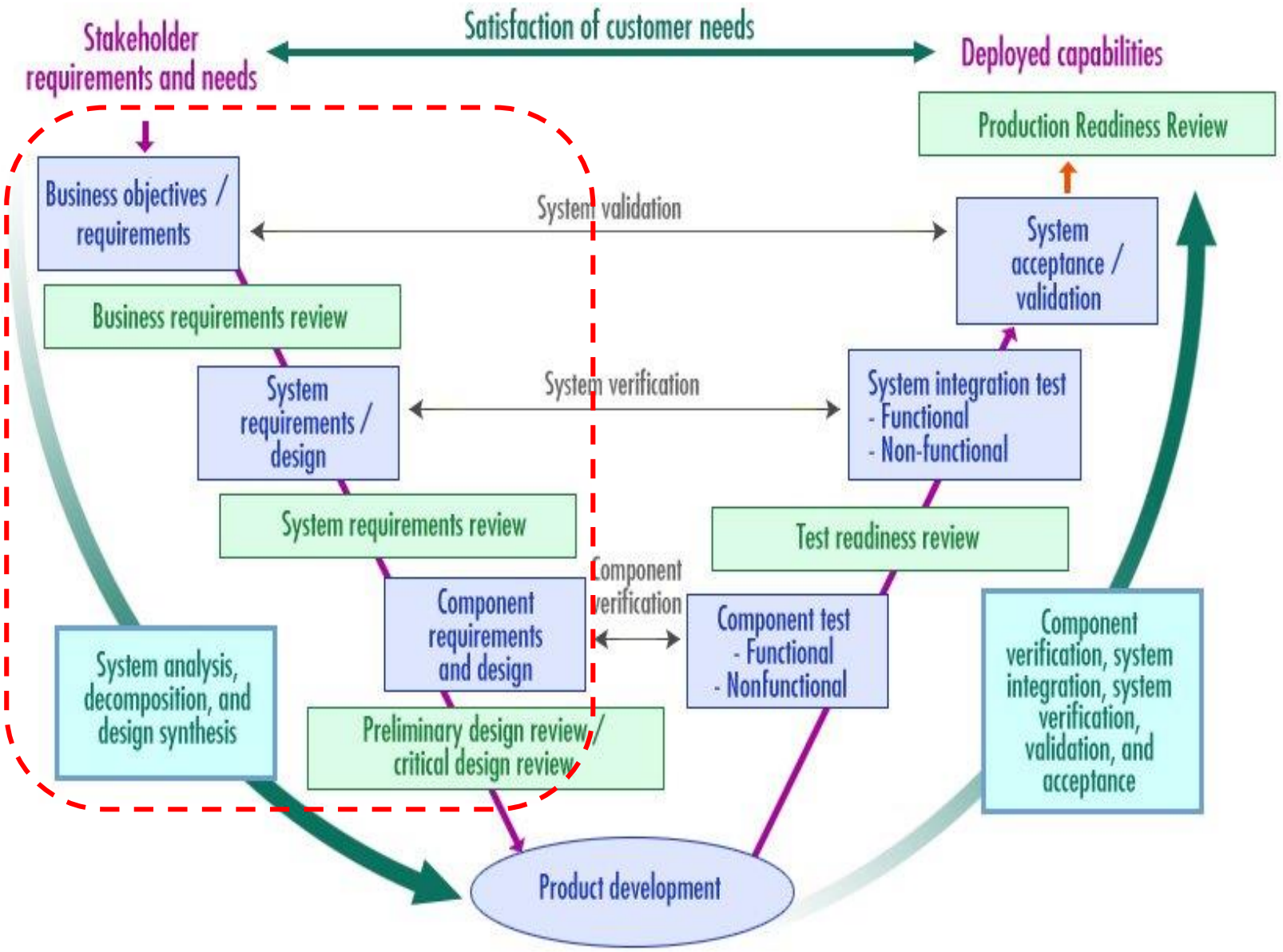
Key Questions: Given the scope and approach defined before, what is the solution and its architecture?

Key activities: Define requirements, solution architecture, schedule, resources, cost. Determine, feasibility and risks.

Key activities: Solution delivery (architecture, design, development, testing, hand-over to production)

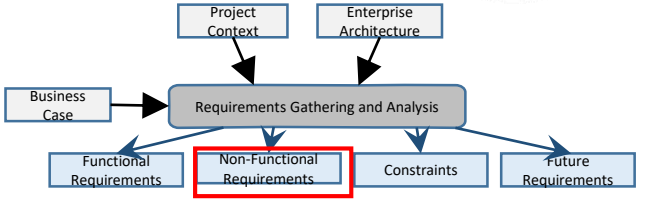
Functional Requirements definition – Level of details (elaboration)

- Customer wants and needs
- Business Requirements
- System Requirements



Module outline

- Introduction and Objectives
- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- System Context Diagram
- Functional Requirements
- **Non-functional Requirements**
- Constraints
- Future Requirements
- Managing Requirements
- Summary and References



Introducing Non-Functional Requirements and Constraints

Non-Functional Requirements define “how good” the solution should be.

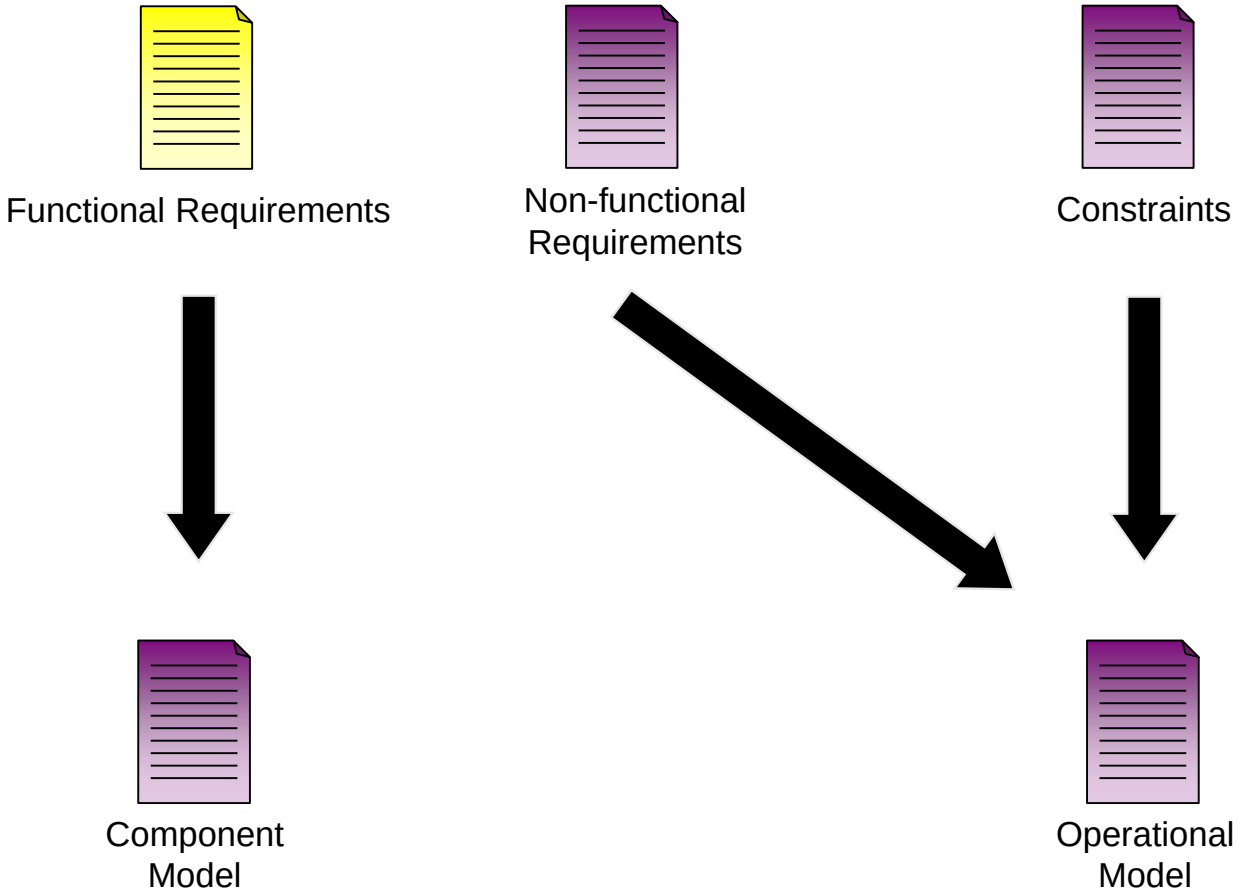
- The “adjectives” that can be applied to the “nouns and verbs” that describe the system’s functional requirements
- Some are associated with the running system:
  - How secure, how fast, how many users…
- Some are associated with other aspects of the system:
  - How portable, how scalable, how safe…

Constraints are limitations imposed upon a solution.

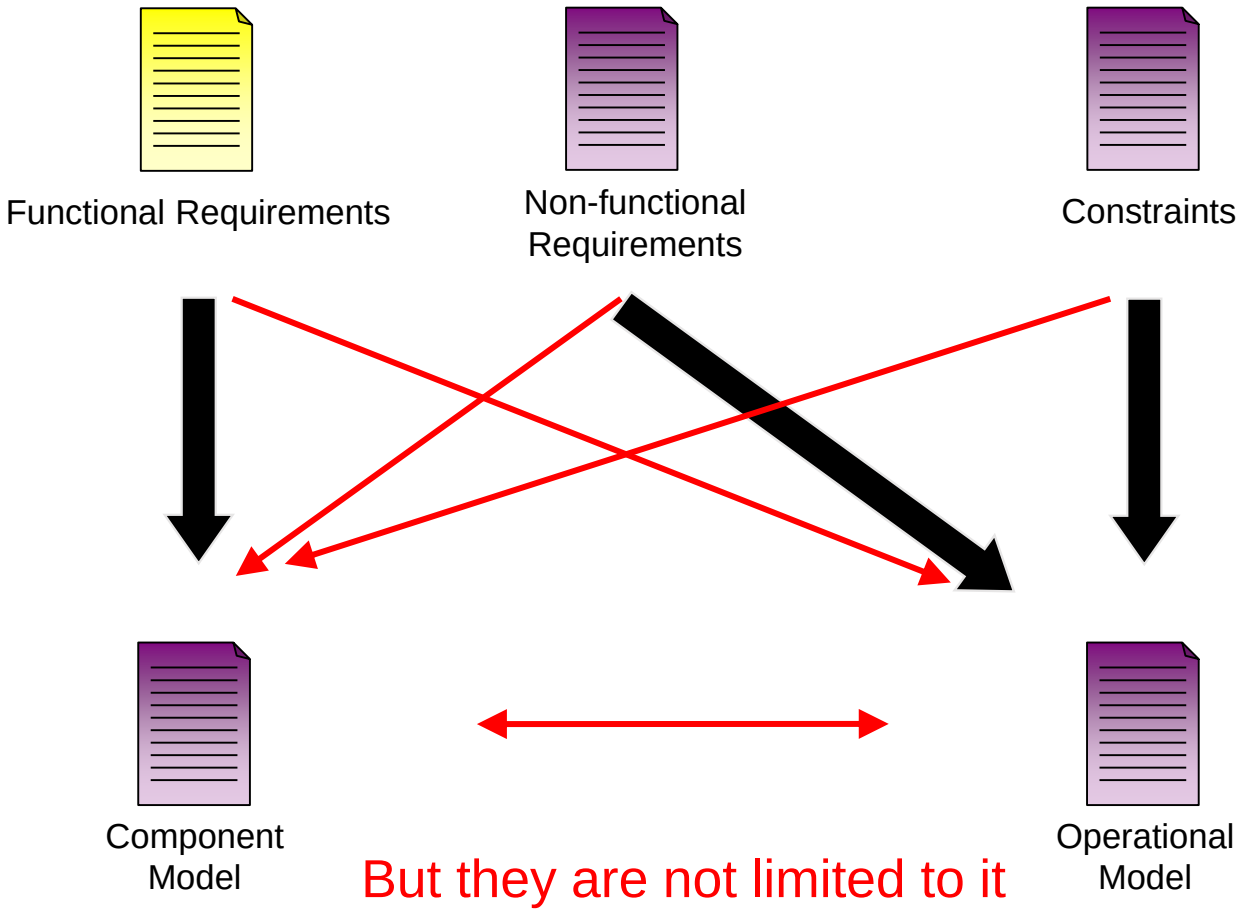
- They can be imposed by the business or from within IT.
  - “*It must be done this way.*”
  - “*It must conform to this legislation.*”
  - “*We only have skills in these areas.*”

*Definition taken from the IBM System Description Standard, V3

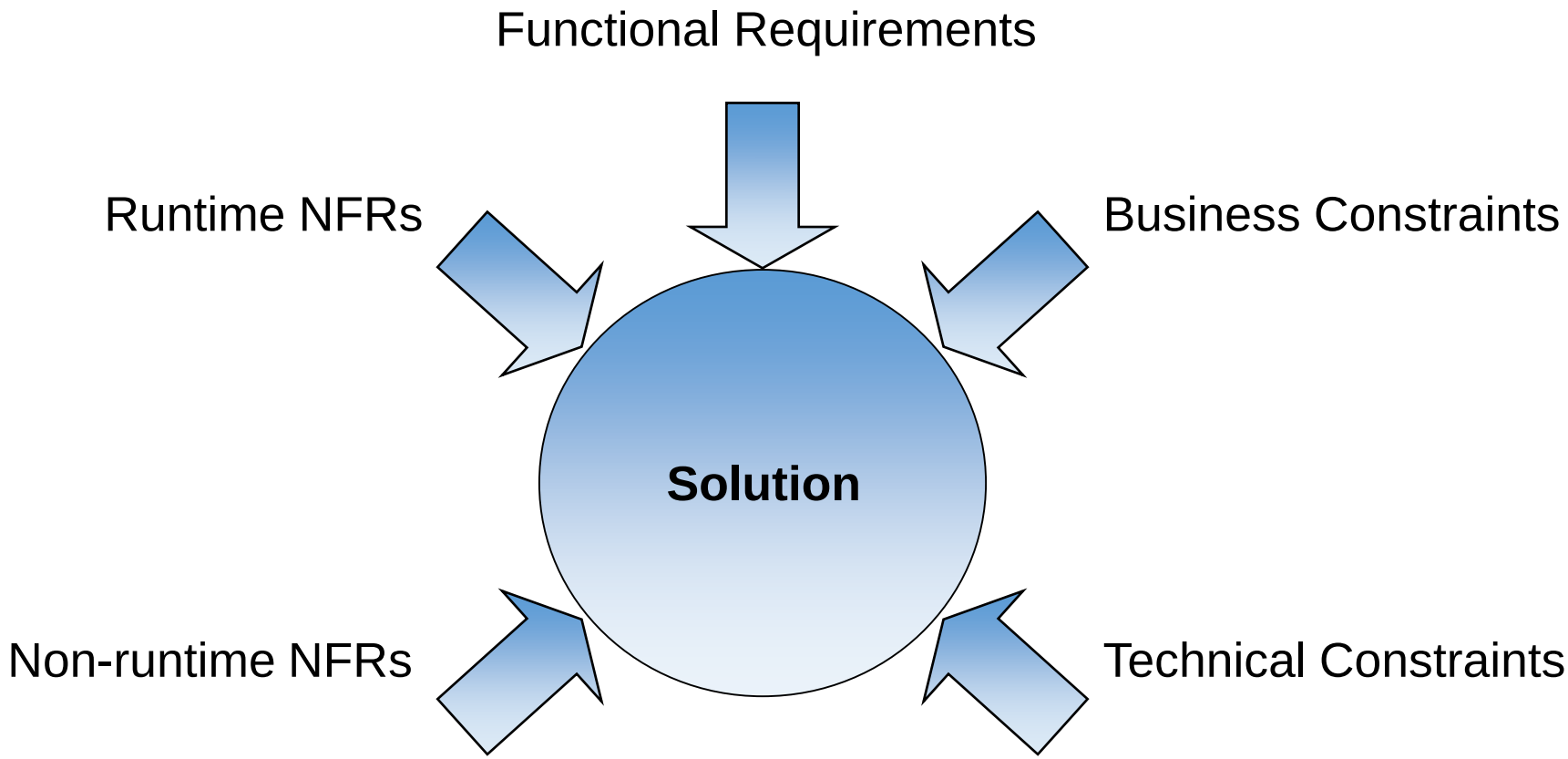
Non-Functional requirements are often thought to apply only to the operational viewpoint (1 of 2)



Non-functional requirements are often thought to apply only to the operational viewpoint (2 of 2)



Designing a solution to meet a number of NFRs involves resolving a number of opposing “forces”



Resolution of two or more opposing forces results in a solution to a problem.

NFRs define the “how good” expectations and characteristics of the target system

NFRs can be:

- Of value to the *user* of the system – called “*runtime* NFRs”
  - Can be empirically tested (e.g., by monitoring the system)
  - Sometimes stated as the system’s service level characteristics, and may be the basis of SLAs
  - Also known as the observable qualities

- Of value to the *operator* of the system – called “*non-runtime* NFRs”
  - Cannot be measured as easily, and may rely on other project metrics
  - Also known as the non-observable or development-time NFRs

What are example user (runtime) NFRs?

- Performance (of the system as it operates currently)
  - Does the system respond fast enough to user actions
  - Can the system support suitable numbers of concurrent users
  - Can the system support enough transactional throughput
- Volumetrics
  - Is the system capable of storing/ processing expected volumes of data
  - Is the system capable servicing expected number of users
- Security
  - Can the system identify and prevent unauthorized access?
- Usability
  - Can visually or hearing impaired people use our system
  - Do the system's user interfaces measure up to best practice usability benchmarks
- Availability
  - What is the maximum length of unscheduled outage the business can tolerate
- Manageability
  - How straightforward the system is to maintain, whether this is to ensure appropriate levels of functional or nonfunctional requirements

What are example operator (non-runtime) NFRs?

- Can it scale?
  - How easily the system can be expanded to accommodate growth in the future.
- Is it compliant?
  - Whether it conforms to the regulatory environment, e.g. privacy, SOX, Anti Money laundering, anti Terrorism Financing laws, international Sanctions
- Can it be maintained?
  - The ease with which the system can be changed, whether for bug fixes or to add new functionality
- Can it be managed?
  - The ease with which the system can be reused, deployed, and tested, started up, and shut down
- Is it environmentally sound?
  - Whether the system will harm users, including physical harm, as in medical systems, or financial harm, as in accounting systems, or other forms of harm
  - How well the system utilizes resources, and its effect on the environment
- Is it portable?
  - Moving the application to other operating systems, application containers, or different hardware
- Is it reliable?
  - The probability that a component will fail, and the nature of that failure

NFR Categories (1 of 2)

NFR Theme	Example or Description
Volumetrics: • **Static** • **Dynamic**	Volumetrics for the data entities that exist within the target system that, although relatively static, are likely to have a significant effect on the overall sizing of the target system E.g. Number of customers, accounts, policies, subscribers Throughput (txn per sec), response time
Reliability:	E.g.: 99.5%
Hours of Operation:	E.g.: 8:00 to 19:00
Security	E.g. Authentication, Authorisation, Confidentiality, Integrity, Non-repudiation of origin, delivery
Quality of Service	E.g. Guaranteed delivery (e.g. of transactions), idempotence (once only delivery)
Scheduling and Temporal	Time constraints imposed by business process (e.g. cut off times for transactions to be processed that need to be met). Schedule of events which drive processing (e.g. especially batch)
Monitoring and Instrumentation	Requirements for special facilities, functions, tools which are necessary to operate and manage the system

*IBM Unified Method Framework. NFR development, Technique Paper. See the Reference section.

NFR Categories (2 of 2)

NFR Theme	Example or Description
Logging and timestamping	What will be logged, and what will drive the timestamps to enable effective use of logs (e.g. a diagnostic log needs to have timestamps allowing re-creation of sequence of events)
Compliance and auditability	E.g. data that needs to be provided for audits that may be required in support of legal actions
Regulatory and legal	E.g. system functions and/or operational parameters needed to enable statutory compliance
Archiving, backup and recovery	What is the data history kept "active" and what can be archived? What is the data retention period for archives? How can the archived data be inspected? What data, how often backed up, retention period for the backup?
Disaster Recovery	Recovery Point Objective (RPO), Recovery Time Objective (RTO)
Supportability, Maintainability	Requirements for special facilities, functions, tools which are necessary to operate the system, and support its users
Localisation	Languages supported
Usability	Parameters defining how easy it is to use the system (e.g. length of training required)
Environmental	E.g. safety

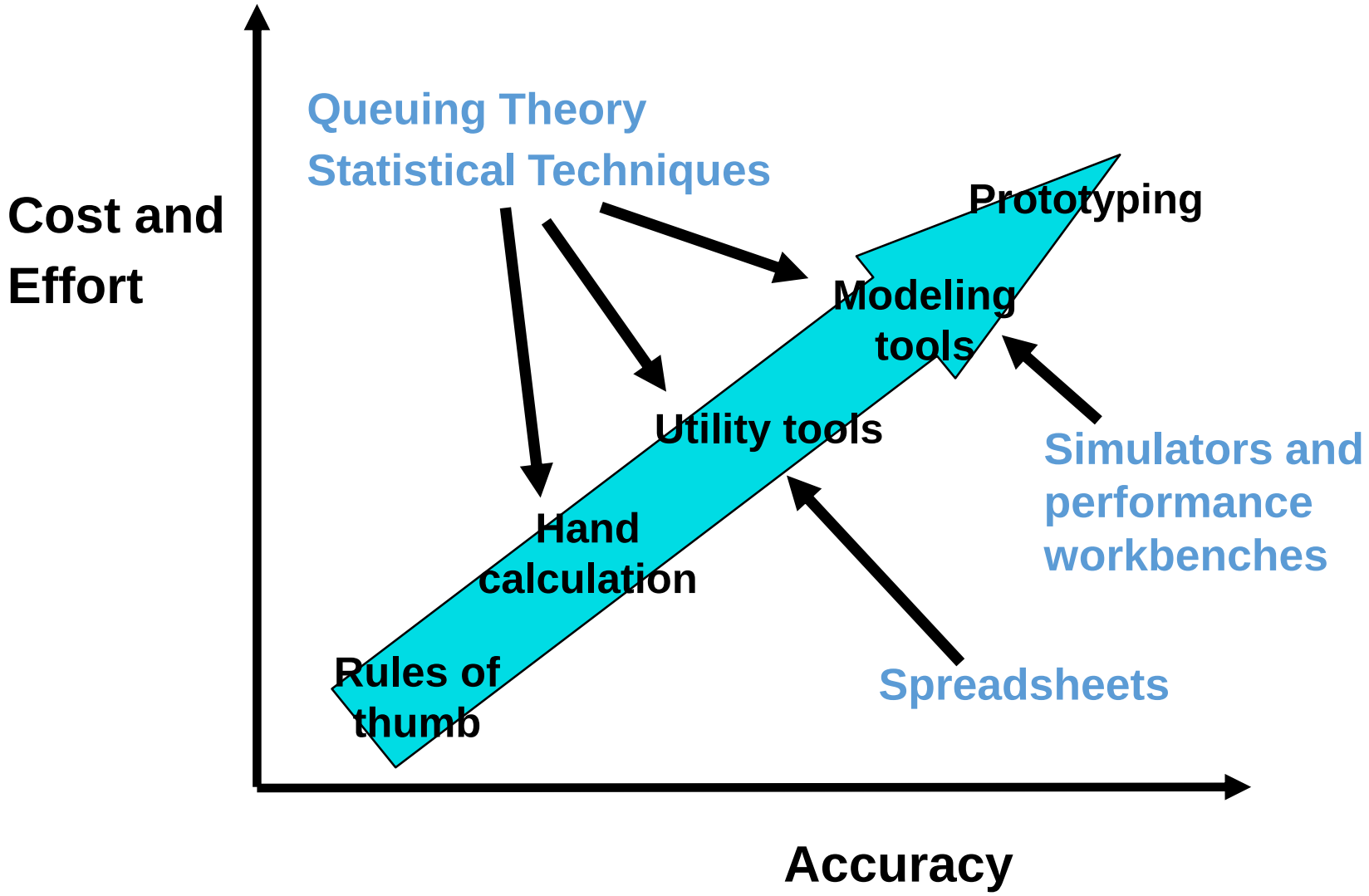
Performance (runtime)

- Definition:
  - "*The degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage*"
    - [IEE-610.12]
- In practical terms:
  - *How fast, how often, how big…*

- A simple performance statement, such as "*a response time of less than 1 second,*" is insufficient. It should always contain several components, such as:
  - What is the performance requirement (response time, transaction time)?
  - When is it to be delivered (in defined service hours)?
  - Where is it to be measured (on which systems, using which tools)?
  - For what workload is it to be valid (are test transactions available)?
  - For which users or systems (and how many)?
  - How can you define the percentile to achieve the given value (95th percentile, average)?

Striking a balance between cost, effort and accuracy

How much cost and effort is needed to ensure the system's performance is satisfactory? Just enough!



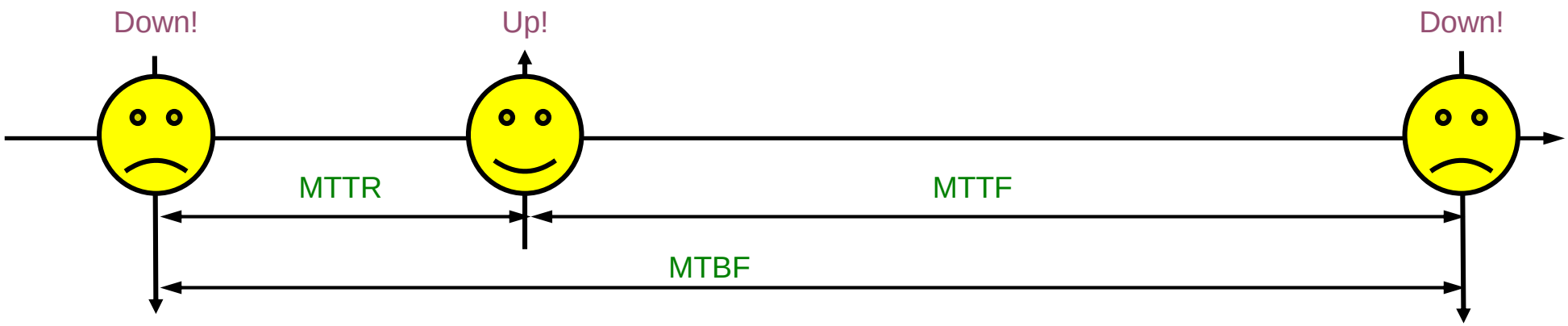
Availability (runtime)

- Definition:
  - "*A measure of readiness for usage, whenever, and for whatever purpose*"
- Systems may be unavailable.
  - At *scheduled times*, whether this is for maintenance, backup, or business reasons
  - At *unscheduled times*, when something breaks
- It is critical to distinguish:
  - **High availability (HA)**
    - System is *always* available *whenever it's expected to be.*
      - This may not be "24 x 7"
      - No unscheduled outages between defined hours on defined days of the year. *Scheduled outages permitted.*
  - **Continuous operations (CO)**
    - System is *intended* to be available *all the time:* "24 by 7."
    - Although unscheduled outages may occur, *no scheduled outages*
  - **Continuous availability (= HA + CO)**
    - System is *always* available *all the time.*

If HA is important, then CO may not be possible, since scheduled outages could be needed to "protect" HA.

	HA	CO	CA
Scheduled outages	Y	N	N
Unscheduled outages	N	Y	N

Availability: Factors affecting availability in an architecture



- Manufacturers and suppliers might be persuaded to quote MTTF…
  - Hardware reliability, software bug rates
- But MTTR is almost totally in our hands!
  - The IT Architect's solution design: Something might break "far away" or "near at home," with widespread or local impact.
  - Operations running the solution: When something breaks, time is taken in detection and action, and more time for recovery and restart procedures.
- MTTR includes:
  - Detect, report, diagnose, quiesce, back up, replace, restore/recover, test, reintegrate/resync, restart…
  - Many different people, skills, roles…
- And if MTTF is long, then failure does not happen often: Can everyone remember what to do?
  - If human intervention is needed, the longer the MTTF, the longer MTTR might be!

Whether we aim for HA, CO, or CA, systems will always "fail." How do we measure "unavailability"?

- Typically, availability is measured as a percentage uptime, *during scheduled hours.*
  - 95%, 99%, 99.9999%...

And don't confuse
CO ("24x7")
with
CA ("100% all the time")!

- How useful is this?
- All of these are "99%"…
  - 99 days "up," 1 day "down"
  - 99 hours "up," 1 hour "down"
  - 99 seconds "up," 1 second "down"

- The *business impact* of these is likely to be very different!

- We therefore need to understand:
  - Mean time to failure – The average time something is "up" ("99")
  - Mean time to repair – The average time something is "down" ("1")
  - Mean time between failures – The average time between "going down" ("100")

Security (runtime)

- Definition:
  - "*The ability to protect an IT system against malicious use while at the same time allowing legitimate use*"

- Security is sometimes referred to as:
  - Safety
    - "To reduce or eliminate danger, anxiety, and risk or liability"
  - Protection
    - "To defend against attacks (insider and outsider) and fraud (misuse of assets or misrepresentation of identity). To defend tangible assets (IT systems and applications or stored information or information in transit) and intangible assets (reputation)."
  - Assurance
    - To ensure correct and reliable operation; to enforce identity and ownership
    - To ensure the design complies with the security policy that the client already has in place.

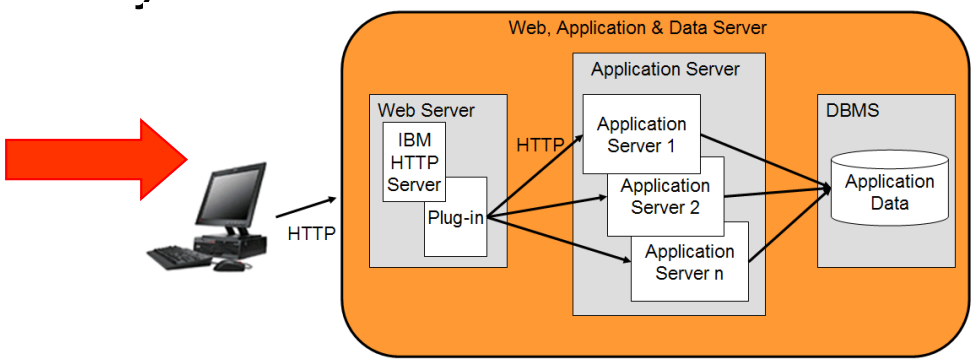
Security: Typical requirements and QoS in architectures

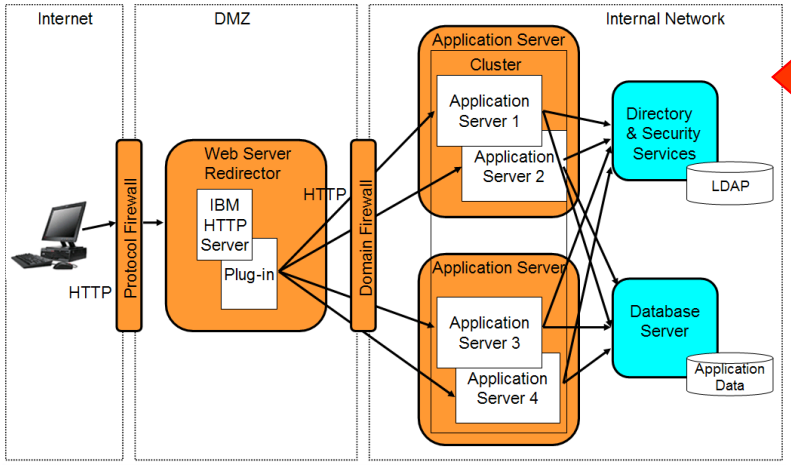
- Authentication
  - Validation that the party is who they claim they are
- Authorization
  - Validation of rights a user (or process) has to access information or to perform actions on resources (e.g. access to a function, right to update specific data)
- Integrity
  - Quality that ensures the immutability of data (e.g. protection form unauthorized update)
- Confidentiality
  - Protecting data from disclosure to unauthorized parties
- Non-repudiation
  - Strong evidence of authenticity of the sender (non-repudiation of Origin)
  - Strong evidence of delivery of message to the receiving party (non-repudiation of Delivery)

Scalability (nonruntime)

- Definition:
  - *A system's ability to increase (or decrease) in size according to changing workloads, such as an increase in user numbers*

- There are two approaches to scalability:
  - Vertical scaling ("scaling up")
    - Adding additional resources into the machine:
      - Adding more memory
      - Using more or faster processors





- Horizontal scaling ("scaling out")
  - Spreading the load across more machines
    - May require a "load balancer" or "redirector" to distribute requests between the computers

Question

Time for a question

Which important nonfunctional requirement is defined as “the ability to respond to increasing numbers of users by adding more resources?”

A. Availability

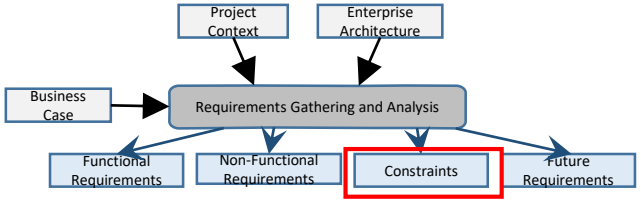
B. Maintainability

C. Performance

D. Scalability

E. Security

Module outline



- Introduction and Objectives
- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- System Context Diagram
- Functional Requirements
- Non-functional Requirements
- **Constraints**
- Future Requirements
- Managing Requirements
- Summary and References

Constraints are limitations imposed upon a solution

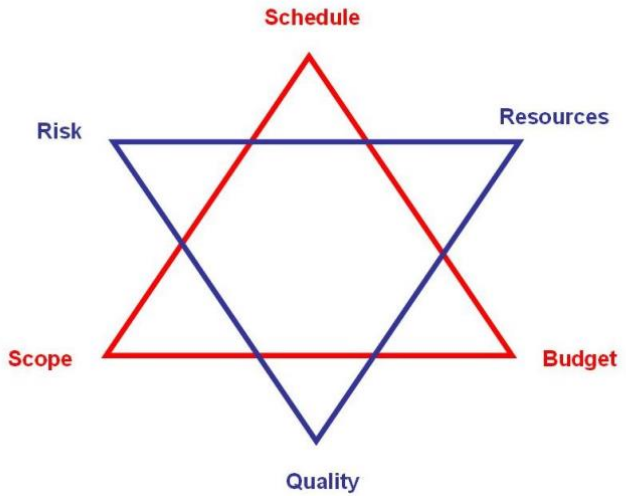
Constraints can be:

- Imposed by the business
  - Adherence to externally imposed business legislation (for example., data protection)
  - Driven by available business capabilities (for example, end user skills)

- Imposed from within IT
  - Use of standard parts (components, platforms, and so on) and approaches (patterns, reference architectures)
  - Reuse of existing systems; limitations on other systems' runtime NFRs

- Imposed by the project context
  - Be the project manager's best friend
    - Consider cost and budget implications
  - How long will it take?
    - Optimise resource
  - How much will it cost?
    - Is it built to order?
    - Is it built to cost?

"Triple Constraint" in Project Management



(*) See New Trends module for discussion on trade-off introduced by Agile approach

What are example business constraints?

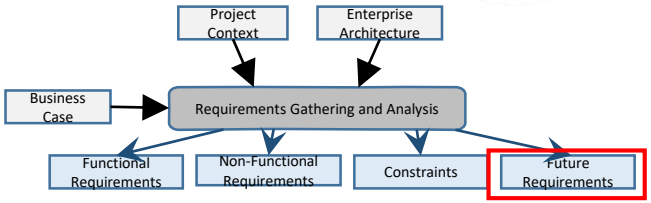
- Regulatory
  - Governmental, legislative, and other rules relating to the operation of IT systems or with which IT systems must comply
- Organizational
  - What is the organization that the system must be deployed into, and what special requirements does this introduce?
- Geographic (including localization)
  - What geographic regions will the system operate in, and what special circumstances do they dictate?
- Risk willingness
  - How willing is the client to take risk? (This affects, for example, how much new or unproven technology might be used on a project.)
- Marketplace factors
  - Such as competitor offerings, cost of goods or services in the domain this system operates in, and so on
- Project
  - Time - How long can it take to design, build, and deploy the new system?
  - Resource (people and budget) - What is available for the project?
  - Scope – What functionality must be built?

What are example IT constraints?

- Development skills
  - What skills are available for the development and support of the new system, in both the client and delivery organizations?
- Existing infrastructure
  - Is there any existing infrastructure (both hardware and software) that must form part of the new solution?
- Technology leadership (“state of the art”)
  - What new technology should or should not be considered? (See also “Risk Willingness” in business constraints.)
- IT architectural standards
  - Principles, building blocks, patterns, and reference models that will apply to the new system
- Implementation constraints
  - Code development (or package configuration) constraints such as:
    - Implementation languages
    - Policies for database integrity
    - Resource limits
    - Operation environments

Module outline

- Introduction and Objectives
- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- System Context Diagram
- Functional Requirements
- Non-functional Requirements
- Constraints
- **Future Requirements**
- Managing Requirements
- Summary and References



What are future requirements and what do they look like?

- Change cases:
  - Describe how the system might change in the future, whether these are functional or nonfunctional enhancements.
  - Conceptually similar to the Agile concept of "..product features which haven't been assigned to any release or a sprint as yet".

- Future changes that might affect the architecture include:
  - New and modified business processes and goals
  - Adaptation of the system to new technologies
  - Changes in the profiles of the average user
  - Changes in the integration needs with other systems

- Understand the importance of:
  - Potential benefits
  - Probability of becoming a reality
  - Customer's priorities

- Understand the impact (cost and resource commitment) if:
  - Change is accommodated now
  - Change is accommodated later

Module outline

- Introduction and Objectives
- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- System Context Diagram
- Functional Requirements
- Non-functional Requirements
- Constraints
- Future Requirements
- **Managing Requirements**
- Summary and References

Determine significant requirements that most likely affect the architecture

- Match requirements to purpose. The degree of detail, accuracy, and precision needed depends on the context:
  - Early pre-project sizing?
  - Proposal development?
  - Outline solution architecture?
  - Focused functional or operational design?

- Whatever the case, “architecturally significant” requirements are ones that:
  - “Touch” critical parts of the solution
  - “Touch” all parts of the solution, maybe from end to end or side to side
  - Are as likely to be non-functional as functional

Samples

High business payback functions
Critical business operations
High volume, high usage functions
Essential functionality of the system
Influential users
Mobile users
Have broad coverage and exercise many architectural elements
Security requirements
Complexity (such as complex business rules)
Stringent demands on the architecture (such as performance and availability requirements)
High potential for change (such as rules relating to taxation)
Communication and synchronization with external systems
Those which highlight identified issues/risks
Operational “Total cost of ownership” issues (such as conformance to an EA)
Information requirements

How not to gather requirements

Analyst: "*Does the product need to allow changes to orders?"*

Stakeholder: "*Oh, I'd not thought of that…. Yes, of course!*"

Analyst: "*Do we need to cater for many languages?*"

Stakeholder: "*That sounds good. We should plan to address foreign markets.*"

Analyst: "*And what about security?*"

Stakeholder: "*Oh yes, the product should be secure.*"

Analyst: "*Tell me about your reliability expectations.*"

Stakeholder: "*Reliability? What's that? Oh, I see.... Definitely 24x7 — no downtime. That's what we need to beat our competitors.*"

Analyst: "*How many customers will be using it?*"

Stakeholder: "*I've no idea, but I am sure you will cope.*"

Question

Time for a question

Nonfunctional requirements in the areas of usability and accessibility are most likely to come from which stakeholders?

A. Communicators

B. Developers

C. Testers

D. Users

Requirements are dynamic and require careful management throughout a project's life cycle

Reasons for change

- Users' needs and technology change as time passes
- As the system evolves, requirements become clearer
- External changes to the environment make requirements obsolete or create new requirements
- Change might be needed to manage requirement conflicts between different stakeholders

Change management

- Iterative documentation, review, and sign-off of requirements
- Well-defined and executed change management procedures
- Sufficient emphasis on developing change cases
- Ongoing evaluation of the application and the architecture

Requirements management tools play important role in this space.

Consider the following Gartner report:

Market Guide for Software Requirements Definition and Management Solutions
Published: 24 June 2016 ID: G00276075
Analyst(s): Thomas E. Murphy, Magnus Revang, Laurie F. Wurster
https://www.gartner.com/doc/reprints?id=1-3F1C5PR&ct=160817&st=sb

Module outline

- Introduction and Objectives
- Introducing Requirements
  - What Are Requirements?
  - Sourcing Requirements
- System Context Diagram
- Functional Requirements
- Non-functional Requirements
- Constraints
- Future Requirements
- Managing Requirements
- **Summary and References**

Summary

- Requirements are the foundation for managing scope, quality, and client expectations and have a profound impact on the architecture of a system, hence their relevance for Architects.
- There are many different sources of requirements and constraints, and these need careful balancing in order to arrive at a viable set acceptable to all stakeholders.
- An Architect working on an IT system will typically rely on the architecturally significant requirements to drive the initial definition of the architecture.
- A good requirement is clear, concise, traceable, design-independent, and verifiable. A good set of requirements is unique, complete, consistent, comparable, modifiable, and attainable.
- Use cases and process models are typically used to document functional requirements, whereas system context is used to capture project scope.

References

- Bass, Clements and Kazman. *Software Architecture in Practice*, ISBN 0-201-19930-0
- Boehm, Barry and In, Hoh. "Identifying Quality-Requirement Conflicts," *IEEE Software*, 2 March 1996, pp. 25-35
- Bredemeyer, Dana and Malan, Ruth. *Defining Non-Functional Requirements*, Bredemeyer Consulting, March 2001, http://www.bredemeyer.com
- Cockburn, Alistair. *Writing Effect Use Cases*. Addison-Wesley, 2001, ISBN: 0201702258
- Eeles, Peter. *Capturing Architectural Requirements*, The Rational Edge, November 2001, http://www.ibm.com/developerworks/rational/rationaledge
- Eeles, Peter and Cripps, Peter. *The Process of Software Architecting*. Addison-Wesley, 2008, ISBN: 0321357485
- Glib, Tom. *Software Engineering Management*, ISBN 0-201-19246-2
- Hope, Gregor and Woolf, Bobby. *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions*, Addison Wesley, 2004, ISBN: 0321200683
- IEEE Software Engineering Committee. *IEEE Recommended Practice for Software Requirements Specifications*, ISBN 1-55937-395-4
- Rozanski, Nick & Woods, Eoin. Software Systems Architecture – Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2005, ISBN: 0321112296
- Shaw and Garlan. *Software Architecture, Perspectives on an Emerging Discipline*, ISBN 0-13-182957-2
- Sommerville, Ian, and Sawyer, Pete. Requirements Engineering: A Good Practice Guide. Wiley and Sons.
- Whitmore, J.J. *A Method for Designing Secure Solutions*, IBM Systems Journal, Volume 40, Number 3, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5386938
- INCOSE Systems Engineering Handbook v3.2.1

THANK YOU